विश्वविद्यालय अनुदान आयोग
University Grants Commission
Quality higher education for all

जावा प्रोग्रामिंग

(बी.एस.सी कंप्यूटर साइंस, एम.एस.सी. कंप्यूटर साइंस, एम.सी.ए., बी. टेक. कंप्यूटर साइंस ,
बी. टेक. आई. टी. के छात्र छात्राओं के लिए)

डॉ अमित कुमार खासकलम
सूचना प्रोद्यौगिकी विभाग ,
गुरु घासीदास विश्वविद्यालय
बिलासपुर, छत्तीसगढ़

समर्पित

पूज्य बाबूजी

स्व. श्याम सेवक लाल जी खासकलम

को

आमुख

जावा एक हाई लेवल प्रोग्रामिंग लैंग्वेज है और हम सभी जावा की उपयोगिता से भलीभांति परिचित हैं । यह पुस्तक उन छात्र छात्राओं की आवश्यकताओं को ध्यान में रखकर लिखी जा रही है जो जावा सीखना चाहते है और अपनी रचनात्मकता का उपयोग

करना चाहते हैं जिससे वे नए नए समाधान समाज को दे पायें परन्तु अंग्रेजी भाषा उनके बीच में आती है। यहाँ पर दो बातें बहुत ही महत्वपूर्ण हैं एक तो जो भी छात्र छात्रा हिंदी माध्यम से स्कूल की पढ़ाई करते हैं उनकी जिज्ञासा एवं रचनात्मकता उन सभी छात्र छात्राओं से किसी भी मायने में कम नहीं है जो अग्रेजी माध्यम से पढ़ते हैं। दूसरी बात जो उससे भी महत्वपूर्ण है कि ज्यादातर छात्र छात्राओं को कोई भी विषय जब हिंदी में समझाया जाता है तो उनकी समझ में ज्यादा अच्छे से आता है और वे इसे बहुत ही अच्छे से उस ज्ञान को अमल में लाते हैं। इन्हीं सब बातों को ध्यान में रखकर यह पुस्तक आप के सामने है जिससे ज्यादा से ज्यादा छात्र छात्राएं जावा प्रोग्रामिंग सीख सकें और अपने भविष्य को उज्जवल बना सकें।

यह पुस्तक जावा प्रोग्रामिंग की है इसलिए जावा की सभी शब्दावली को जैसा का तैसा मतलब इंग्लिश में ही लिखा गया है जिससे जावा की शब्दावली भी न बदले और पुस्तक के उदाहरणों का उपयोग करके प्रायोगिक अभ्यास किया जा सके। इस पुस्तक को कुल 12 भागों में विभाजित किया गया है। पहले भाग में जावा के लिए सबसे आवश्यक ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (OOPs) की अवधारणा का विस्तृत वर्णन किया गया है क्योंकि ऑब्जेक्ट ओरिएंटेड अवधारणा के बिना जावा में प्रोग्राम लिखना संभव ही नहीं है।

जावा की बुनियादी परिभाषा एवं कैसे शुरू करें की जानकारी दूसरे भाग में विस्तृत रूप से दी गयी है।

इस पुस्तक के तीसरे भाग में जावा में पायी जाने वाली विभिन्न एक्सप्रेशन को उदाहरण सहित बताया गया है जिससे प्रोग्रामर अपने लॉजिक को विस्तार से उपयोग कर सके।

पुस्तक के चौथे भाग में जावा में उपलब्ध कंट्रोल स्ट्रक्चर की जानकारी दी गयी है जिससे विभिन्न स्थितियों में प्रोग्रामर अपने लॉजिक के हिसाब से प्रोग्रामिंग कर सके।

क्लास (Class) कीवर्ड का उपयोग करके एक टेम्पलेट बनाया जाता है जिसकी सहायता से ऑब्जेक्ट बनातें हैं। इस पांचवे भाग में Class और उसके अवयवों का वर्णन किया गया है

पुस्तक के छटवें एवं सातवें भाग में क्रमशः इनहेरिटेंस(Inheritance) और एक्सेस कंट्रोल (Access Control) का वर्णन किया गया है जिसे सीखकर प्रोग्रामर पहले से विकसित कोड का पुनः उपयोग अपने प्रोग्राम में कर सकता है और अपने समय की बचत के साथ साथ गलतियों की सम्भावना को भी कम करता है।

जावा प्रोग्राम में इनपुट आउटपुट के लिए बहुत सारे विकल्प उपलब्ध हैं उन्हें कैसे उपयोग किया जाये उसका उदाहरण सहित विस्तृत वर्णन पुस्तक के आठवें भाग में किया गया है ।

जावा में एक थ्रेड्स (Threads)का विकल्प भी है जिसके द्वारा कई कार्य को स्वतंत्र रूप से साथ साथ किया जा सकता है। थ्रेड्स (Threads)का उपयोग करने का विवरण भाग नौ में किया गया है।

जावा में एक और सुविधा जावा नेटिव इंटरफ़ेस (Java Native Interface JNI) के रूप में उपलब्ध है जिसकी सहायता से प्रोग्रामर किसी दूसरी लैंग्वेज (इस पुस्तक में C लैंग्वेज) में विकसित कोड को जावा में या जावा कोड के साथ चला सकता है। JNI का वर्णन पुस्तक के दसवें भाग में किया गया है।

पुस्तक के ग्यारहवें भाग में अपवाद (Exceptions) का वर्णन किया गया है जिसकी सहायता से प्रोग्रामर गलतियों को पकड़ने एवं एक्सेप्शन हैंडलिंग (Exceptions handling) की सहायता से उसका समाधान कर सकते हैं।

जावा में Ant एक निर्माण उपकरण है जिसकी सहायता से जावा एप्लीकेशन को कम्पाइल(Compile), पैक (pack) , डिप्लॉय (Deploy) एवं डॉक्यूमेंटेशन किया जाता है। इसका विवरण पुस्तक के बारहवें एवं अंतिम भाग में किया गया है।

यह पुस्तक ICTP, इटली में एक वर्कशॉप में मेरे इंस्ट्रक्टर रहे डॉ कार्लोस कावका (Carlos Kavka) के नोट्स पर आधारित है और इसके कुछ उदाहरणों को भी इस पुस्तक में विस्तृत रूप के उपयोग किया गया है। अतः लेखक कार्लोस कावका को विशेष रूप से धन्यवाद ज्ञापित करता है। इसी के साथ साथ ओलेक्सीय त्य्खोम्य्रोव (Olexiy Tykhomyrov) के नोट्स के सन्दर्भ का भी उपयोग ऑब्जेक्ट ओरिएंटेड की अवधारणा

में किया गया है। अतः लेखक ओलेक्सीय त्य्खोम्ग्रोव को भी धन्यवाद ज्ञापित करता है ।

इस पुस्तक को और उपयोगी बनाने के लिए विद्वतगणों के जो भी सुझाव प्राप्त होंगे हम उनके सहृदय आभारी रहेंगे।

आशा करते हैं कि यह पुस्तक न केवल छात्र छात्राओं के लिए बल्कि अपितु जावा प्रोग्रामर एवं अकादमिक व्यक्तिओं की आवश्यकताओं के अनुरूप सिद्ध होगी।

अमित कुमार खासकलम

स्वीकृति

लेखक अपने जीवन में आये सभी गुरुओं का आभारी है जिनके द्वरा प्रदान किये गए ज्ञान का ही परिणाम है कि यह पाठ्य पुस्तक इस स्वरुप में आप सभी के सामने प्रस्तुत है।

लेखक उन सभी असंख्य छात्र छात्राओं का भी आभारी है जिनको पिछले 26 वर्षों में कक्षाओं में , प्रयोगशाला में और प्रोजेक्ट के दौरान उनकी प्रतिदिन आने वाली समस्याओं से सीखा और यहाँ पर उद्धृत किया।

मैं अपने मार्गदर्शक स्वर्गीय प्रोफेसर राम कृपाल सिंह को भी धन्यवाद ज्ञापित करता हूँ कि उनके मार्गदर्शन काएक परिणाम इस पुस्तक के रूप में आप के सामने प्रस्तुत है।

प्रोफेसर आलोक कुमार चक्रवाल, कुलपति, गुरु घासीदास विश्वविद्यालय की दिन प्रतिदिन की प्रेरणा से मुझे प्रोत्साहन मिला और मैं अपने लिखने की प्रतिभा की ओर

अग्रसर हुआ और आपके सामने यह पुस्तक के रूप में प्रस्तुत है। इस अवसर पर मैं प्रोफेसर आलोक कुमार चक्रवाल जी को बहुत बहुत धन्यवाद ज्ञापित करता हूँ।

मैं इस शुभ अवसर पर में अपनी माताजी श्रीमती लक्ष्मी खासकलम, सासुमां श्रीमती रत्ना श्रीवास्तव एवं ससुरजी श्री शान्ति नारायण श्रीवास्तव का उनके सहयोग एवं व्यवहारिक ज्ञान के लिए बहुत बहुत धन्यवाद देता हूँ।

स्वीकृति के अंतिम पड़ाव में मैं अपनी पत्नी श्रीमती अनुऋता पुत्र यशस्विन और पुत्री अनिका का इस पूरी यात्रा में उनके धैर्य एवं सहयोग के लिए धन्यवाद देता हूँ।

अमित कुमार खासकलम

विषयसूची

इतिहास (History):

यह किताब इस तरह से लिखी गयी है कि आप को एक बेसिक OOPS जिसका मतबल है कि Object Oriented Programming और अब हम इस पुस्तक में इसे OOPS लिखेंगे। जावा कूल है, जावा फ्री है और जावा लगभग सभी हार्डवेयर प्लेटफार्म पर चल सकती है। यही कारण है कि जावा प्रोग्रामिंग के अलावा और बहुत कुछ बनाने की अनुमति देती है उदाहरण के लिए Graphical User Interface(GUI).

जावा एक OOPS लैंग्वेज है अतः जावा में प्रोग्राम लिखने से पहले हमें OOPS का बुनियादी ज्ञान होना आवश्यक है। यदि आप C++ में प्रोग्रामिंग करते है तो कुछ हो सकता है परन्तु बिना OOPS के ज्ञान के जावा में प्रोग्रामिंग नहीं हो सकती है क्योंकि जावा प्योर OOPS लैंग्वेज है।

जावा का एक सरल प्रोग्राम भी ऑब्जेक्ट ओरिएंटेड द्रष्टिकोण (Object Oriented approach) के बिना नहीं लिखा जा सकता है। आप हो सकता है कि बहुत अच्छे प्रोग्रामर हों परन्तु जावा प्रोग्राम के लिए आपको ऑब्जेक्ट का उपयोग करना ही होगा। यह सही कि जावा में कुछ विशेषताएं जटिल एवं परिष्कृत हैं जैसे मल्टी थ्रेडिंग। यह भी सत्य है कि जावा सीखना बहुत ही आसान है। जावा में प्रोग्राम लिखने के लिए आपको निम्न जानना आवश्यक है :

१. उत्पादक रूप से सीखना कि कैसे बहुत सारी class लाइब्रेरी जो दर्जनों classes और सैकड़ों मेथोड्स जावा डेवलपमेंट किट (JDK) के रूप में जावा में उपलब्ध हैं और जावा को परिशिष्ट प्रदान करते हैं।

२. कैसे प्रोग्राम को ऑब्जेक्ट ओरिएंटेड आदर्श में डिजाईन करना और कोड लिखना सीखना होगा।

उपर्युक्त चुनौतिओं में से पहली चुनौती क्रमिक रूप से प्राप्त किया जा सकता है। यहाँ यह आवश्यक नहीं है कि एक अच्छा प्रोग्राम बनने के लिए सभी को सारी class लाइब्रेरी को जाना जाए। जावा में उपलब्ध नए नए टूल्स और क्षमता को सीखने से आप एक शक्तिशाली प्रोग्राम और अधिक उत्पादन कर सकते हैं।

दूसरी चुनौती को क्रमिक रूप से प्राप्त नहीं किया जा सकता है क्योंकि एक सरल जावा प्रोग्राम को भी बिना OOPS के नहीं किया का सकता है।

फंक्शनल और OOPS आदर्श (Functional and OOPS Paradigm): प्रायः एक समस्या जिसके लिए प्रोग्राम लिखना है की शुरुआत पहले आमतौर पर abstraction level पर हम उपलब्ध संसाधनों की गणना करते हैं उसके बाद problem को decompose करते हैं और अंत में problem को आयोजित (organize)करते हैं उदाहरण के लिए यदि हमारे पास पेंट कम है तब हम कुछ चीजों को छोड़कर बाकी जगह में पेंट करते हैं इसमे किस किस वस्तु को छोड़ेंगे यह हमारी प्राथमिकता पर निर्भर करता है। यहाँ हम समाधान को आयोजित करते हैं। ये तीन तत्वों abstraction,decomposition और organization को एक साथ paradigm कहते हैं। पहले हमारे प्रोग्राममर्स का अनुभव फंक्शनल पैरादिग्म पर अधारित है।

इसे समझाने के लिए हम एक उदाहरण लेते है कि हमको कार से भोपाल जाना है यदि पुराने फंक्शनल पैरादिग्म के अनुसार हम, फंक्शन डाटा पर एक्ट करते हैं तो इसके हिसाब से योजना बनायेंगे। यहाँ पर फंक्शन डाटा पर एक्ट का मतलब है :

On abstraction level: इसमें हम समस्या के समाधान के लिए उस प्रक्रिया की योजना बनाते हैं जिसके द्वारा समाधान मिलेगा। उपर्युक्त उदाहरण के अनुसार कार कैसे चलाएंगे, कैसे यात्रा के लिए आवश्यक चीजें एकत्र करेंगे आदि।

On decomposition level: यहाँ पर हम प्रोसेसिंग को ऐसे छोटे छोटे भागों में तोड़ते हैं जिनका प्रबंधन आसानी से हो सके। उपर्युक्त उदाहरण के अनुसार कार को एक पॉइंट से दूसरे पॉइंट तक कैसे चलाएंगे। जैसे शहर में कैसे और हाईवे पर कैसे। कैसे और कहाँ हम कार रोकेंगे और शुरू करेंगे। इसके लिए हम फंक्शन बनाते हैं।

On organizational Level: इसके बाद हम बनाये हुए फंक्शन को नियम के हिसाब से स्थापित करते हैं। फंक्शन को कॉल करने, आर्गुमेंट को पास करने के अनुक्रम को स्थापित करते हैं। यह हमारी पुराणी व्यवस्था है जिसमें हम डाटा को उन मेथोड्स से अलग रखते हैं जिन मेथड का उपयोग हम डाटा को मेनिपुलेट करने के लिए करते हैं।

यह टिपिकल सोचने का तरीका नहीं है असली परिस्थिति में पहले रस्ते के बारे में सामान्य तौर पर विचार करते हैं उसके बाद हम ट्राफिक पर विचार करते हैं कैसे हम शहर में और कैसे हम गाँव में कार चलाएंगे। इसके बाद हम इस बात पर विस्तृत विवरण पर विचार करते है जैसे रूककर खाना खाने की लिए कौनसी जगत ठीक है, रात

में रुकने के लिए कौनसी जगह आदि। इसके बाद हम एक अनुसूची बनायेंगे जिसमें हम बताएँगे कि किस समय हम कहाँ होंगे। ऑब्जेक्ट ओरिएंटेड सोच का यह एक स्वरूप है। जहां पर ऑब्जेक्ट आपस में इंटरैक्ट करते हैं।

On abstraction level: यहाँ पर हम स्वतंत्र एजेंट्स (Objects) जो एक साथ काम करते हैं के रूप में सोचते हैं। जैसे कार, हम, चीजें जो लेकर जानी हैं आदि।

On decomposition level: यहाँ पर हम विभिन्न प्रकार के ऑब्जेक्ट्स पर विचार करते हैं जिनके अनुसार बड़े काम को अलग अलग करते हैं।

On organizational Level: यहाँ पर हम आवश्यकता अनुसार उपयुक्त संख्या में प्रत्येक ऑब्जेक्ट बनाते हैं।

ऑब्जेक्ट ओरिएंटेड वर्ल्ड (Object Oriented World):OOPS पैरादिग्म(paradigm) में प्रोग्रामर के द्रष्टिकोण से ऑब्जेक्ट ओरिएंटेड लैंग्वेज को आवश्यक रूप से तीन बहुत ही महत्वपूर्ण एवं स्पष्ट विशेषताओं की सहायता से प्रोग्रामिंग करनी चाहिए। इस अवधारणा को बड़े रूप में रियल वर्ल्ड प्रोब्लेम्स का मॉडल बनाने में उपयोग करते हैं। और ये तीनों अवधारणायें है :

- **एन्काप्सुलेसन (Encapsulation)**
- **इनहेरिटेंस(Inheritance)**
- **पोलिमोर्फिस्म(Polymophism)**

इसमें निहित विशेषता है abstraction. इससे हम नया abstract टाइप डाटा (ADT)को उल्लेखित करते हैं।

एन्काप्सुलेसन (Encapsulation): हमने यह तय किया है कि हम कार से भोपाल जा रहे हैं। Encapsulation को समझने के लिए हम कार के स्टीयरिंग को लेते हैं किसी भी कार में स्टीयरिंग समान रूप से कार्य करता है और इसका तंत्र क्या है यह Encapsulation में है अतः हमें स्टीयरिंग से कार को चलने के लिए यह जानना आवश्यक नहीं है कि स्टीयरिंग का तंत्र कैसे काम करता है। हम जानते हैं कि यदि हम स्टीयरिंग को घड़ी की दिशा में घुमाएंगे तो कार राईट तरफ मूढ़ जाएगी और यदि हम घड़ी की विपरीत दिशा में घुमाएंगे तो कार लेफ्ट तरफ मूढ़ जाएगी। यही OOPS पैरादिग्म (paradigm) में ऑब्जेक्ट कहलाता है। हम इसका कार्यान्वन जाने बिना ही उपयोग कर सकते हैं।

स्टीयरिंग तंत्र के ऑब्जेक्ट में और भी बहुत सारे विवरण अन्तर्निहित ऑब्जेक्ट्स होते हैं इनमें से प्रत्येक का अपना state, behaviour, interface और implementation होता है यह interface कार ड्राईवर को नहीं पता होता है पर स्टीयरिंग को कैसे चलाना है यह पता होता है।

OOPS में सामान्य कोशिश होती है कि implementation को छुपाया जाये और interface को Encapsulation के द्वारा अनावृत करते हैं।

इनहेरिटेंस(Inheritance): OOPS अगला पॉइंट है इनहेरिटेंस है। इसे हम एक उदाहरण से समझते हैं। यदि कोई व्यक्ति एक पुराना घर खरीदता है और इस परिस्थिति में सामान्तः पुराने घर को तोड़कर फिर से नया नहीं बनाते हैं बल्कि उसमें सुधार करके कुछ नया जोड़कर घर का उपयोग करता है। OOPS की शब्दावली के अनुसार व्यक्ति घर को पुराने घर से ही आवश्यकता अनुसार बनाता है। और दूसरे शब्दों में पुराने घर को एक्सटेंड (Extend) करता है या पुराने घर से इन्हेरिट करके नया घर प्राप्त करता है। OOPS का एक आदर्श है जो पहले से डेवलप्ड कोड है उनको पुनः नए सुधार के साथ उपयोग किया जाये।

हर बार नए कोड बनाने से अच्छा है कि पुराने या बने बनाये कोड को सुधार के साथ पुनः उपयोग किया जाये। इससे समय की बचत होती है और गलती होने की सम्भावना कम हो जाती है क्योंकि हम पहले से विकसित कोड जो पहले टेस्ट हो चुका है का उपयोग करते हैं।

पोलिमोर्फिस्म(Polymophism): polymorphism एक ग्रीक शब्द है इसका अर्थ है एक नाम और कई फॉर्म्स। और यही कारण है कि इसे समझना आसान नहीं है। इसे समझने के लिए हम एक आटोमेटिक कार का उदाहरण लेते है। आटोमेटिक कार में गियर के तीन स्टेज होते हैं ड्राइव , रिवर्स और न्यूट्रल। इसमें जब रिवर्स गियर लगाते हैं तो कार पीछे चलती है और जब न्यूट्रल गियर लगते हैं तो कार का इंजन डिसकनेक्ट हो जाता है। ड्राइव के कई version उपलब्ध हैं। और जब हम ड्राइव गियर लगते हैं तब कार की वर्तमान परिस्थिति के अनुसार आटोमेटिक ट्रांसमिशन सिस्टम मैकेनिज्म यह तय करते हैं कि ड्राइव का कौन सा version उपयोग होगा। स्पेसिफिक ड्राइव version अप्लाई होता है और यही Polymophism कहलाता है। अतः हम कह सकते हैं कि जब कार ड्राइव गियर की सहायता से आगे की ओर जाती है तब

Polymophism का पालन करती है और जब रिवर्स गियर से पीछे जाती है तब Polymophism नहीं दिखाती है।

Polymophism की सहायता से ऑब्जेक्ट के इंस्टांस के व्यवहार(behaviour) को अनुकूलित (customise) करते हैं।

हम ऑब्जेक्ट ओरिएंटेड वर्ल्ड में रहते हैं। ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग paradigm एक प्रयास है जिससे कंप्यूटर प्रोग्राम्स की सहायता से वर्ल्ड को समझा जा सके।

OOPS शब्दावली (OOPS vocabulary): प्री OOPS और OOPS की प्रोग्रामिंग एवं शब्दावली अलग अलग है उदाहरण के कौर पर OOPS में प्रक्रिया(Procedure) कुछ नहीं है यहाँ पर class, मेथड है। एक दूसरा उदाहरण है कि OOPS प्रोग्रामर abstract डाटा टाइप के implementation को Encapsulate करते हैं और इसके interface को class के अन्दर बना के, डिफाइन करते हैं।

एक class के एक से अधिक इंस्टांस बना सकते हैं। एक class के इंस्टांस को ऑब्जेक्ट कहते हैं। हर ऑब्जेक्ट का अवस्था (state) और व्यवहार(behaviour) होता है। एक ऑब्जेक्ट की state को इंस्टांस में स्टोर वर्तमान (current) वैल्यू से निकाल सकते है। और behaviour को class की उस इंस्टांस मेथड से निकाल सकते हैं जिससे से ऑब्जेक्ट को बनाया गया है। इनहेरिटेड abstract डाटा टाइप को बेस class या super class की class या subclass से व्युत्पन्न किया जा सकता है।

ऑब्जेक्ट (Object): OOPS में प्रोग्रामिंग या जावा सीखने से पहले हम ऑब्जेक्ट को समझते हैं।

पहला प्रश्न है ऑब्जेक्ट क्या है और इसका उपयोग कैसे होता है ?

इस प्रश्न का उत्तर बहुत ही उदारवादी सोच के अनुसार निम्न होगा :

ऑब्जेक्ट एक डाटा टाइप का इंस्टांस है।अब हम निम्न उदाहरण को देखते हैं जिसमें दो प्रकार के ऑब्जेक्ट को डिफाइन किया गया है। पहला ऑब्जेक्ट है जो कि integer का इंस्टांस है जिसका नाम happy है। कुछ कंप्यूटर के वैज्ञानिक इसे ऑब्जेक्ट नहीं मानते है पर हम उदारवादी सोच के अनुसार इसे ऑब्जेक्ट ही मानेगे।

दूसरा ऑब्जेक्ट हम abstract डाटा टाइप(ADT) का इंस्टांस बनाते हैं यहाँ हम abstract डाटा टाइप(ADT) Person का इंस्टांस बनाते है और उसका नाम man रखा।

.....

int happy; // यह integer का इंस्टांस है

Person man; // यह abstract डाटा टाइप(ADT) का इंस्टांस है

.....

इससे हम निष्कर्ष निकल सकते है कि पहला इंस्टांस integer टाइप का है जो जावा में उपलब्ध विभिन्न प्रकार के डाटा टाइप्स से आता है। ये डाटा टाइप्स जावा में डिफाइन हैं अतः इन्हें लैंग्वेज के आतंरिक डाटा टाइप्स कहते है और दूसरे शब्दों में primitive types.

यह माना जाता है कि abstract डाटा टाइप(ADT) जिसका नाम Person है वो आप ने बनाया है और कहीं डिक्लेअर कर दिया है जावा लैंग्वेज के द्रष्टिकोण से यह एक नया डाटा टाइप है।

सारग्रहण (Abstraction): यह abstract डाटा टाइप(ADT)का एक विनिर्दिष्ट है जिसे Abstraction कहते हैं। इसमें इस विशेष प्रकार के डाटा का प्रतिनिधित्व एवं व्यवहार शामिल है। यह दर्शाता है कि यह किस प्रकार का नया डाटा टाइप है और यह प्रोग्रामर को अपने अनुसार डिफाइन करने की सुविधा भी देता है। abstract डाटा टाइप(ADT) जावा लैंग्वेज की आतंरिक डाटा टाइप नहीं है अतः कम्पाइलर इसके बारे में कुछ नहीं जनता है। यह जिम्मेदारी प्रोग्रामर की है कि वो abstract डाटा टाइप(ADT)को उचित तरह से डिफाइन करे जिससे कम्पाइलर इसे समझे और कम्पाइल करे

जावा प्रोग्रामर इस नए डाटा का प्रतिनिधित्व एवं व्यवहार डिफाइन class कीवर्ड की सहायता से करते हैं जिसे कम्पाइलर को प्रस्तुत किया जा सके। इसको हम ऐसे भी समझ सकते है कि कीवर्ड class का उपयोग डाटा प्रतिनिधित्व एवं व्यवहार को इस तरह डिफाइन करे कि कम्पाइलर समझ सके और इसे कम्पाइल कर सके।

एक बार जब नया डाटा टाइप डिफाइन हो गया है तब इस डाटा टाइप के एक या उससे अधिक ऑब्जेक्ट बनाये जा सकते हैं जिससे इस डाटा टाइप को Abstraction से

असलियत में लाया जा सके। दूसरे शब्दों में इस तरह के ऑब्जेक्ट का इंस्टांस बना सकते हैं।

Abstraction से जब इंस्टांस बन गया तब ऑब्जेक्ट में अवस्था (state) और व्यवहार (behaviour) आ गए। किसी ऑब्जेक्ट की अवस्था (state) को मौजूदा वैल्यू से प्राप्त कर सकते है और इस ऑब्जेक्ट के व्यवहार (behaviour) को उसमें उपलब्ध मेथड से प्राप्त कर सकते हैं।

इसे समझने के लिए हम एक उदाहरण GUI के तत्वों में से एक button को लेते हैं। यदि हम button को एक ऑब्जेक्ट की तरह मानते हैं तब ऑब्जेक्ट के अवस्था (state) और व्यवहार (behaviour) को समझाना आसान होगा। इस button की विभिन्न स्टेट हो सकती हैं जैसे साइज़ ,पोजीशन और कैप्शन आदि। इसकी हर स्टेट को button के इंस्टांस वेरिएबल में स्टोर डाटा से प्राप्त कर सकते हैं। एक या उससे अधिक इंस्टांस वेरिएबल का संयोजन उस ऑब्जेक्ट की विशिष्ट स्टेट को ऑब्जेक्ट की प्रॉपर्टी (Property) कहते हैं

क्लास की परिभाषा (Class Definition): क्लास को समझने के लिए यहाँ पर हम human के द्वारा मॉडलिंग के लिए एक प्रोग्राम लिखने का प्रयास करेंगें। इसमें Human class जो दो सामान्य स्टेट्स working और resting का वर्णन करती है। class को यह बताना है कि वर्तमान स्टेट : काम करने के लिए उपलब्ध और आराम की स्थिति में काम के लिए उपलब्ध नहीं है। एक इंस्टांस वेरिएबल के लिए हम एक integer वैल्यू tired, और दो स्ट्रिंग name एवं origin को लेते हैं।

यहाँ हम केवल स्ट्रक्चर बना रहे हैं विस्तृत कोड नहीं लिख रहे हैं

```
class Human {
// class code
}
```

इस परिभाषा में class कीवर्ड है और इसका नाम Human है। इस नए टाइप के व्यवहार को परिभाषित करने के लिए तीन इंस्टेंट मेथोड्स का उपयोग करेंगे। पहली मेथड होगी setPerson(), इस मेथोड्स का उपयोग करके इस नए टाइप के ऑब्जेक्ट में डाटा स्टोर करेंगे। दूसरी मेथड है getHumanInfo() जिसका उपयोग करके हम पहले से स्टोर डाटा को प्राप्त कर सकते हैं। तीसरी मेथड है work यह मेथड

का उपयोग करके class Human क ऑब्जेक्ट tired को बदल सकते हैं। इसे निम्न प्रकार से लिख सकते हैं

```
 class Human {
// class code
// instance method to store data
Voide serPerson (int state, String name,String orig){
…
}

// instance method to display info of Human
String getHumanInfo(){
……..
}
// inatance method Work
String Work(){
…….
}
```

इसमें आवश्यकता के अनुसार और मेथोड्स का उपयोग करके ऑब्जेक्ट के व्यवहार को बढाया जा सकता है। इस नए टाइप के डाटा को परिभाषित करने के बाद इसके ऑब्जेक्ट बनाये जा सकते है और इसे primitive टाइप के डाटा की तरह उपयोग कर सकते हैं।

जावा एक परिचय (JAVA Introduction):

जावा एक हाई लेवल प्रोग्रामिंग लैंग्वेज है जिसे जेम्स गोसलिंग ने सन माईक्रोसिस्टम में रहकर सन 1995 में डेवलप किया था। ओरेकल कारपोरेशन ने बाद में इसे अधिकृत कर लिया। जावा एक सरल, क्लास पर अधारित प्रोग्रामिंग लैंग्वेज है। जावा को इस तरह से डिजाईन किया गया है कि इसका इम्प्लीमेंटेशन में कम से कम निर्भरता रहे और यह ऑब्जेक्ट ओरिएंटेड अवधारणा पर काम करे। इस प्रोग्रामिंग लैंग्वेज को बनाने का एक प्रयोजन यह भी है कि प्रोग्राम को डेवलपर एक बार लिखे एवं ये प्रोग्राम कहीं भी रन हो जाये (write once, and run anywhere (WORA)). इसका अभिप्राय है कि जावा कोड एक बार कम्पाइल होने के बाद बना कम्पायेल्ड कोड किसी भी प्लेटफ़ॉर्म

पर बिना कम्पाइल किये रन हो जाये , इसे ही प्लेटफार्म इंडिपेंडेंट कहते हैं। आज के समय में जावा व्यापक रूप से एप्लीकेशन डेवेलप करने में उपयोग कि जा रही है उदाहरण के लिए मोबाइल डिवाइसेस, गेम्स, वेब सर्वर्स, अप्लिकेसन सर्वर्स डेटाबेस कनेक्शंस आदि। जावा सादगी, मजबूती, सिक्यूरिटी विशेषताओं एवं पुनर्प्रयोग के कारन ही एंटरप्राइज लेवल एप्लीकेशन के लिए लोकप्रिय पसंद बन गयी है। जावा विभिन्न प्लेटफॉर्म्स जैसे Windows, Mac, Linux, Raspberry Pi, etc पर काम करती है। यह एक ओपन सोर्स , फ्री सिक्योर, फास्ट,पावरफुल , सिखने में आसान एवं उपयोग करने में सरल प्रोग्रामिंग लैंग्वेज है। जावा एक ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (OOPs) लैंग्वेज है जो प्रोग्राम को स्पस्ट संरचना एवं पुनर्प्रयोग प्रदान कर विकास की लागत को कम करती है। जावा C++ एवं C# से मिलती जुलती है जिससे प्रोग्रामर्स आसानी से कोड इस लैंग्वेजेस में परिवर्तित कर लेते हैं। जावा में प्रोग्रामिंग करने में सहायता के लिए जावा कम्युनिटी बहुत बड़ी है।

जावा प्लेटफार्म (Java Platform):

अब हम जावा को विंडो में इनस्टॉल करने की विधि को समझेंगे।

जो भी जावा को विंडोज में इनस्टॉल करना चाहते हैं उन्हें निम्न प्रक्रिया का पालन करना होगा

1. सबसे पहले निम्न लिंक को ब्राउज़र में टाइप करें .
   https://www.java.com/en/download/manual.jsp
2. इस वेब पेज पर windows online पर क्लिक करें। इससे जावा की एक्सिक्युटेबल फाइल (example: jre-8u371-windows-i586-iftw.exe) डिफॉल्ट डाउनलोड लोकेशन पर डाउनलोड हो जाएगी।
3. अब आप उपर्युक्त एक्सिक्युटेबल फाइल को अपने कंप्यूटर सिस्टम में किसी भी लोकेशन
   पर सेव कर सकते हैं।
4. अब इंस्टालर को रन करने के लिए एक्सिक्युटेबल फाइल पर दो बार क्लिक करना है
   जिससे यूजर कंट्रोल डायलॉग बॉक्स खुल जायेगा और आप से आप के कंप्यूटर पर होने
   वाले चेंज की अनुमति ली जाएगी।
5. यहाँ पर आपको yes पर क्लिक करना है।

6. अब इंस्टालेशन शुरू होगा और जावा सेटअप का वेल्कम डायलॉग बॉक्स आएगा। अब

install बटन पर क्लिक करके license term को स्वीकार करने पर इंस्टालेशन

सुरु हो जायेगा।

इंस्टालेशन कम्पलीट होने के बाद जावा सेटअप कम्पलीट डायलॉग बॉक्स आएगा अब हमें close पर क्लिक करना है।

जावा प्रोग्रामिंग का प्रथम उदाहरण (First Example of Java):

अब हम Hello World के साधारण उदाहरण से शुरू करते हैं

```
 //My first code of Java

/** Hello World Application
* My first program
*/

class HelloWorld {
public static void main (String[] args) {
System.out.println("Hello World !");
}
}
```

उपर्युक्त एप्लीकेशन में बहुत सारे अवधारणाएं शामिल हैं। हम एक के बाद एक सभी को इस पुस्तक में समझेंगे। इस समय मुख्य पहलू पर विचार करते हैं।
इस एप्लीकेशन में केवल एक class है जिसका नाम Hello World है। यहाँ ध्यान रखना है कि इस फाइल को HelloWorld.java के नाम से ही सेव करना है। इसका मतलब है कि जावा में फाइल का नाम एवं class का नाम बिना एक्सटेंसन(.java) के हमेशा एक ही होगा।
केवल एक ही class HelloWorld में केवल एक ही मेथड main() ही डिफाइन है। main() मेथड को निम्नुसार ही डिफाइन करना है

```
public static void main (String[] args)
```

यह मेथड main() arguments के रूप स्ट्रिंग्स का array प्राप्त करती है एवं कुछ भी रिटर्न नहीं करती। यही वह मेथड है जहां से प्रोग्राम का एक्सिक्युसन शुरू होता है।

```
System.out.println("Hello World !");
```

उपर्युक्त स्टेटमेंट में System class को Java Application programming Interface (Java API) में डिफाइन किया गया है। यह System class सिस्टम की कार्यक्षमता (system functionality) तक पहुँच बनता है। इसमें class

System का वेरिएबल out एक मेम्बर है जो स्टैण्डर्ड output स्ट्रीम प्रदान करता है। उपर्युक्त स्टेटमेंट में एक मेथड println( ) भी है जो arguments से प्राप्त स्ट्रिंग को प्रिंट करने के लिए उपयोग में आती है।

जावा में दो प्रकार के कमेंट्स हैं जो कि डाक्यूमेंट्स के लिए उपयोग में आते हैं। ये कमेंट्स कंपाइलेशन के समय छोड़ दिए जाते है ये सिर्फ प्रोग्रामर की जानकारी के लिए उपयोग में आते हैं। एक लाइन के कमेंट के लिए // का उपयोग करते हैं। एक से ज्यादा लाइन के कमेंट को निम्नुसार लिखा जाता है

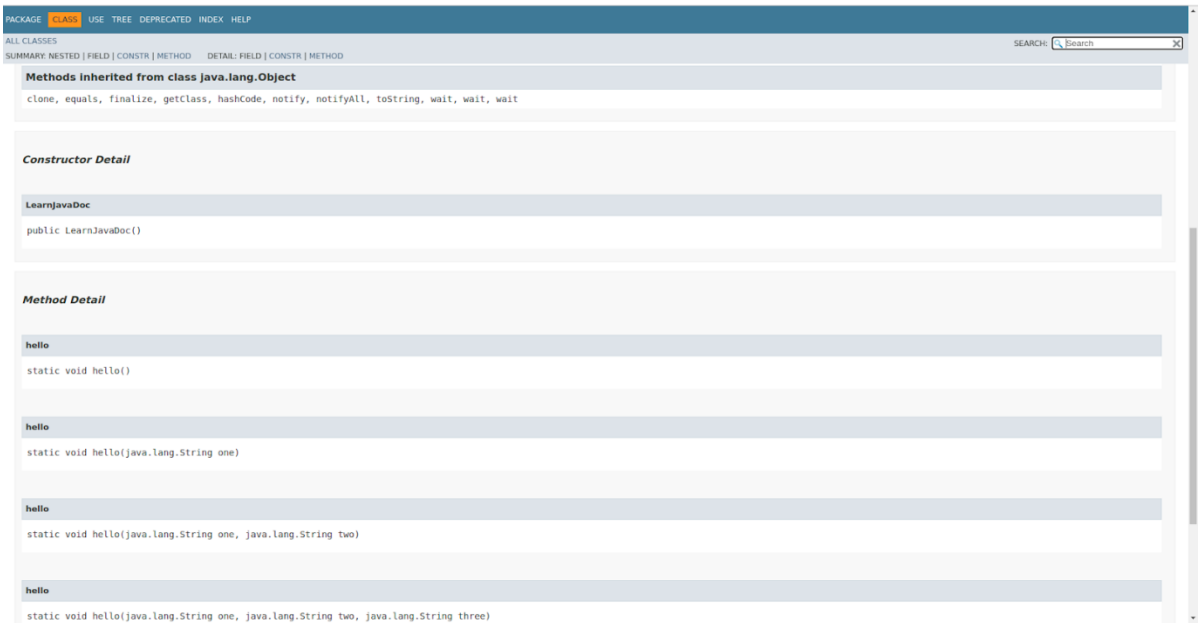
/* ..कमेंट्स .. */

कमेंट्स के लिए जावा में एक और विकल्प है जिसे एक विशेष प्रकार के डॉक्यूमेंटेशन की सुविधा javadoc के द्वारा करते समय इसका उपयोग निम्न प्रकार से करते हैं

/** ...कमेंट्स... */

इसका उपयोग प्रथम उदाहरण में भी किया है।

यह तीसरे प्रकार का कमेंट की व्याख्या javadoc द्वारा आटोमेटिक डॉक्यूमेंटेशन जनरेशन के समय करते हैं। javadoc के द्वारा डॉक्यूमेंटेशन को वेब ब्राउज़र पर देखा जा सकता है। कमेंट की इस केटेगरी में HTML एंड स्पेशल कमांड्स के लिए उपयोग होता है। javadoc में डॉक्यूमेंट निम्न प्रकार से दिखता है इस चित्र में constructor का डिटेल दिखाया गया है।



जावा एप्लीकेशन डेवलपमेंट साइकिल(Development Cycle for Java Application): जावा एप्लीकेशन डेवलपमेंट में मुख्य तीन कार्य करना होता है । सोर्स फाइल बनाना , सोर्स फाइल को कम्पाइल कर एक्सिक्युटेबल फाइल बनाना एवं एक्सिक्युटेबल फाइल को रन करना ।

सोर्स फाइल बनाना (Creation of source file): कोई भी टेक्स्ट एडिटर में प्रोग्राम कोड लिखकर फाइल को `.java` का एक्सटेंसन लगाकर सेव करते हैं यहाँ पर ध्यान देना होगा कि `class` का नाम एवं फाइल का नाम एक ही हो ।

```
# Notepad HelloWorld.java
```

सोर्स फाइल को कम्पाइल करना (Compilation of source file): सोर्स फाइल को जावा कम्पाइलर के द्वारा कम्पाइल किया जाता है जिसमें सोर्स कोड बाइट कोड में बदल जाता है और यह बाइट कोड जावा वर्चुअल मशीन (JVM) की सहायता से एक्सिक्युट किया जाता है । इसका सिंटेक्स

```
Javac HelloWorld.java
```

इससे बनने वाली फाइल का एक्सटेंसन `.class` होता है ।

`HelloWorld.java`	सोर्स फाइल
`HelloWorld.class`	काम्पायेल्ड फाइल (बाइट कोड)

एक्सिक्युसन (रन) करना (Execution):

इस प्रकार से बनी एप्लीकेशन की व्याख्या JVM द्वारा की जाती है। एप्लीकेशन को एक्सिक्युट करने के लिए class के नाम (बिना एक्सटेंसन के) को arguments जैसे प्रदान करके कॉल करते हैं। उदाहरण के लिए

```
java HelloWorld
```

प्रथम एप्लीकेशन का रिजल्ट

```
Hello World
```

Javadoc उपयोगिता (Application of Javadoc):

आटोमेटिक डॉक्यूमेंटेशन बनाने की एक विधि जावा में है और वो है javadoc । इसका उपयोग class और उसके सभी कंपोनेंट्स का डॉक्यूमेंटेशन बनाने में किया जाता है, विशेषतः इसमें con
HTML फाइल बनायीं जाती है इसका सिंटेक्स निम्न है

```
javadoc HelloWorld
```

इस स्टेटमेंट से HelloWorld.html फाइल बनती है

.बेसिक टाइप (Basic type):

जावा में दस प्रिमिटिव टाइप्स हैं जिनमे चार टाइप के integers, दो प्रकार के floating point numbers, characters, Boolean, the special type void and strings हैं।

निम्न तालिका में कुछ के मिनिमम एवं मैक्सिमम वैल्यूज दर्शायी गयी हैं

S.No.	Type	Size	Minimum value	Maximum value
1	Byte	8 bits	-128	127
2	Short	16 bits	-2^{15}	$2^{15} - 1$
3	Int	32 bits	-2^{31}	$2^{31} - 1$
4	long	64 bits	-2^{63}	$2^{63} - 1$

5	Float	32 bits	1.4E – 45	3.45E38
6	Double	64 bits	4.9E - 324	1.7E308
7	Char	16 bits	Unicode 0	Unicode 2^{16} - 1

उपर्युक्त तालिका के अतिरिक्त boolean टाइप जिसमें दो वैल्यूज हैं एक true और false. जब भी हम मेथड डिफाइन करते हैं तब वो मेथड क्या रिटर्न करेगी यह भी डिफाइन करना होता है। void टाइप, मेथड के लिए उपयोग होता है। void मेथड का मतलब है कि यह मेथड कुछ भी रिटर्न नहीं करेगी उदाहरण के लिए main() मेथड जो हमने पहले उदाहरण में डिफाइन किया है।

```
public static void main (String [] args )
```

स्ट्रिंग टाइप में केरेक्टर्स का ऐरे होता है और यह C लैंग्वेज के array से अलग है। टाइप character 16 बिट्स का होता है और यह ASCII character के स्टैण्डर्ड सेट के साथ कार्य करने की अनुमति देता है।

वैरिएबल्स (Variables): वैरिएबल्स को डिक्लेअर करने के लिए जावा में वैरिएबल का नाम एवं टाइप बताना होता है।
वैरिएबल्स को इनिशियलाइज़ या तो डिक्लेअर करते समय या फिर बाद में भी कर सकते हैं। उदाहरण के लिए

```
int x;
x = 5;
double f = 0.24;
char c = 'b';
string s = "akud"
```

लिटरल्स (Literals) :

डेसीमल , हेक्साडेसीमल , ऑक्टल और integer के लॉन्ग फॉर्म्स में वैल्यूज लिखने का तरीका

```
int x = 56;                    // डेसीमल वैल्यू
```

```
int y = 0x3bd;                  // हेक्साडेसीमल वैल्यू
int z = 0755;                   // ऑक्टल वैल्यू
long a = 534261773L;            // long इन्टिजर वैल्यू
```

floating point में वैल्यूज डिफॉल्ट में double टाइप की होती हैं और float टाइप बनाने के लिए F का उपयोग अंत में करते हैं जैसा कि निम्न स्टेटमेंट्स में देखा जा सकता है

```
double s = 3.65;        // 3.65  double टाइप में है
 float c = 3.65F;       // 3.65  float टाइप में बदला है
```

केरेक्टर टाइप वैल्यूज को निम्न प्रकार से लिख सकते हैं

```
char s= 'c';            // c करेक्टर टाइप है
char a = '\n'           // नयी लाइन करेक्टर
char x = '\u3456'       // यूनिकोड करेक्टर
```

यूनिकोड वैल्यू को ¥u से लिखते हैं।

Boolean टाइप की दो वैल्यूज true और false को boolean वेरिएबल में लिखने का तरीका

```
boolean name = true;
 boolean ready = false;
```

कांस्टेंट (Constant) :

कांस्टेंट का डिक्लेरेशन वेरिएबल के पहले final शब्द लगाकर करते हैं। और दूसरे तरीके से पहले कांस्टेंट डिक्लेअर करते हैं और बाद में वैल्यू देते हैं क्योंकि कांस्टेंट कि वैल्यू एक बार डिक्लेअर करने के बाद बदली नहीं जा सकती है।

```
final int s = 345;                  // integer टाइप  कांस्टेंट
final double r = 4.22;              // double टाइप कांस्टेंट
final char firstletter = 'f';
final String  word = "Hello" ;
```

एक्सप्रेशन (Expressions) : जावा में ऑपरेटर्स के सेट हैं जिन्हें एक्सप्रेशंस में उपयोग किया जा सके। एक एक्सप्रेशन वेरिएबल्स , ऑपरेटर्स एवं मेथड आदि शामिल होने से बनती है। निम्न उदाहरण से इसे समझा जा सकता है :

Arithmetic Operators :

Arithmetic operators में जोड़ (+), घटाना (-), गुणा (*), भाग (/) एवं मोडुलस (%) जावा में उपलब्ध हैं। इनके प्रयोग को निम्न उदाहरण से समझ सकते हैं।

```
/**
*Arithmetic Operators
*/

class Arithmetic {

public static void main (String [] args) {

int a = 11;
int b= 3 x a;
System.out.println (b);

int c = (b-a) % 5;
System.out.println (c);
final float pi = 3.1415F;
final d = pi / 0.15F;
System.out.println (d);
}
}
```

इसका आउटपुट

33

4
20.94333333

निम्न उदाहरण शॉर्टहैंड ऑपरेटर्स का है

```
/**
* shorthand operators
*/

class shorthand {

public static void main (String [] args) {

int a = 11;
x+=4;                          // x = x+4
System.out.println (x);
x*=2;                          // x = x*2
System.out.println (x);
}
}
```

उपर्युक्त उदाहरण का आउटपुट

15
30

इस केटेगरी में दो और ऑपरेटर्स जावा में है increment(++) एवं decrement (--). ये दोनों ऑपरेटर्स दो तरीके से उपयोग में आते हैं पहला प्रीफिक्स और दूसरा पोस्टफिक्स . प्रिफिक्स में पहले ऑपरेशन (increment या decrement) होगा उसके बाद वैल्यू रिटर्न होगी जबकि पोस्टफिक्स में पहले वैल्यू रिटर्न होगी उसके बाद ऑपरेशन (increment या decrement) होगा। यह निम्न उदाहरण से स्पस्ट होगा

```
/**
* increment operators
 */

class Increment {

public static void main (String [] args){

int a= 9, b = 9;
```

```
System.out.println (a++);    // a पहले प्रिंट होगा फिर  a की वैल्यू में increment होगा
System.out.println (a);
System.out.println (++b);    // b की वैल्यू पहले increment होगी फिर प्रिंट होगी
System.out.println (b);
}
}
```

इस उदाहरण का आउटपुट

```
9
10
10
10
```

रिलेशनल ऑपरेटर (Relational Operators):

जावा में जो रिलेशनल ऑपरेटर उपलब्बध हैं वे equivalent (= =), not equivalent (!=), less than (<), greater than (>), less than or equal (<=) और greater than or equal (>=) आदि हैं। यहाँ ध्यान देने वाली बात है कि रिलेशनल ऑपरेटर हमेशा boolean value रिटर्न करते हैं। निम्न उदाहरण में कुछ रिलेशनल ऑपरेटर को लिया गया है जिससे रिलेशनल ऑपरेटर का उपयोग स्पस्ट हो जायेगा

```
/**
* Boolean operator application
 */

class Boolean {

public static void main (String [] args){

int a= 12, b = 22;

System.out.println (a<b);
System.out.println (a !=b - 6);

boolean test = a>=8;
System.out.println (test);
}
}
```

उपर्युक्त प्रोग्राम का आउटपुट :

```
true
false
true
```

बिट लेवल ऑपरेटर्स (Bit level operators) : जावा में बिट लेवल ऑपरेटर्स उपलब्बध हैं जिनका उपयोग करके किसी बिट को आवश्यकता के अनुसार बदला जा सकता है। जावा में कुछ बिट लेवल ऑपरेटर्स बूलियन अलजेब्रा (Boolean algebra) को बिट्स पर अप्लाई करते हैं : and(&),or(|)and not (~) while others perform bits shifting : shift left(<<), shift right with sign extension (>>) and shift right with zero extension(>>>).

बाइनरी बिट लेवल ऑपरेटर्स and (&) बूलियन ऑपरेशन एंड (AND)(लॉजिकल ऑपरेशन) ऑपरेशन दो आर्गुमेंट्स की बिट्स के बीच में करता है। बाइनरी बिट लेवल ऑपरेटर्स or (|) बूलियन ऑपरेशन और (OR)(लॉजिकल ऑपरेशन) ऑपरेशन दो आर्गुमेंट्स की बिट्स के बीच में करता है। इसी तरह बाइनरी बिट लेवल ऑपरेटर्स not (~) बूलियन ऑपरेशन नॉट (NOT)(लॉजिकल ऑपरेशन) ऑपरेशन दो आर्गुमेंट्स की बिट्स के बीच में करता है।

बिट वाइस left-shift operator (<<) पहले आर्गुमेंट की बिट्स को लेफ्ट शिफ्ट करता है और कितनी बिट्स शिफ्ट होनी हैं यह दूसरे आर्गुमेंट से पता चलता है इसमें लोअर साइड में 0 जुड़ता जाता है। बिट वाइस right-shift operator (>>>) पहले आर्गुमेंट की बिट्स को राईट शिफ्ट करता है और कितनी बिट्स शिफ्ट होनी हैं यह दूसरे आर्गुमेंट से पता चलता है इसमें हायर साइड में 0 जुड़ता जाता है। बिट वाइस right-shift operator (>>) चिन्ह (sign)के साथ पहले आर्गुमेंट की बिट्स को राईट शिफ्ट करता है और कितनी बिट्स शिफ्ट होनी हैं यह दूसरे आर्गुमेंट से पता चलता है इसमें हायर साइड में 0 या 1 जुड़ता जाता है जिससे पहले आर्गुमेंट का चिन्ह (sign)को समान रखा जा सके। यदि पहला आर्गुमेंट पॉजिटिव है तो 0 जुड़ेगा अन्यथा (नेगेटिव) की स्थिति में 1 जुड़ेगा।

ये ऑपरेटर्स interger टाइप पर ऑपरेट होते हैं। यदि आर्गुमेंट char, short और byte हैं तो पहले इनका टाइप interger में कन्वर्ट होगा एवं उसके बाद ऑपरेशन होगा।

```
/**
```

```
* Boolean algebra bit level operators application
 */

class Bits {

public static void main (String [] args){

int a= 0x16,              //00000000000000000000000000010110

 b = 0x33;                //00000000000000000000000000110011

System.out.println(a&b);  //00000000000000000000000000010010

System.out.println(a|b);  //00000000000000000000000000110111
System.out.println (~x);  //11111111111111111111111111001000
x &= 0xf;                 //00000000000000000000000000000110
System.out.println (x);   //00000000000000000000000000000110
short s=7;                //0000000000000111
 System.out.println (~s); //11111111111111111111111111111000
}
}
```

उपर्युक्त उदाहरण में अंतिम दो लाइन से यह स्पस्ट हो जाता है कि यदि आर्गुमेंट short टाइप का है तब भी ऑपरेटर interger टाइप में रिजल्ट देता है। हर लाइन में दिए गए कमेंट्स पर ध्यान दें उसमें बाइनरी वैल्यूज कैसे बदल रही हैं।

निम्न उदाहरण में बिट लेवल शिफ्ट ऑपरेटर्स कैसे कम करते हैं दर्शाया गया है।

```
 /**
* Bit level operators application
 */

class Bits2 {

public static void main (String [] args){

int a= 0x16,              //00000000000000000000000000010110
System.out.println(a<<4); //00000000000000000000000101100000

 b = 0xfe;                //00000000000000000000000000001111
b >>= 4;
System.out.println (b);   //00000000000000000000000000001111

a=9;                      //00000000000000000000000000001001

System.out.println(a>>3); //00000000000000000000000000000001
System.out.println(a>>>); //00000000000000000000000000000001
```

```
a = - 9                    //11111111111111111111111111110111
System.out.println(a>>3);// 11111111111111111111111111111110

System.out.println(a>>>3);//00011111111111111111111111111110
}
}
```

लॉजिकल ऑपरेटर्स (Logical Operators) :

जावा में उपलब्बध लॉजिकल ऑपरेटर्स हैं and(&&), or(||) और not(!) ये ऑपरेटर्स केवल बूलियन एक्सप्रेशन पर लागू होते हैं एवं बूलियन वैल्यू ही रिटर्न करते हैं ।

```
 /**
* Logical operators application
 */

class Logical {

public static void main (String [] args){

int a= 12, b = 22;
double c= 4.23, d = 2.34;

System.out.println (a<b  && c>d);
System.out.println (!(a<b));

boolean test = 'x' > 'z';
System.out.println (test || d – 1.5 >0);
 }
}
```

इस एप्लीकेशन के आउटपुट कुछ लॉजिकल एक्सप्रेशंस की वैल्यूज रिटर्न करके बताया गया है :

```
true
false
true
```

स्ट्रिंग ऑपरेटर्स (String operators) : जावा स्ट्रिंग ऑपरेटर्स का पूरा एक सेट उपलब्बध कराती है । इस भाग में केवल concatenation operator (+) पर ही

विचार किया गया है। बाकी के ऑपरेटर्स के बारे में बाद के भाग में विचार करेंगे। concatenation operator (+) यह ऑपरेटर आर्गुमेंट्स से प्राप्त दो स्ट्रिंग्स को जोड़ता है एवं एक नया स्ट्रिंग बनता है। इस ऑपरेटर की यह विशेषता है कि एक्सप्रेशन स्ट्रिंग से शुरू होती है एवं + द्वारा अगले आर्गुमेंट को भी स्ट्रिंग में परिवर्तित कर देता है ।

```
/**
* String operators application
 */

class String {

public static void main (String [] args){

String s1= “Hello” + “World”;
System.out.println (s1);
Int i=44, j=21;
System.out.println (“The value of i is “+ i + “and The value
of j is “ + j);
}
}
```

इसका आउटपुट निम्न है :

```
Hello World

The value of i is 44 and the value of j is 21
```

इसमेंi एवं j की वैल्यू पहले integer से स्ट्रिंग में बदली फिर श्रंखलाबद्ध हुई।

कास्टिंग (Casting):

जावा में आटोमेटिक टाइप कन्वर्जन की अनुमति है यदि कन्वर्जन से डाटा लॉस का जोखिम नहीं है क्योंकि इसमें हम टाइप का साइज़ बड़ा (Widening)रहे हैं । इसकी विशेषताएं निम्न उदाहरण से समझ सकते हैं।

```
/**
* Test Widening conversions Application
 */

class TestWide {

public static void main (String [] args){

int a =’x’;                          // x एक कैरक्टर है
```

```
long b=24;                          // 24 एक integer है
float c = 1002;                     // 1002 एक integer है
double d = 3.42F;                   // 3.42 एक float है
}
}
```

जब हम कन्वर्जन में डाटा का साइज़ छोटा कर रहे हैं तब शायद डाटा लॉस हो सकता है। ऐसी स्थिति में कास्ट ऑपरेटर (cast) का उपयोग आवश्यक हो जाता है। इसको निम्न उदाहरण से समझा जा सकता है :

```
/**
* Test Narrowing conversions Application
 */
class TestNarrow {

public static void main (String [] args){

long a = 45;
int  b= (int) a;             // a long से integer में बदला
double c = 2.56;
float d = (float)c;         //  c double se float में बदला
}
}
```

जावा में अनुशंसित है कि कन्वर्जन तभी करना है जब पक्का हो जाये कि डाटा लॉस नहीं होगा।

कंट्रोल स्ट्रक्चर (Control Structure) :

जावा में विभिन्न कंट्रोल स्ट्रक्चर में कंडीशनल एक्सप्रेशन में बूलियन वैल्यूज ही उपयोग में आती हैं।

The if control statement:

If कंट्रोल स्टेटमेंट के दो फॉर्म जावा में उपलब्बध हैं जो कि बूलियन एक्सप्रेशन की वैल्यू पर निर्भर करते हैं।

```
Form 1
If (boolean expression) {
 statement
}
Form2
If (boolean expression) {
 statement
}
else {
Statement
}
```

```
 /**
* If control statement Application
 */

class If {

public static void main (String [] args){

char c = ‘x’

if ((c>=’a’ && c <=’z’) || ((c>=’A’ && c <=’z’)
System.out.println (“Letter: “ +c);
else if ((c>=’0’ && c <=’9’)
System.out.println (“Digit: “ +c);
else {

System.out.println (“The character is : “ +c);
System.out.println (“It is not a letter “);
System.out.println (“and it is not a Digit”);
}
}
}
```

इसका आउटपुट :

```
Letter: x
```

Iteration control statement: जावा में while and do-while iteration कंट्रोल स्टेटमेंट्स हैं । ये iteration कंट्रोल स्टेटमेंट्स बूलियन एक्सप्रेशन जब तक true है तब तक एक्सिक्युसन दोहराया जाता है ।

```
while (boolean-expression)
statement
```

and

```
do
statement
while (boolean-expression);
```

यहाँ पर while स्टेटमेंट जीरो और ज्यादा बार एक्सिक्युट होता है जबकि do-while एक और ज्यादा बार एक्सिक्युट होता है क्योंकि इसमें बूलियन एक्सप्रेशन की जांच एक बार एक्सिक्युट होने के बाद ही होती है । निम्न उदाहरणों का अवलोकन करें :

```
/**
* While control statement Application
 */

class While {

public static void main (String [] args){

final float initialvalue = 2.34F;
final float step = 0.11F;
final float finalvalue = 4.69F;
float variable=initialvalue;
int count =0;

while(variable < finalvalue) {
variable += step ;
count ++;
}
System.out.println("The number of counts is " + count
+"Times");
}
}
```

इस उदाहरण का आउटपुट

```
The number of counts is 12 Times
```

निम्न उदाहरण Do-While का है जिससे दोनों कंट्रोल स्ट्रक्चर में तुलना हो सके

```
/**
* Do-While control statement Application
 */

// Class
class DWhile {

    // Main driver method
    public static void main(String[] args)
    {
        // initial counter variable
        int i = 0;

        do {
            // Body of loop that will execute minimum
            // 1 time for sure no matter what
            System.out.println("The value of i is" +i);
            i++;
        }
        // Checking condition
        // Note: It is being checked after
        // minimum 1 iteration
        while (i < 0);
    }
}
```

जावा में तीसरा कंट्रोल स्ट्रक्चर for()है जिसका सिंटेक्स निम्न है

```
for (initialValue; boolean-expression; step)
{      Statement;
}
```

for() लूप स्ट्रक्चर में तीन एक्सप्रेशंस हैं पहली initialValue; दूसरी boolean-expression; तीसरी step (boolean expression)। इसी आर्डर में ये एक्सप्रेशंस एक्सिक्युट होती हैं। बूलियन एक्सप्रेशन जब तक true है तब तक for() एक्सिक्युट होता है। for() में तीनो एक्सप्रेशंस एक ही स्टेटमेंट में लिखी जाती हैं और इस तरह दो लाइन बच जाती हैं ।

```
 /**
* For control statement Application
 */
```

```
class While {

public static void main (String [] args){

final float initialvalue = 2.34F;

final float step = 0.11F;

final float finalvalue = 4.69F;

int count =0;

for (float variable = initialValue; Variable  < finalvalue;
variable +=step) {
count ++;
System.out.println(“The number of counts is “ + count
+”Times”);
}
}
```

यहाँ पर एक ही उदाहरण को लिया गे है जिससे आपस में तुलना हो सके।

ब्रेक एंड कंटिन्यू (break and continue):

जावा में दो स्टेटमेंट्स break and continue जिनका उपयोग लूप के अन्दर कंट्रोल करने के लिए होता है। continue से वर्तमान में चल रहा iteration वहीँ समाप्त हो जाता है और लूप के अन्दर ही नया iteration शुरू हो जाता है जबकि break का उपयोग करने पर वर्तमान में चल रहा iteration वहीँ समाप्त हो जाता है और कंट्रोल लूप से बाहर आ जाता है। इसको दूसरे शब्दों में इस प्रकार समझ सकते हैं कि continue के उपयोग में सिर्फ वर्तमान iteration ही समाप्त होता है और लूप के बचे हुए iteration भी एक्सीक्यूट होते हैं जबकि break के उपयोग से कंट्रोल लूप से बहार आने पर वर्तमान के साथ साथ सभी बचे हुए iteration समाप्त हो जाते हैं और कंट्रोल अगले इंस्ट्रक्शन पर चला जाता है।

```
 /**
* Break and continue statements  Application
 */
class BreakContinue {

public static void main (String [] args){

for (count = 0; count <15; count++) {
```

// यदि boolen expression की वैल्यू true है तो नया iteration शुरू होगा इसका मतलब है कि काउंट एक विषम संख्या है।

```
if (count %2 = = 1) continue;

// abandon the loop if the value of count is 8

if( count = =10) break;

// print the value

System.out.println(count);
}
System.out.println("End of the loop using break");
}
}
```

इसका आउटपुट निम्न है

```
0
2
4
6
8
```

यहाँ पर कंट्रोल लूप से बाहर चला गया और लूप बंद हो गया।

यहाँ पर पहले if में count की वैल्यू बिषम होने पर इसकी बूलियन एक्सप्रेशन true होती है इस केस में continue स्टेटमेंट मौजूदा iteration को समाप्त कर for स्ट्रक्चर की step एक्सप्रेशन को एक्सिक्युट करती है एवं उसके परिणाम के आधार पर for स्ट्रक्चर की बूलियन एक्सप्रेशन को चेक करती है। यदि बूलियन एक्सप्रेशन true वैल्यू देता है तब नया iteration शुरू हो जाता है।

इसी तरह दूसरे if में count की वैल्यू 10 होने पर इसकी बूलियन एक्सप्रेशन true होती है इस केस में break स्टेटमेंट मौजूदा iteration को समाप्त लूप से बहार आ जाता है और इस तरह कंट्रोल for लूप से बहार हो जाता है।

स्विच कंट्रोल स्टेटमेंट (Switch control statement):

Switch कंट्रोल स्टेटमेंट अपने इंटीग्रल एक्सप्रेशन को चेक करता है और उसकी वैल्यू के आधार पर एक कोड (केस case) को एक्सिक्युट करने के लिए चुनता है।

```
switch (integral-expression) {
case integral-value: statement; [break;]
```

```
case integral-value: statement; [break;]
case integral-value: statement; [break;]
case integral-value: statement; [break;]
[default: statement;]
}
```

इसमें [ ] (बड़ा कोष्टक) यह दर्शाता है की यह वैकल्पिक है इसका मतलब है की प्रोग्रामर लॉजिक एवं स्थिति के अनुसार इस उपयोग कर भी सकता है और नहीं भी। switch स्टेटमेंट की इंटीग्रल वैल्यू चेक होती है और उसके आधार पर जिस केस की वैल्यू इसके बराबर है उस केस को switch स्टेटमेंट एक्सिक्युट करता है। यहाँ पर break स्टेटमेंट वैकल्पिक है। यदि break स्टेटमेंट है तब switch स्टेटमेंट वहीँ समाप्त हो जाता है और कंट्रोल switch से बहार आ जाता है। परन्तु यदि break स्टेटमेंट नहीं है तब switch स्टेटमेंट अगले केस को एक्सिक्युट करता है और यह सिलसिला तब तक चलता है जब तक break स्टेटमेंट नहीं मिल जाता या switch स्टेटमेंट के सभी केस एक्सिक्युट नहीं हो जाते।

इसमें ध्यान देने की बात है कि कोई भी एक्सप्रेशन जैसे char, short, byte or int हो वो जो भी वैल्यू रिटर्न करते हैं वह वैल्यू integer में बदल जाती है।

```
/**
* Switch control statement Application
 */

class Switch {

public static void main (String [] args){

boolean leapyear = true;

int days=0;

for( int month =1; month <=12; month++){

switch (month) {
case 1:
case: 3
case 5:
case 7:
case 8:
case 10:
case 12:
days += 31;
```

```
break;
case 2:
if (leapyear)
days +=29;
else days +=28;
break;
default :
days +=30;
break;
}
}
System.out.println(“The number of days in a year is “ +
days);
}
}
```

यह उदाहरण एक वर्ष में कितने दिन होते हैं इसकी गणना करने के लिए बनाया गया है। इसमें केस (1,3,5,7,8,10,12,) महीने 31 दिन के हैं और फरवरी के महिने में दिन लीप इयर से पता चलते हैं एवं बाकी महीने 30 दिन के होते हैं। इसका लॉजिक switch स्टेटमेंट लगाकर प्रोग्राम बना है।

ऐरे (Arrays):

ऐरे स्ट्रक्चर एक तरह के कई अवयवों (Elements) को स्टोर करने के लिए उपयोग होता है निम्न उदाहरण से इसे समझने का प्रयास करते हैं

```
int [ ] a;      //a एक integer का  array है जो अभी इनिशियलाइज़ नहीं है

float [ ] b;     // b एक float का array है जो अभी इनिशियलाइज़ नहीं है

 string [ ] c;  // c एक स्ट्रिंग  का array है जो अभी इनिशियलाइज़ नहीं है
```

यहाँ पर array का साइज़ नहीं बताया गया है क्योंकि इस डिक्लेरेशन में अभी अवयवों के लिए मेमोरी में जगह भी आवंटित नहीं हुई है। अतः array का साइज़ डिक्लेरेशन के समय इनिशियलाइज़ करने पर निर्दिष्ट किया जाता है।

```
int  [ ] a = {2,4,6,15,26,56};                    //size : 6

float [ ] b = {1.4F,1.5F,4.5F,6.6F};              // size :4

string [ ] c = { “Java”, “is”, “very”, “useful”}; // size:4
```

दूसरी संभावना यह है कि new ऑपरेटर का उपयोग करके मेमोरी में जगह भी आवंटित की जाती है और विशेष बात यह है कि array का साइज़ की गणना एक्सिक्युसन के समय की जा सकती है।

```
int i=2, j=4;

double [] b;        // b एक double  का array है जो अभी इनिशियलाइज़ नहीं है
b = new double [i + j];

            // इस array का साइज़ की गणना एक्सिक्युसन के समय की जा सकती है।
```

जिसका साइज़ i एवं j का जोड़ 6 double टाइप्स अवयवों का है।

यहाँ पर new ऑपरेटर का उपयोग मेमोरी एलोकेशन को गतिशील (dynamically) बनाने के लिए किया गया है। और इसमें array के अवयव को संख्यात्मक वैल्यू के लिए डिफ़ॉल्ट वैल्यू 0, कैरेक्टर या नल(null) के लिए डिफ़ॉल्ट वैल्यू '\0' से इनिशियलाइज़ करते हैं।

array के अवयवों का अभिगम एक integer अनुक्रमणिका से किया जाता है ये 0 से array के साइज़ में से एक कम तक होता है उदाहरण के लिए यदि array के अवयवों का साइज़ 20 है तब इसकी अनुक्रमणिका 0 से शुरू होकर 19(20-1) तक होगी।

```
a[2] =1000;                    // यहाँ पर array के तीसरे एलिमेंट का संशोधित होगा
```

निम्न उदाहरण से array का उपयोग को समझा जा सकता है :

```
/**
* Arrays Application
 */

class Arrays {

public static void main (String [] args) {

int [ ] a = {2,4,3,1];

//  सभी एलेमेंट्स के जोड़ की गणना
```

```
int sum =0;

for ( int i=0; i < a.length; i++){
sum += a[i];
}

// एक floats का array बनाना जिसका साइज़ उपर्युक्त sum की वैल्यू होगी

float [ ] b =new.float[sum];

// assign some values

for ( int i=0; i < b.length; i++){
b[i] = 1.0F/i;
}

// print the values in odd positions

for ( int i=0; i < b.length; i += 2){
System.out.println("b["; + i + "]=" + b[i]);
}
}
}
```

इसका आउटपुट निम्न है :

```
b[1] = 1.0
b[3] = 0.3333334
b[5] = 0.2
b[7] = 0.14285715
b[9] = 0.11111111
```

एक बहुआयामी array को वही तरीके से डिक्लेअर करते हैं। निम्न उदाहरण में array में 72 अवयवों को 8 rows एवं 9 column में स्टोर कर सकते हैं।

```
int[ ][ ] x = new.int[8][9];
```

कमांड लाइन आर्गुमेंट्स (Command line arguments) :

```
public static void main (String []args)
```

उपर्युक्त स्टेटमेंट का अवलोकन करने पर पता चलता है कि इसमे एक मेथड है जिसका नाम main() है। यह मेथड एक आर्गुमेंट स्ट्रिंग के array के रूप में लेता है। जब प्रोग्राम को JVM में एक्सिक्युट कने के लिए भेजा जाता है तब कमांड लाइन

arguments का उपयोग इस array के द्वारा करते हैं। निम्न उदाहरण में कमांड लाइन आर्गुमेंट को प्रिंट करके का तरीका बताया गया है।

```
 /**
* Command Line Arguments Application
 */

class CommandArguments {

public static void main (String [] args) {

for ( int i = 0; i <args.length; i++){
System.out.println("args = " args[i]);
}
}
}
```

इस उदाहरण का आउटपुट

```
#  Command line argument: Java World
Java
World
# Command line argument: I have 15 Rupees
I
Have
15
Rupees
```

इस आउटपुट में 15 एक integer हैं परन्तु यह वैल्यू कमांड लाइन arguments से आ रही है अतः ये स्ट्रिंग की तरह उपयोग में आएगा और ये args[2] में स्टोर होगा। एक और मेथड है parseInt() है जो कि class integer की मेथड है। ये parseInt()का उपयोग स्ट्रिंग से integer को बदलने के लिए होता है।

निम्न उदाहरण में दो integer जो arguments को कमांड लाइन प्राप्त करने एवं दोनों को मिलाकर प्रिंट करना है।

```
/**
* Addition of two command line arguments Application
 */

class AddCommands {
```

```
public static void main (String [] args) {

if (args.length !=2){
System.out.println(“Error”);
System.exit(0);
}

int arg1 = Integer.parseInt(args[0]);
int arg2 = Integer.parseInt(args[1])

System.out.println(arg1 + arg2);
}
}
```

इस प्रोग्राम का समापन एक मेथोस exit()जो कि class सिस्टम की हैं , से किया जाता है exit()मेथड 0 वैल्यू रिटर्न करती है।

क्लासेज(Classes)

जावा में एक कीवर्ड Class का उपयोग होता है जिससे Class को उसके नाम के साथ बनाया जाता है। उदाहरण के तौर पर

```
Class Car{
}
```

यहाँ पर एक Class जिसका नाम कार है को बनाया गया है अब new कीवर्ड का उपयोग करके इस Class के कितने भी इंस्टांस बनाये जा सकते है। यहाँ पर दो प्रकार से बनाये जाने का उल्लेख है पहला एक step में

```
Car c1 = new Car ( );
Car c2 = new Car ( );
```

दूसरा दो steps में

```
Car c3;
c3 = new Car( );
```

अभी ये Class में कुछ नहीं होने के कारण इनका उपयोग नहीं हो सकता है। Class को उपयोगी बनाने के लिए Class के अन्दर डाटा मेम्बेर्स या फ़ील्ड्स को डिफाइन किया जाता है जो इनफार्मेशन को स्टोर करने के काम में आती हैं इसके आलावा Class के अन्दर ही मेथोड्स या मेम्बर फंक्शन बनाये जाते है और ये मेथड का उपयोग Class के इंस्टांस के बीच में बातचीत (communicate) करने के लिए होता है।

यदि हम Car की कुछ इनफार्मेशन जैसे मॉडल , मालिक का नाम , कार का एवरेज आदि। इसके लिए निम्न उदाहरण है।

```
class Car {

String Model;
String Owner;
int AverageMileage;
}
```

इसमें हमने तीन फ़ील्ड्स डिफाइन की हैं तो जब भी हम इस Class का इंस्टांस बनायेंगे तो उसमें तीन ये तीनों फ़ील्ड्स होंगी। इन फील्ड का अभिगम डॉट ऑपरेटर (.)का उपयोग करके किया जाता है। ये डॉट (.)ऑपरेटर इंस्टांस के नाम एवं फील्ड के नाम के बीच में उपयोग होता है।

```
/**
*  Example of Cars with fields  Application
 */

class Car {
String Model;
String Owner;
int AverageMileage;
}


class CarExample  {

public static void main (String [] args) {

Car c1;

c1 = new Car ( );

c1.Model = “WagonR” ;
c1.Owner = “Amit Kumar”;
c1.AverageMileage = 18;

System.out.println(c1.Model  +  “:-”  +  c1.Owner  +  “-”  +
c1.AverageMileage);

}
}
```

इस उदाहरण का आउटपुट निम्न है

```
WagonR :- Amit Kumar – 18
```

कंस्ट्रकटर (Constructors):

कंस्ट्रकटर एक मेथड है जिसका नाम उसी Class का नाम होता है। जिसके लिए कंस्ट्रकटर बना है। इसका मतलब हुआ कि दोनों का नाम एक ही होता है। कंस्ट्रकटर का उपयोग ऐसा इंस्टांस को बनाने के लिए होता है जो कि अच्छी तरह से इनिशियलाइज़ होता है। कंस्ट्रकटर कोई वैल्यू रिटर्न नहीं करता है।

कंस्ट्रकटर का उदाहरण निम्न है :

```
/**
*  Example-2 of Cars with fields  Application
 */
class Car {
String Model;
```

```
String Owner;
int AverageMileage;

Car(String mod, String own, int am){
Model = mod;
Owner =own;
AverageMileage = am;
}
}


class CarExample2  {

public static void main (String [] args) {

Car c1;

c1 = new Car (“WagonR”,”Amit Kumar”,18);

System.out.println(c1.Model  +  “:-”  +  c1.Owner  +  “-”  +
c1.AverageMileage);

}
}
```

इसका आउटपुट निम्न है

```
WagonR :- Amit Kumar – 18
```

इस उदाहरण से स्पष्ट है कि कंस्ट्रकटर को इंस्टांस बनाने के लिए ही कॉल करते हैं। एक और बात ध्यान रखने की है कि हर class में एक डिफ़ॉल्ट कंस्ट्रकटर होता ही है। इसीलिए उपर्युक्त उदाहरण में कंस्ट्रकटर को सीधे कॉल निम्नानुसार किया गया

```
c1=new Car( );
```

यहाँ पर कंस्ट्रकटर बिना आर्गुमेंट के ही कॉल किया गया है।

किसी भी Class में डिफ़ॉल्ट कंस्ट्रकटर तभी कॉल कर सकते है जब तक इसका कंस्ट्रकटर डिफाइन नहीं किया है यदि कंस्ट्रकटर डिफाइन कर दिया तब डिफ़ॉल्ट कंस्ट्रकटर कॉल नहीं होगा। इसलिए दूसरे उदाहरण में जो कंस्ट्रकटर तीन फ़ील्ड्स के साथ डिफाइन किया गया है उसे ही कॉल किया गया।

हम एक ही Class के कई कंस्ट्रकटर जिसमें अलग अलग संख्या में आर्गुमेंट्स के साथ या अलग अलग प्रकार के आर्गुमेंट्स के साथ डिफाइन कर सकते हैं। जब भी कोई

कंस्ट्रकटर को काल किया जाता है उसके आर्गुमेंट्स के आधार पर कम्पाइलर यह पता करता है कि उपलब्बध कंस्ट्रकटर में से किस कंस्ट्रकटर को कॉल किया गया है। उदाहरण के लिए :

```
  /**
*  Example-3 of Cars with fields  Application
 */

class Car {
String Model;
String Owner;
String Registration_number;
int AverageMileage;


Car(String mod, String own, int am) {

Model = mod;
Owner =own;
REGISTRATION_NUMBER =”Not Known”
AverageMileage = am;
}

Car(String mod, String own, String Regn, int am){

Model = mod;
Owner =own;
REGISTRATION_NUMBER =Regn;
AverageMileage = am;
}
}


class CarExample3  {

public static void main (String [] args) {

Car c1,c2;

c1 = new Car (“WagonR”,”Amit Kumar”,18 );

System.out.println(c1.Model  +  “:-”  +  c1.Owner  +  “-”  +
c1.AverageMileage + “:” + c1.REGISTRATION_NUMBER);

c2 = new Car (“WagonR”,”Amit Kumar”,”CG10FA5136”,18 );
```

```
System.out.println(c2.Model + ":-" + c2.Owner + "-" + c2.REGISTRATION_NUMBER +":" + c2.AverageMileage);

}
}
```

इसका आउटपुट निम्न है :

```
WagonR :- Amit Kumar – 18 : Not Known
WagonR :- Amit Kumar – CG10FA5136 : 18
```

इस आउटपुट से एक बात साफ़ है की जब कंस्ट्रकटर कॉल करते समय तीन आर्गुमेंट्स दिया गए तब पहला कंस्ट्रकटर कॉल हुआ और जब चार आर्गुमेंट्स दिए गए तब दूसरा कंस्ट्रकटर कॉल हुआ।

मेथड (Methods):

एक मेथड का उपयोग एक मेसेज जो एक इंस्टांस या क्लास द्वारा प्राप्त किया गया हो, को अमल में लाने के लिए किया जाता है। एक मेथड को फंक्शन जैसे अमल में लाया जाता है जो आर्गुमेंट्स को निर्दिष्ट करती है और एक टाइप की वैल्यू रिटर्न करती है। मेथड भी डॉट (.) ऑपरेटर का उपयोग करके कॉल की जाती है।

मेथड को समझाने के लिए Car का एक और उदाहरण प्रस्तुत है :

```
/**
*  Example-4 of Cars with methods  Application
 */

class Car {
String Model;
String Owner;
int AverageMileage;

Car(String mod, String own, int am){
Model = mod;
Owner =own;
AverageMileage = am;
}

public String getInitials( ){
String initials = " ";

for ( int i = 0; i < Owner.length( ); i++){
```

```
  char currentChar = Owner.charAt(i);
if (currentChar >= 'A' && currentChar<= 'Z'){
initials = initials + currentChar +'.';
}
}
   return initials;
}
}


class CarExample4  {
public static void main (String [] args) {

Car Car1;

Car1 = new Car ("WagonR","Amit Kumar",18 );
System.out.println("Initials : " + Car1.getInitials( ));
}
}
```

इसका आउटपुट है :

```
Initials:  A.K.
```

इस उदाहरण में एक मेथड का प्रोटोटाइप getInitials( ) को डिफाइन किया गया है

```
public String getInitials( )
```

getInitials( ) मेथड public डिफाइन है इसका मतलब है कि इस मेथड को दूसरी Class में भी कॉल किया जा सकता है। इसे हम इसी किताब में बाद में समझेंगे। इस मेथड में कोई भी आर्गुमेंट पास नहीं किया गया है। ऑब्जेक्ट ओरिएंटेड टर्मिनोलॉजी में कहते है कि एक मेसेज "getInitials" ऑब्जेक्ट Car1 को भेजा गया। अतः ऑब्जेक्ट Car1 मेसेज का रिसेप्टर है।

```
System.out.println("Initials : " + Car1.getInitials( ));
```

यहाँ पर स्पष्ट हो जाता है कि कैसे getInitials( ) मेथड एवं Car1 इंस्टांस को डॉट (.) ऑपरेटर्स के उपयोग किया जाता है। और यह मेथड कोई भी आर्गुमेंट को प्राप्त नहीं किया है फिर भी स्ट्रिंग टाइप की वैल्यू रिटर्न करती है।

मेथड का कार्यान्वयन निम्नानुसार होता है :

```
public String getInitials( ){
```

```
String initials = “ ”;

for ( int i = 0; i < Owner.length( ); i++){
  char currentChar = Owner.charAt(i);
if (currentChar >= ‘A’ && currentChar<= ‘Z’){
initials = initials + currentChar +’.’;
}
}
   return initials;
}
```

इसका एक्सिक्युसन निम्नानुसार होता है

उपर्युक्त उदाहरण में एक फील्ड है Owner है जो की मेसेज के लिए Car1 इंस्टांस का रिसेप्टर है। मेथड getInitials( ) एक खाली स्ट्रिंग एक वेरिएबल initials में बनता है और उसे ट्रान्सफर करता है जिससे उसके हर शब्द में Uppercase लैटर को ढूंडा जा सके। जब भी uppercase लैटर मिलता है उसे स्ट्रिंग वेरिएबल initials में डॉट (.) के साथ जोड़ देता है। जैसा की उदाहरण में है।

यहाँ पर दो मेथोड्स length( ) and charAt( ) का उपयोग किया गया है जिससे स्ट्रिंग की लम्बाई पता चलता है और charAt( ) मेथड से character की पोजीशन स्ट्रिंग में पता चलती है।

यहाँ पर समझाने वाली बात है कि मेथड getInitials( ) Owner में स्टोर डाटा पर प्रक्रिया करती है और एक विशिष्ट फील्ड के रिसेप्टर से मेल खाती है। निम्न उदाहरण में इस अवधारणा को समझाने के लिए चार Car का array डिफाइन किया है और अलग अलग डाटा से इनितिअलिज़ किया है।

```
class CarExample5  {

public static void main (String [] args) {

Car Car;

Car = new Car[4];

Car[0] = new CAR (“WagonR”,”Dr Amit Kumar”,18);
Car[1] = new CAR (“Xcent ”,” Avinash Kumar”,19);
Car[2] = new CAR (“Alto”,” Ad Anurag Kumar ”,16);
Car[3] = new CAR (“HondaCity”,” Prafulla Shrivastava”,15);

for (i=0; i<= Car.length ( ); i++) {

System.out.println(“Initials : ” + Car[i].getInitials( ));
```

```
}
}
}
```

इसका आउटपुट निम्न है:

```
Initials: D.A.K.
Initials: A.K.
Initials: A.A.K.
Initials: P.S.
```

एकुआलिटी और एकुविवेलेंस (Equality and equivalence):

जब भी हम दो ऑब्जेक्ट्स की तुलना equality ऑपरेटर (==)से करते हैं तो उसका परिणाम भ्रामक होता है

निम्न उदाहरण को देखें :

```
class CarExample6  {

public static void main (String [] args) {

Car Car1, Car2;

Car1 = new CAR (("WagonR","Amit Kumar",18);
Car2 = new CAR (("WagonR","Amit Kumar",18);

if(Car1 == Car2)
System.out.println("The two Cars are same);
else
System.out.println("The two different Cars are there");
}
}
```

इसका आउटपुट निम्न आता है

```
The two different Cars are there.
```

विचार करने योग्य बात यह है कि ऐसा क्यों हुआ। इसका कारण यह है कि जब भी equality ऑपरेटर (==) चेक करता है तो वह चेक करता है की दोनों ऑब्जेक्ट्स जिन्होंने आर्गुमेंट्स पास लिए हैं क्या वो दोनों ऑब्जेक्ट वही ऑब्जेक्ट हैं। यहाँ पर equality ऑपरेटर (==) आर्गुमेंट्स द्वारा पास वैल्यूज को चेक नहीं कर रहा है। इसको निम्न उदाहरण से समझा जा सकता है।

```
class CarExample7  {

public static void main (String [] args) {

Car Car1, Car2;

Car1 = new CAR ((“WagonR”,”Amit Kumar”,18);
Car2 = Car1;

if(Car1 == Car2)
System.out.println(“The two Cars are same);
else
System.out.println(“The two different Cars are there”);
}
}
```

अब इसका आउटपुट है

```
The two Cars are same.
```

इस परिणाम से स्पष्ट होता है की इस केस में एक्सप्रेशन `Car1 == Car2` true वैल्यू रिटर्न कर रही है क्योंकि दोनों वेरिएबल्स `Car1` एवं `Car2` एक ही ऑब्जेक्ट को उद्घृत कर रहे हैं।

अब प्रश्न यह उठता है कि जब दो ऑब्जेक्ट में समान वैल्यूज हैं तब equality कैसे चेक होगी। इसके लिए हमें एक मेथड डिफाइन करनी होगी जो एक एक वैल्यू को चेक करेगी। सामान्तः इस मेथड का नाम `equals( )` ही रखा जाता है इसलिए हम भी इसी नाम से मेथड निम्नानुसार बना सकते हैं।

```
 public boolean equals (Car c) {
return(Model.equals(c.Model)  &&  Owner.equals(c.Owner)  &&
AverageMileage.equals(c.AverageMileage)&&
REGISTRATION_NUMBER.equals (c.REGISTRATION_NUMBER);
}
```

यहाँ पर `equals( )` दोनों ऑब्जेक्ट के एक एक वेरिएबल को चेक करती है इस मेथड को `class CarExample` में उपयोग करे का तरीका :

```
class CarExample8  {

public static void main (String [] args) {

Car Car1, Car2;

Car1 = new CAR ((“WagonR”,”Amit Kumar”,18);
```

```
Car2 = new CAR (("WagonR","Amit Kumar",18);

if(Car1.equals(Car2))
System.out.println("The two Cars are same);
else
System.out.println("The two different Cars are there");
}
}
```

और इस प्रकार वांछित परिणाम आया और वह है :

```
The two Cars are same.
```

इसमें मेथड equals() एक रिफरेन्स आर्गुमेंट से प्राप्त करती है और बूलियन वैल्यू रिटर्न करती है। और फिर इस बूलियन वैल्यू की गणना करके प्रत्येक फील्ड का परिणाम बनती है।

स्टेटिक डाटा मेम्बर (Static Data members):

class Car में चार डाटा मेम्बेर्स हैं (Model, Owner, REGISTRATION_NUMBER, AverageMileage) और ये चारों डाटा मेम्बेर्स इस class के प्रत्येक इंस्टांस में उपलब्बध रहेंगे। ये डाटा मेम्बेर्स प्रत्येक इंस्टांस में स्वतंत्र रूप से उसकी वैल्यू स्टोर करते हैं। इससे एक इंस्टांस में विशिष्ट डाटा मेम्बर की वैल्यू दूसरे इंस्टांस उसी डाटा मेम्बर की वैल्यू से भिन्न होती है।

सभी Static data members (या class variables) वो फ़ील्ड्स हैं जो सिर्फ class से सम्बंधित हैं और ये इंस्टांस से सम्बंधित नहीं हैं। इसका मतलब है कि Static data members की सिर्फ एक ही कॉपी होती है चाहे उसके कितने भी इन्स्टान्सेस बनाये जाएँ। Static data members का उपयोग के लिए निम्न उदाहरण देखें

```
/**
*  Example-9 of Cars with static data members  Application
 */

class Car {
String Model;
String Owner;
String Registration_number;
int AverageMileage;
```

```
Car(String mod, String own, int am) {
Model = mod;
Owner =own;
REGISTRATION_NUMBER =”Not Known”;
AverageMileage = am;
}

Car(String mod, String own, String Regn, int am){
Model = mod;
Owner =own;
REGISTRATION_NUMBER =Regn;
AverageMileage = am;
}

public String getInitials( ){
String initials = “ ”;
for ( int i = 0; i < Owner.length( ); i++){
  char currentChar = Owner.charAt(i);
if (currentChar >= ‘A’ && currentChar<= ‘Z’){
initials = initials + currentChar +’.’;
}
}
   return initials;
}

public boolean equals (Car c) {
return(Model.equals(c.Model)    &&    Owner.equals(c.Owner)&&
AverageMileage.equals(c.AverageMileage)&&
REGISTRATION_NUMBER.equals (c.REGISTRATION_NUMBER);
}

public void setColor (String col){
 color = col;
}

public string getColor ( ){
return color;
}
}

 class CarExample9  {

public static void main (String [] args) {

Car c1,c2;
```

```
c1 = new Car (“WagonR”,”Amit Kumar”,18 );
c2  new Car (“xcent”,”Avinash Kumar”,16);
c1.setColor(“White”);

System.out.println(“The color of”+ c1.model+ “is” +
c1.getColor());
System.out.println(“The color of”+ c2.model+“is” +
c2.getColor());
}
}
```

इसका आउटपुट :

```
The color of WagonR is White
The color of Xcent is White
```

उपर्युक उदाहरण में car c1 का कलर एक static वेरिएबल एक मेथड setColor()के द्वारा ऑब्जेक्ट c1 के लिए white डिफाइन किया गया। क्योंकि एक ही static डाटा मेम्बर कलर है इसे किसी भी इंस्टांस में कॉल किया जा सकता है। और यहाँ पर दोनों इंस्टांस c1 और c2 के लिए कॉल किया गया और दोनों का परिणाम भी समान आया।

इस उदाहरण से यह निष्कर्ष निकलता है कि Static Data member का उपयोग करके एक ग्लोबल वैल्यू को class लेवल पर स्टोर किया जा सकता है जिससे एक ही क्लास में Static Data member का उपयोग विभिन्न इंस्टांस के बीच में संचार का माध्यम बना सकते हैं।

स्टेटिक मेथड (Static Method (या Class Method)):

जावा में Static Method (या Class Method) भी उपलब्बध है। उपर्युक उदाहरण में Static data member की तरह ही Static Method (या Class Method) को भी डिफाइन किया जा सकता है। ये Static Method सीधे class के लिए ही काम करती हैं न कि Class के इंस्टांस के साथ।

class Car के लिए हम निम्नानुसार Static Method डिफाइन कर सकते हैं :

```
public static String trafficRules( ){
return “Please wear seat belt for safety”;
}

class CarExample10  {
```

```
public static void main (String [] args) {

Car Car1;

Car1 = new CAR (("WagonR","Amit Kumar",18);

System.out.println(c1.trafficRules());
System.out.println(Car.trafficRules());

}
}


Please wear seat belt for safety
Please wear seat belt for safety
```

यहाँ पर एक ही static method को दो तरह से कॉल किया गया है पहले इंस्टांस से और बाद में class से कॉल किया गया है। दोनों ही स्थिति में मेथड ने एक ही वैल्यू को रिटर्न किया है। एक और महत्वपूर्ण बिंदु है कि trafficRules() की तरह static मेथड केवल static डाटा मेम्बर को ही कॉल करती है।

स्टेटिक एप्लीकेशन(Static Application):

अभी तक हमने जो भी उदाहरण देखें हैं उनमें class में एक मेथड main() होती है जिसमें प्रायः दूसरी class से इंस्टांस बनाते हैं। जावा में class केवल static मेथड एवं static डाटा मेम्बेर्स से भी डिफाइन करना संभव है।

```
/**
*  Example-11 all static class Application
 */

import java.io.*

class AllStatic{
static int a;
static String s;

public static String asString (int number){
       return " " + number;
}

Public static void main( String [ ] args){
a = 132;
s = asString(a);
```

```
System.out.println(s);
}
}
```

इस उदाहरण में दो static फ़ील्डस a और s परिभाषित की गयी हैं इसी के साथ दो static मेथोड्स भी हैं asString() और main(). इसमें main() मेथड asString()मेथड को कॉल करती है यह तभी संभव है जब दोनों मेथोड्स static हैं। इसलिए इसमें मेसेज भेजने के लिए इंस्टांस बनाने की जरुरत नहीं है। जावा में जब केवल static फ़ील्ड्स एवं static मेथोड्स परिभषित होती हैं तब यह class फंक्शन एवं ग्लोबल डाटा के साथ स्टैण्डर्ड C का प्रोग्राम लगता है। इस उदाहरण में एक बात और ध्यान देने की है कि मेथड asString(),concatenation operator (+) ऑपरेटर का उपयोग करके integer वैल्यू को स्ट्रिंग में बदल देती है। क्योंकि concatenation operator (+) ऑपरेटर की विशेषता है कि यदि पहला आर्गुमेंट स्ट्रिंग है तो अगला भी स्ट्रिंग हो जायेगा।

डाटा मेम्बेर्स इनिशियलाइजेसन (Data Members Initialization):

किसी भी ऑब्जेक्ट के डाटा मेम्बेर्स की एक वैल्यू होती है यदि वैल्यू इनिशियलाइज़ नहीं की है तो डिफ़ॉल्ट वैल्यू आती है। जावा में डिफ़ॉल्ट वैल्यू निम्नुसार होती है

टाइप	डिफ़ॉल्ट वैल्यू
Byte	0
Short	0
int	0
long	0
float	0.0 F
double	0.0
char	'\0'
boolean	false

सभी ऑब्जेक्ट की इनिशियल वैल्यू null होती है।

```
/**
*  Example-12 Initial Values Application
 */

class Values{
int a;
float b;
```

```
String s;
Car c;
}
 class InitialValues {
 Public static void main ( String [ ] args){

Values v = new Values ( );

System.out.println(v.a);
System.out.println( v.b);
System.out.println( v.s);
System.out.println( v.c);
}
}
```

इसका आउटपुट

```
0
0.0
null
null
```

यहाँ पर वैल्यूज इनिशियलाइज़ नहीं की थीं इसलिए डिफ़ॉल्ट वैल्यूज आयी है। यदि हम चाहें तो वैल्यूज को निम्नुसार इनिशियलाइज़ किया जा सकता है।

```
/**
*  Example-13 Initial Values Application-2
 */

class Values{
int a=2;
float b = inverse(a);
String s;
Car c;
Values (String st){
s = st;
}
public float inverse( int val){
return 1.0F / val;
}
}

 class InitialValues2 {
 public static void main ( String [ ] args){

Values v = new Values (“Hello” );
```

```
System.out.println(v.a);
System.out.println(v.b);
System.out.println(v.s);
System.out.println(v.c);
}
}
```

इसका आउटपुट निम्न आएगा

```
2
0.5
Hello
null
```

कीवर्ड (Keyword) "this": जावा में एक कीवर्ड है "this". यह कीवर्ड मेथड के अन्दर रिसीवर ऑब्जेक्ट (Receiver object) को उद्घृत करता है। यह मेथड में दो प्रकार से उपयोग होता है। पहला मेथड में रिसीवर ऑब्जेक्ट के रिफरेन्स को रिटर्न करने में उपयोग होता है एवं दूसरा constructor को किसी अन्य constructor से कॉल करने में भी "this"का उपयोग होता है। उदाहरण के लिए

```
public Car setColor (String col){
 color = col;
 return this;
}
```

यह मेथड कार को एक रिफरेन्स को रिटर्न करती है एवं रिसीवर ऑब्जेक्ट के रिफरेन्स को एक वैल्यू रिटर्न करती है। उपर्युक्त मेथड को निम्न प्रकार से उपयोग करते हैं

```
 Car c1 = new CAR (("WagonR","Amit Kumar",18);
System.out.println(c1.setColor ("White").getInitials());
System.out.println(c1.getColor);
```

इस मेथड में एक मेसेज setColor(), c1 को भेजा गया। इस मेथड ने रिसेप्टर ऑब्जेक्ट को रिफरेन्स रिटर्न किया जो की c1 है। उसके बाद एक मेसेज getInitials() c1 को भेजा गया और निम्न आउटपुट आया

```
A.K.
White
```

कीवर्ड "this" का उपयोग करके एक constructor को अन्य constructor से कॉल करने का उदाहरण

```
/**
*  Example-14 of Cars with static data members  Application
 */

class Car {
String Model;
String Owner;
String Registration_number;
int AverageMileage;


Car(String mod, String own, int am) {
Model = mod;
Owner =own;
REGISTRATION_NUMBER =”Not Known”;
AverageMileage = am;
}

Car(String mod, String own,String Regn, int am) {
Model = mod;
Owner =own;
REGISTRATION_NUMBER = Regn;
AverageMileage = am;
}
```

इस उदाहरण में दूसरा constructor को पहले से कॉल करके प्रोग्राम को छोटा किया जा सकता है

```
Car(String mod, String own, int am) {
this(mod,own,am);
REGISTRATION_NUMBER = Regn;
}
```

इसमें पहले constructor को दूसरे constructor में कॉल किया है और उसमें पहले constructor के डाटा मेम्बेर्स में एक डाटा मेम्बर REGISTRATION_NUMBER उसकी वैल्यू Regn के साथ जोड़ दिया। इस तरह प्रोग्राम कोड भी छोटा हो गया और गलती की सम्भावना भी कम हो गयी।

काम्प्लेक्स नंबर क्लास (Complex Number class) एक उदाहरण :

यहाँ पर हम को यह समझना है कि काम्प्लेक्स नंबर की एप्लीकेशन के लिए कैसे काम्प्लेक्स नंबर class को लिखा जाता है। इसे निम्न उदाहरण से समझते हैं

```
/**
*  Example-15 of Complex number class Application
 */

class TestComplex {
public static void main ( String [ ] args){

 Complex a1 = new Complex (1.24,2.45);
Complex a2 = new Complex (3.22,5.43);

Complex c = a1.add(a2)
Syste.out.prinln("a1+a2="+c.getReal()+","+c.getImaginary());

 Complex d = c.sub(a2)
Syste.out.prinln("c-a2="+d.getReal()+","+d.getImaginary());
}
}
```

इस उदाहरण में दो काम्प्लेक्स नंबर a1 और a2 बनाये गए और उनमें वैल्यूज के रियल और इमेजिनरी पार्ट्स को दिया गया। उसके बाद एक और काम्प्लेक्स नंबर c बनाया गया जिसमें पहले दो काम्प्लेक्स नंबर का जोड़ सौंपा गया एवं जोड़ के रियल और इमेजिनरी पार्ट्स को प्रिंट किया गया। उसके बाद c नंबर में से a2 को घटाया गया एवं उसे एक नए काम्प्लेक्स नंबर d को सौंप दिया गया एवं d नंबर के रियल और इमेजिनरी पार्ट्स को भी प्रिंट किया गया। इसका आउटपुट :

```
 a1+a2 = 4.46,7.88
c-a2 =1.24,2.45
```

हम जानते हैं कि काम्प्लेक्स नंबर दो पार्ट्स से मिलकर बनता है रियल और इमेजिनरी। इसका मतलब है कि काम्प्लेक्स नंबर की class में दो डाटा मेम्बेर्स चाहिए। एक डाटा मेम्बर काम्प्लेक्स नंबर के रियल पार्ट को स्टोर करने के लिए एवं दूसरा डाटा मेम्बर इमेजिनरी पार्ट को स्टोर करने के लिए। यह व्यवस्था बनाने के लिए हम एक constructor बनायेंगे जो arguments की सहायता से काम्प्लेक्स नंबर के दोनों पार्ट्स को इनिशियलाइज़ करेगा। इसके साथ ही दो मेथोड्स डिफाइन करेंगे एक रियल

पार्ट को प्रदान करेगा एवं दूसरा इमेजिनरी पार्ट को प्रदान करेगा। निम्न उदाहरण का अवलोकन करें

```
/**
*  Example-16 of Complex number class
 */

 public class Complex {
double = real;
double = imag;

/**
*यह constructor एक काम्प्लेक्स नंबर उसके रियल एवं इमेजिनरी पार्ट्स के साथ बनाएगा।
 */

Complex(double r,double i){
 real = r;
 imag  = I;
}

/**
*यह मेथड रियल पार्ट को रिटर्न करने के लिए
*/

public double getReal(){
 return real;
}

 /**
*यह मेथड इमेजिनरी पार्ट को रिटर्न करने के लिए
*/

public double getImaginary(){
 return imag;
}
```

उदाहरण 15 में दो और मेथोड्स उपयोग में लायी गयी। एक मेथड जोड़ के लिए (a1.add(a2)) एवं दूसरी मेथोस घटाने के लिए (c.sub(a2))। इन दोनों मेथोड्स को देखने पर पता चलता है कि दोनों मेथोड्स एक आर्गुमेंट प्राप्त करती हैं और दूसरे नंबर से जोड़ या घटाना करती हैं ।

```
Complex c = a1.add(a2);
```
इसमें एक आर्गुमेंट a2 प्राप्त करता है और a1 के पार्ट में जोड़ देता है।
इसका मतलब हुआ कि यह मेथड एक नया काम्प्लेक्स नंबर बनाएगी और उसे परिणाम के तौर पर रिटर्न करेगी। यहाँ पर ध्यान देने वाली बात है कि रियल पार्ट से रियल पार्ट एवं इमेजिनरी पार्ट इमेजिनरी पार्ट का ऑपरेशन होगा।
निम्न उदाहरण में हम देखेंगे कि कैसे दो मेथड जोड़ एवं घटाने के लिए लिखी जा सकती है

```
/** ये मेथड एक नया काम्प्लेक्स नंबर को रिटर्न करती है
* जो रिसेप्टर और आर्गुमेंट से प्राप्त काम्प्लेक्स नंबर का जोड़ का परिणाम होता है
*/

public Complex add(Complex a) {
return new Complex (real+a.real, imag+a.imag);
}

/** ये मेथड एक नया काम्प्लेक्स नंबर को रिटर्न करती है
* जो आर्गुमेंट से प्राप्त काम्प्लेक्स नंबर को रिसेप्टर में से घटाने का परिणाम होता है
*/

public Complex sub(Complex a) {
return new Complex (real-a.real, imag-a.imag);
}
```

उपर्युक्त मेथड में new कीवर्ड से काम्प्लेक्स नंबर का नया इंस्टांस बनाया उसके बाद constructor को कॉल करके उसे इनिशियलाइज़ किया और अंत में उसे रिटर्न किया।
यदि हम एक मेथड addReal( ) डिफाइन करना चाहते हैं जो सिर्फ रिसेप्टर का रियल पार्ट को arguments से प्राप्त रियल पार्ट के साथ जोड़े। इस मेथड में ध्यान रखना है कि मेथड रिसेप्टर को संशोधित करे। इसके प्रयोग का उदाहरण :

```
a1.addReal(2.0);
```

यदि हम उदाहरण 15 को देखें तो उपर्युक्त मेथड के रन होने के बाद दो वैल्यूज 3.24 रियल पार्ट और 2.45 इमेजिनरी पार्ट a1 में मिलाता है।

```
a1.addReal(2.0).addReal(3.12);
```

इस स्टेटमेंट के एक्सिक्युसन में addReal() मेथड को दो बार कॉल करना है। पहली बार में a1 के रियल पार्ट में 2.0 का जोड़ होगा और यह जोड़ रिसेप्टर ऑब्जेक्ट (करेंट

ऑब्जेक्ट) को रिफरेन्स का रिटर्न होगा जिससे addReal( ) मेथड को दोबारा कॉल करने पर मेथड उसी काम्प्लेक्स नंबर को नयी वैल्यू के साथ ऑपरेट करे।
हम जानते है कि "this" रिसेप्टर ऑब्जेक्ट का रिफरेन्स है इसलिए मेथड को निम्न प्रकार से भी लिख सकते हैं।

```
/** यह मेथड में arguments से प्राप्त वैल्यू की वृद्धि रियल पार्ट में करती है।
*  यहाँ ध्यान देना होगा कि ये मेथड रिसेप्टर को भी संशोधित करती है
*/
public Complex addReal(double c) {
 real +=c;
return this;
}
```

```
Complex b = a;
```

हम इस स्टेटमेंट पर विचार करें तो पता चलेगा कि a नया काम्प्लेक्स नंबर न हो के सिर्फ a का जो रिफरेन्स ऑब्जेक्ट है b उसको रेफ़र करता है इसका मतलब है कि b की वैल्यू बढाने पर a की भी वैल्यू बढेगी।

अतः यदि हम एक नया काम्प्लेक्स नंबर बनाना चाहते हैं तो हमें constructor का उपयोग निम्नानुसार करना होगा

```
Complex b = new Complex (a);
```

एक ऐसा नया constructor जो एक काम्प्लेक्स नंबर को आर्गुमेंट की तरह से प्राप्त करे, बनाने का दिलचस्प तरीका

```
/** यह constructor में जो आर्गुमेंट से प्राप्त काम्प्लेक्स नंबर है उसकी कॉपी से एक नया काम्प्लेक्स नंबर बनाने की *विधि है
*/

Complex (Complex b) {
This(b.real,b.imag);
}
```

यह constructor एक काम्प्लेक्स नंबर को आर्गुमेंट से प्राप्त करता है और this के द्वारा पूर्व में डिफाइन constructor को कॉल करता है।

इनहेरिटेंस(Inheritance):

एक class से दूसरी class बनाने के लिए इनहेरिटेंस का उपयोग होता है पहली class को बेस क्लास (Base class) और दूसरी को सबक्लास (Subclass) कहते हैं।

एक नयी सबक्लास जो की बेसक्लास की तरह हो को extends कीवर्ड के साथ बेसक्लास का नाम लिखकर बनाते हैं।

उदाहरण के लिए यदि हम पूर्व में डिफाइन की हुई class Car बेसक्लास मानकर extends करें तो यह नयी सबक्लास Electric Car को डिफाइन करती है। यहाँ हम कुछ नए डाटा मेम्बेर्स जो कि पुरानी class Car में नहीं हैं को जोड़ कर और ज्यादा इनफार्मेशन को रख सकते हैं।

उदाहरण के लिए

```
Class ElectricCar extends Car{
  String powerOfBattery;
 boolean Hybrid = true;
}
```

इस ElectricCar के इंस्टांस में 6 डाटा मेम्बेर्स होंगे। चार डाटा मेम्बेर्स बेसक्लास से इनहेरिट होंगे और दो डाटा मेम्बेर्स इसी class में डिफाइन किये गए हैं। इसके डाटा मेम्बेर्स
होंगे:Model,Owner,REGISTRATION_NUMBER,AverageMileage,powerOfBattery और Hybrid। इस class की डिफ़ॉल्ट ElectricCar Hybrid टाइप की होगी।

कंसट्रकटर (Constructor):

यहाँ हम Constructor को बेसक्लास से डिफाइन करते हैं

```
ElectricCar (String mod, String own, String Regn, int am, String pob){
super(mod,own,Regn,am)
powerOfBattery = pob;
}
```

इस Constructor को पहले से डिफाइन class Car के Constructor से बनाया गया है और उसमें एक डाटा मेम्बर powerOfBattery जोड़ा गया है पहले चार डाटा मेम्बेर्स पहले से डिफाइन Constructor से हैं अतः उन्हें फिर से डिफाइन करना जरूरी नहीं है। उन डाटा मेम्बेर्स को पुरानी बेसक्लास से super() मेथड से कॉल करते हैं एवं नए डाटा मेम्बर को यहीं पर डिफाइन करते हैं।

यहाँ पर ध्यान रखना होगा कि मेथड super() , class की बॉडी में पहला इंस्ट्रक्शन होना चाहिए। यदि ऐसा नहीं हुआ तो जावा कम्पाइलर बिना पैरामीटर्स के super() मेथड को कॉल करेगा। extends के द्वारा पूर्व class के पदानुक्रम में नयी class बनती है और नए डाटा मेम्बेर्स एवं मेथोड्स को जोड़ कर नयी class का उपयोग किया जा सकता है।

Constructor का उपयोग करके ElectricCar class को भी डिफाइन निम्नानुसार किया जा सकता है

```
ElectricCar ec;
ec= new ElectricCar(“Amaze”, ”Yashasvita Shastri”,
”MPO4AS5136”, 18,”12v”);
```

मेथड (Method):

किसी भी subclass में उसके व्यव्हार को उल्लेखित करने के लिए नयी मेथड को डिफाइन कर सकते हैं। इसके आलावा जो भी मेथोड्स पूर्व पदानुक्रम में डिफाइन हैं उन्हें भी कॉल कर सकते हैं।

जब भी मेसेज ऑब्जेक्ट को भेजा जाता है तो पहले जिस class (रिसेप्टर ऑब्जेक्ट) में मेथड है उसमें ही मेथड को सर्च किया जाता है यदि यहाँ पर मेथड नहीं मिलती है तब उसे ऊपर के पदानुक्रम में जो class है उसमें सर्च करते हैं यह प्रक्रिया तब तक चलती जब तक मेथड मिल नहीं जाती।

इनहेरिटेंस का उपयोग करके रिलेटेड class में डेवलप्ड कोड का पुनः उपयोग (रीयूस) कर सकते हैं कुछ केसेस में subclass में मेथड का व्यव्हार बदलना पड़ता है इस स्थिति में मेथड को पुनः डिफाइन करना होता है। जैसा की हम जानते हैं की मेथड सर्च रिसेप्टर class से शुरू होती है और हमेशा सबसे विशिस्ट मेथड का चयन होता है।

अब हम एक मेथड getInitials() को रीयूस के माध्यम से class ElectricCar में उपयोग करेंगे। क्योंकि getInitials() मेथड डाटा मेम्बर owner पर कार्य करती है और यह डाटा मेम्बर दोनों class के इंस्टांस में है। इसलिए getInitials() को class ElectricCar में डिफाइन करने की जरूरत नहीं है। इसे हम निम्नानुसार उपयोग कर सकते हैं :

```
System.out.println(ec.getInitials());
```

यहाँ पर ec, ElectricCar का एक इंस्टांस है जिसे ऊपर डिफाइन किया गया है।

यहाँ पर इसी तरह से equals() मेथड का उपयोग नहीं किया जा सकता है क्योंकि ElectricCar को चेक करने के लिए दो डाटा मेम्बेर्स लगेंगे जो पूर्व class में डिफाइन नहीं है। एक तरीका है equals() मेथड को रीयूस करने का और वह है कि हम चार डाटा मेम्बेर्स को पूर्वे class Car से और दो डाटा मेम्बेर्स को ElectricCar से equals() मेथड का प्रयोग करें।

```
public boolean equals(ElectricCar ec){
```

```
return super.equals(ec) &&
powerOfBattery.equals(ec.powerOfBattery) && Hybrid
==ec.Hybrid;
}
```

यहाँ पर equals से दो डाटा मेम्बेर्स powerOfBattery एवं Hybrid की तुलना की गयी है और बचे हुए 4 डाटा मेम्बेर्स की तुलना बेस class से equals() मेथड को super के द्वारा कॉल करके की गयी है। इस तरह से equals() मेथड को पुनः उसी नाम से डिफाइन किया गया

यहाँ यह ध्यान देना है कि इस नयी equals() मेथड में बेस class से equals() मेथड को कॉल करके इसका पार्ट बनाया गया है। जब भी मेथड को बेस class से super के द्वारा कॉल करते हैं तब इसे पुरी बॉडी में कहीं भी यूज़ कर सकते हैं।

उपर्युक्त मेथड को निम्न प्रकार से भी लिखा जा सकता है

```
public boolean equals(ElectricCar ec){

return (Model.equals(ec. Model)&& Owner.equals(ec.Owner)&&
REGISTRATION_NUMBER.equals(ec REGISTRATION_NUMBER)&&
AverageMileage ==ec.AverageMileage && powerOfBattery.equals
(ec.powerOfBattery)&& Hybrid ==ec.Hybrid);
}
```

मेथड को इस प्रकार से लिखने पर उसे बेस class से कॉल करने की आवश्यकता नहीं है।

हमेशा यह जरूरी नहीं है कि रिडिफाइंड मेथड को बेस class से कॉल किया जाये। उदाहरण के लिए Car class में एक static मेथड trafficRules( ) डिफाइंड है इसे हम ElectricCar class में रिडिफाइंड कर सकते है

```
public static String trafficRules( ){
return “Number plate for Electric Car should be green ”;
}
```

एक नयी मेथड भी हम डिफाइन कर सकते हैं उदाहरण के लिए Hybrid की वैल्यू सेट करने एवं उसे चेक करने के लिए हम निम्नानुसार मेथड लिख सकते हैं

```
public void setHybrid(){
 Hybrid = false;
```

```
}

public boolean isHybrid(){
return Hybrid;
}
```

यहाँ पर ध्यान रखना है कि यह मेथड subclass में डिफाइन है अतः setHybrid का मेसेज subclass में भेज सकते हैं पर बेस class में नहीं भेज सकते।

एक उदाहरण ElectricCar class की एप्लीकेशन के लिए

```
/**
* Electric Car class का उदाहरण अभी तक के विकल्पों के साथ
*/

Class TestElectricCar {

Public static void main (String [] args){

ElectricCar ec1,ec2;

ec1 = new ElectricCar ("Brezza","Aseem Shrivastava",
"MP045136",14,"12V");

ec2 = new ElectricCar ("Brezza"," Aseem Shrivastava",
"MP045136",18,"12V");

ec2.setHybrid();

System.out.println(ec1.getInitials());
 System.out.println(ec1.equals(ec2));
System.out.println(ec2. trafficRules( ));
System.out.println(ec2. isHybrid( ));
}

}
```

इस उदाहरण का आउटपुट निम्न आएगा

```
A.K.
false
 Number plate for Electric Car should be green
false
```

Instanceof and getClass मेथोड्स:

मेथड instanceof()बूलियन वैल्यू रिटर्न करती है और यह बताती है कि ऑब्जेक्ट किस विशिष्ट class का है। वहीं getClass() मेथड एक स्ट्रिंग रिटर्न करती है और ऑब्जेक्ट जिस class का इंस्टांस है उस class का नाम बताती है।
उदाहरण के लिए

```
 /**
* Test class Application
*/

class TestClass {

public static void main (String[]args){

Car c1;

ElectricCar ec1;

c1 = new Car ((“WagonR”,”Amit Kumar”,18);

ec1 = new ElectricCar (“Brezza”,” Aseem Shrivastava”, ”MP045136”,18,”12V”);

System.out.println(c1.getClass());
System.out.println(ec1.getClass( ));
System.out.println(c1 instanceof Car);
System.out.println(ec1 instanceof Car);
System.out.println(c1 instanceof ElectricCar);
System.out.println(ec1 instanceof ElectricCar);

}

}
```

इस उदाहरण का आउटपुट निम्न होगा :

```
class Car
class ElectricCar
true
true
false
true
```

यह आउटपुट बहुत ही दिलचस्प है। इसमें पहले दो आउटपुट तो साधारण उस class का नाम बता रहे हैं जहां का इंस्टांस ऑब्जेक्ट है। तीसरा आउटपुट बता रहा है की class Car का इंस्टांस c1 है। चौथा आउटपुट बता रहा है कि ec1 भी class Car का इंस्टांस है और यह इसलिए की कोई भी subclass का इंस्टांस subclass के साथ साथ हायर classes का भी इंस्टांस होता है परन्तु इसका उल्टा नहीं हो सकता है जो पाचवे आउटपुट में देख सकते हैं। यहाँ पर class ElectricCar का इंस्टांस c1 नहीं है।

पैकेजेस (Packages) :

पैकेजेस एक तरह का स्ट्रक्चर है जिसमें classes को सुव्यवस्थित करते हैं एक पैकेज में कई classes जो किसी उद्देश्य या विरासत(inheritance) द्वारा एक हो सकती हैं सिस्टम की स्टैण्डर्ड classes एक पैकेज में व्यवस्थित होती हैं उदाहरण के लिए एक class Date जावा में उपलब्बध है जिसे हम अपनी class में date के लिए उपयोग कर सकते हैं। जावा में यह class Date जावा के एक पैकेज java.util में है।

कम्पाइलर को यह बयाने के लिए कि हम class Date को अपनी class में उपयोग करना चाहते हैं, हमें इस class Date के पैकेज java.util को import स्टेटमेंट द्वारा अपनी class में importकरना होगा। जिसका सिंटेक्स निम्नानुसार है

```
import java.util.Date;
```

और यदि हम किसी पैकेज की एक से ज्यादा classes का उपयोग अपने प्रोग्राम में करना चाहते है तब हमें केवल पैकेज का नाम उल्लिखित करना होगा। उधाहरण के लिए :

```
import java.util.*;
```

इसका मतलब है कि हम util पैकेज की एक से ज्यादा class एस का उपयोग कर सकते हैं।

```
/**
*An application of Date class
*/
```

```
import java.util.*;

class TestDate{

public static void main (String[]args){
System.out.println(new Date());
}
}
```

इसके आउटपुट में जिस दिन यह प्रोग्राम एक्सिक्युट होगा उस दिन की date प्रिंट होगी ।

कोई भी नया पैकेज डिफाइन करने के लिए Package स्टेटमेंट का उपयोग निम्न प्रकार से करना होगा

```
Package LMVehicle;
```

सभी classes जो इस फाइल में होंगी वो vehicle पैकेज के अंतर्गत होंगी । इसके आलावा कुछ फाइल और भी पैकेज में होंगी । कुछ ऐसी फाइल जो दूसरे पैकेज में हैं उन्हें भी import करके इस पैकेज का हिस्सा बना सकते हैं ।

एक्सेस कंट्रोल (Access Control):

जावा में एक और सुविधा है कि किसी मेथड या वेरिएबल को किसी दूसरी class से उपयोग करने को नियंत्रित किया जा सकता है । यह नियंत्रण तीन संसोधाकों public,private एवं protected के द्वारा किया जाता है । किसी भी मेथड या वेरिएबल के लिए डिफ़ॉल्ट access होता है जिससे सभी classes जो एक ही पैकेज में हैं, का फुल access देता है । हम अभी तक ज्यादातर उदाहरणों में इसका उपयोग कर रहे हैं । फुल access का मतलब है कि दूसरी classe के सभी डाटा मेम्बेर्स और मेथोड्स का access कर सकते हैं । उदाहरण के लिए हम ElectricCar class के Hybrid डाटा मेम्बर को TestElectricCar classe से सेट निम्नानुसार कर सकते हैं

```
ec2.setHybrid();
```

और दूसरा तरीका है

```
ec2.Hybrid= true;
```

प्रायः एन्केप्सुलेसन (encapsulation) को बनाये रखने के लिए हम डाटा मेम्बेर्स या मेथोड्स को सीधा access नहीं देना चाहते हैं। ऐसी स्थिति में private संशोधक का उपयोग करते हैं। इस संशोधक के उपयोग से डाटा मेम्बेर्स का access सिर्फ उन्हीं मेथड से हो सकता है जो इसी classe में हैं । उदाहरण के लिए डाटा मेम्बेर्स powerOfBattery एवं Hybrid को private डिफाइन करते हैं

```
Class ElectricCar extends Car{
  private String powerOfBattery;
 private boolean Hybrid = true;
}
```

इसमें private डिफाइन करे के बाद डाटा मेम्बर Hybrid को किसी दूसरी classe से access नहीं किया जा सकता है । अतः केवल ElectricCar class द्वारा ही setHybrid() मेथड के सहायत से Hybrid की वैल्यू को सेट कर सकते हैं।

आम तौर पर डाटा मेम्बर को private डिफाइन करते हैं जिससे डाटा मेम्बेर्स को केवल उसी class की मेथोड्स से पर्तिवार्तित किया जाता है।

यह भी सत्य है कि abstract डाटा टाइप (ADT) की ये एक विशेष प्रॉपर्टी है जिसे एन्केप्सुलेसन (encapsulation) कहते हैं।

public संशोधक (modifier) डाटा मेम्बेर्स को फुल access बिना किसी प्रतिबन्ध के देता है। प्रायः मेथोड्स में public का उपयोग करते हैं इससे किसी भी class के ऑब्जेक्ट को से मेसेज भेजा जा सकता है।

और अंत में protected संशोधक डाटा मेम्बेर्स का access subclass और किसी भी class जो उसी पैकेज की हो को देता है।

final एंड abstract:

final एंड abstract ये दो संशोधक है जो मेथड और class के लिए उपयोग में लाये जाते हैं। final मेथड को subclass में भी पुनः डिफाइन या परिवर्तन नहीं किया जा सकता है। इसका मतलब है की मेथड को final डिफाइन कर दिया है तो अब ये मेथड ऐसे ही उपयोग होगी बिना किनी परिवर्तन के। और यदि किसी class को final डिफाइन कर दिया तो उस class की subclass नहीं बन सकती।

abstract संशोधक सिर्फ मेथड को अमूर्त रूप में डिफाइन करता है इसका मतलब है abstract मेथड को subclass में पुनः डिफाइन करना आवश्यक है क्योंकि बिना डिफाइन किये इसका उपयोग नहीं कर सकते है । इसी तरह abstract class के इंस्टांस नहीं बनाये जा सकते हैं जबकि इसकी subclass बनायीं जा सकती हैं और abstract class की subclass के इंस्टांस बन सकते हैं । इसे एक उदाहरण से समझते हैं । माना कि हम एक class बना रहे हैं जिसका नाम computerSystem है । जैसा की हम जानते है कि computerSystem बहुत प्रकार के होते हैं जैसे डेस्कटॉप , लैपटॉप , सर्वर और सुपर कंप्यूटर आदि । इस सभी प्रकार के कंप्यूटर में कुछ अवयव Ram,ROM, Mother Board, I/O devices, Microprocessor etc. में होते हैं और इन सामान्य अवयवों के साथ साथ कुछ विशेष अवयव किसी भी एक प्रकार के कंप्यूटर में होते हैं । इसका प्रोग्राम बनाने की लिए हम सामान्य अवयवों के साथ एक class computerSystem बनाते हैं और इस class को हमने इसके लिए सशोधक abstract का उपयोग कर abstract class computerSystem बना दिया । इस प्रक्रिया से इस उदाहरण के हिसाब से चार subclass डेस्कटॉप , लैपटॉप , सर्वर और सुपर कंप्यूटर के लिए बनेंगी । अब हम प्रत्येक प्रकार के कंप्यूटर की subclass में सामान्य अवयव एवं इस प्रकार के विशेष अवयव सभी को डिफाइन करेंगे ।

इसके बाद हम जरूरत के हिसाब के किसी भी प्रकार के कंप्यूटर की subclass के इंस्टांस बनाकर उस टाइप के कंप्यूटर को व्यवस्थित तरीके से उपयोग कर सकते हैं। यहाँ ये बात भी ध्यान देने की है कि हमेश जब भी हम कंप्यूटर की इनफार्मेशन या तो स्टोर करेंगे या उपयोग करेंगे दोनों ही स्थिति में किसी एक प्रकार के कंप्यूटर की इनफार्मेशन के बारे में करेंगे कभी भी हम सामान्य कंप्यूटर की बात नहीं करेंगे अतः computerSystem class के इंस्टांस की जरूरत ही नहीं पड़ेगी और इसलिए हमने इसे abstract डिफाइन कर दिया है । यही एक अच्छी डिजाईन की प्रक्रिया मानी जाती है ।

इसे हम एक उदाहरण से समझते हैं :

एक computerSystem abstract class बनाते हैं जिसमें हम जनरल अवयवों को सिर्फ डिफाइन करेंगे क्योंकि हर प्रकार के सिस्टम में उसके हिसाब से विशिष्ट वैल्यू के लिए एक subclass बनायेंगे । subclass के अन्दर कुछ abstract मेथोड्स भी बनायेंगे जिससे उस विशिष्ट प्रकार के सिस्टम का व्यवहार डिफाइन हो सके ।

```
/**
*Computer System Class
```

```
*/

abstract class computerSystem {

String type;

int noOfSoftware = 0;

computerSystem (String t){
System.out.println("computerSystem constructor");
type = t;
}

final public void Software(){
 noOfSoftware ++;
}


final public int getSoftware(){
 return noOfSoftware;
}

abstract public void initialize();
abstract public void on();
abstract public void off();
}
```

इस class की कोई भी subclass मेथड Software() को पुनः डिफाइन नहीं कर पायेगा क्योंकि यह मेथड यहाँ पर final डिफाइन है । और class computerSystem का कोई भी इंस्टांस नहीं बनेगा क्योंकि यह abstract class है।

अब subclass बनाते हैं

```
 /**
*Desktop Computer System Class
*/

class Desktop extends computerSystem{

int usbports;

Desktop (String t, int p, int hd){
super (t);
usbport =p;
noOfHardDisk = hd;
```

```
System.out.println("Desktop constructor");
}

public void initialize (){
System.out.println("initialise method for Desktop");
// specific code to initialise a Desktop
}


public void on (){
System.out.println("on method for Desktop");
// specific code to initialise a Desktop
}


public void off (){
System.out.println("off method for Desktop");
// specific code to initialise a Desktop
}
}
```

यह subclass Desktop computer सिस्टम को डिफाइन करने के लिए बनायीं है। इस subclass में एक constructor बनाया है जो तीन आर्गुमेंट्स प्राप्त करेगा type,port और HardDisk। इसमें से data member type बेस class में डिफाइन है और बचे हुए दो data members port एवं HardDisk इसी subclass में डिफाइन हैं। यहाँ पर subclass का constructor बेस class के constructor को super के द्वारा कॉल कर रहा है। यहाँ पर हम तीनों मेथोड्स initialize, on और off का कोड विस्तृत में नहीं लिख रहे है इसे प्रोग्रामर अपने हिसाब से लिख सकता हैं।

```
/**
* Server Computer System Class
*/

class Server extends computerSystem{

int usbports;

Server (String t, int p, int pr){
super (t);
usbport =p;
noOfProcessors = pr;
System.out.println("Server constructor");
}
```

```
public void initialize (){
System.out.println(“initialise method for Server”);
// specific code to initialise a Server
}


public void on (){
System.out.println(“on method for Server”);
// specific code to initialise a Server
}


public void off (){
System.out.println(“off method for Server”);
// specific code to initialise a Server
}
}
```

यह subclass Server सिस्टम को डिफाइन करने के लिए बनायीं है । इस subclass में एक constructor बनाया है जो तीन आर्गुमेंट्स प्राप्त करेगा type,port और noOfProcessors । इसमें से data member type बेस class में डिफाइन है और बचे हुए दो data member port एवं noOfProcessors इसी subclass में डिफाइन हैं । यहाँ पर subclass का constructor बेस class के constructor को super के द्वारा कॉल कर रहा है । यहाँ पर हम तीनों मेथोड्स initialize, on और off का कोड विस्तृत में नहीं किख रहे है इसे प्रोग्रामर अपने हिसाब से लिख सकता हैं।

```
/**
* Laptop Computer System Class
*/

class Laptop extends computerSystem{

int usbports;

Laptop (String t, int p){
super (t);
usbport =p;
System.out.println(“Laptop constructor”);
}
```

```
public void initialize (){
System.out.println("initialise method for Laptop");
// specific code to initialise a Laptop
}


public void on (){
System.out.println("on method for Laptop");
// specific code to initialise a Laptop
}


public void off (){
System.out.println("off method for Laptop");
// specific code to initialise a Laptop
}
}
```

यह subclass Laptop सिस्टम को डिफाइन करने के लिए बनायीं है । इस subclass में एक constructor बनाया है जो दो आर्गुमेंट्स प्राप्त करेगा type और port इसमें से data member type बेस class में डिफाइन है और दूसरा data member port इसी subclass में डिफाइन हैं । यहाँ पर subclass का constructor बेस class के constructor को super के द्वारा कॉल कर रहा है । यहाँ पर हम तीनों मेथोड्स initialize, on और off का कोड विस्तृत में नहीं किख रहे है इसे प्रोग्रामर अपने हिसाब से लिख सकता हैं।

अब हम एक टेस्ट class बनाकर इस सब classes का उपयोग को समझेंगे । यहाँ पर हम subclass का इंस्टांस बना रहे हैं क्योंकि ये abstract डिफाइन नहीं है इसलिए इसका इंस्टांस बन सकता है ।

```
/**
*Test Computer System Application
*/

class TestCompuerSystem {

public static void main(String[] args){

 Server netserver = new Server ("Networ server", 10);

netserver.initialize();
netserver.on();
netserver.off();
```

```
}
}
```

इसका आउटपुट निम्न होगा

```
computerSystem constructor
Server constructor
initialise method for Server
on method for Server
off method for Server
```

यहाँ आर ध्यान से देखने पर पता चलता है की पहला constructor बेस क्लास का एवं दूसरा constructor subclass का एक्सिक्युट हुआ है। और एक बात कि बेस क्लास की मेथोड्स को अनदेखा किया गया है ।

पोलिमोर्फिसम (Polymorphism) :

पोलिमोर्फिसम का मतलब होता है कि कई फॉर्म्स । जावा में पोलिमोर्फिसम एक महत्वपूर्ण प्रॉपर्टी है यहाँ पर विभिन्न ऑब्जेक्ट्स समान मेसेज पर उत्तर देते हैं।

```
 /**
*Test Computer System Application
*/

class TestCompuerSystem {

public static void main(String[] args){

computerSystem [] system = new computerSystem[2];

system[0] = new Laptop(“This is my first Laptop”);

system[1] = new Server(This server if powerful”);

for (i=0;i<2;i++){
system [i].initialize();
}


for (i=0;i<2;i++){
system [i].on();
}
```

```
for (i=0;i<2;i++){
system [i].off();
}

}
}
```

यहाँ पर एक रोचक तथ्य आय कि computerSystem() class abstract डिफाइन है तब उसका इंस्टांस कैसे बना। हम सभी जानते है कि कोई भी subclass का इंस्टांस बेस class का भी इन्स्तास होता है यहाँ पर यही लॉजिक का उपयोग किया गया एवं दो इंस्टांस एक लैपटॉप का एवं एक सर्वर का बनाया गया जिसे इसकी बेस class computerSystem() class का बताया गया है । यहाँ पर तीन मेथोड्स इनिशियलाइज़, ओन एवं ऑफ तीनो को मेसेज के अनुसार समान इंटरफ़ेस के द्वारा उपयोग किया जा सकता है।

इंटरफेस (Interfaces):

उपर्युक्त उदाहरण में हमने देखा कि जब हम एक से ज्यादा class के लिए एक से डाटा मेम्बेर्स या एक सी मेथोड्स का उपयोग करना चाहते है तब हम एक class abstract डिफाइन करते है एवं उसकी कई subclass बनाते हैं। इसी विचार को आगे बढ़ाते हुए हम interface का उपयोग करते है। interface एक ऐसी class है जिसमें सभी डाटा मेम्बेर्स एवं static और final होते हैं एवं सभी मेथोड्स पब्लिक टाइप की होती है और उनकी बॉडी नहीं होती। interface का कोई भी इंस्टांस नहीं बनता।

interface के सभी डाटा मेम्बेर्स या फ़ील्ड्स एक नियत वैल्यू ही देते है क्योंकि वे static और final होते हैं interface की मेथोड्स केवल एक व्यवहार परिभाषित करती हैं। कीवर्ड implements का उपयोग करके किसी भी class में interface इम्प्लीमेंट करते हैं।

उदाहरण के लिए यदि हम दो डाटा मेम्बेर्स type एवं noOfSoftware जो कि abstract class computerSystem() में डिफाइन हैं, का उपयोग नहीं करना चाहते है तब computerSystem() एक interface की तरह निम्नानुसार डिफाइन कर सकते हैं :

```
/**
* Computer System Interface
```

```
*/

interface computerSystemInterface {

        public void initialize();
 public void on();
 public void off();
}
```

एक class Server की डिफाइन करते है जहां पर interface इम्प्लीमेंट करेंगे

```
/**
* Server Interface Application
*/

class nServer implements computerSystemInterface{

int usbports;

newServer (int p){
usbport =p;
System.out.println("Server constructor interface application
");
}

public void initialize (){
System.out.println("initialise method for Server");
// specific code to initialise a Server
}


public void on (){
System.out.println("on method for Server");
// specific code to initialise a Server
}


public void off (){
System.out.println("off method for Server");
// specific code to initialise a Server
}
}
```

अगले उदाहरण में उपर्युक्त class का उपयोग देखेंगे

```
/**
```

```
*Test Computer System Interface Application
*/


class TestServer {

public static void main (String [] args){

 nServer sever2 = new nServer(4);

server2.initialize();
server2.on();
server2.off();
}
}
```

यहाँ पर एक बात और ध्यान रखने की है कि एक class में एक से ज्यादा interface को इम्प्लीमेंट किया जा सकता है। उदाहरण के लिए :

```
 /**
* working of Server Interface Application
*/

 interface workingBehaviour{

public String getName();

public String getStatus();

}
```

अब यदि हम चाहते है कि computerSystemInterface की सभी मेथोड्स और workingBehaviour की सभी मेथोड्स को nserver में इम्प्लीमेंट करना है तब उसे निम्नानुसार किया जा सकता है।

```
/**
* Server Interface Application 2
*/

class nServer implements computerSystemInterface, workingBehaviour {

int usbports;

newServer (int p){
usbport =p;
```

```
System.out.println("Server constructor interface application
");
}

public void initialize (){
System.out.println("initialise method for Server");
// specific code to initialise a Server
}


public void on (){
System.out.println("on method for Server");
// specific code to initialise a Server
}


public void off (){
System.out.println("off method for Server");
// specific code to initialise a Server
}
}
```

जावा कम्पाइलर class की इस परिभाषा को तभी मानेगा जब दोनों interface की सभी मेथोड्स इस class में डिफाइन होंगी।

अब हमें यह ज्ञात हुआ है कि abstract classes एवं interface classes दोनों ही समान रूप से उपयोगी हैं एक तरह से दोनों ही किसी class को विशिष्ट व्यवहार को अपनाने की लिए बाध्य करते हैं। परन्तु दोनों में एक अंतर भी है और वह है कि एक class एक से जयादा interface को इम्प्लीमेंट कर सकती है पर एक subclass एक से ज्यादा class से inherit (की विरासत) नहीं हो सकती है।

अतः हम यह निष्कर्ष निकाल सकते है कि interface के माध्यम से एक से अधिक inheritance (विरासत) को इम्प्लीमेंट कर सकते हैं।

एक्सेप्शन (Exceptions):

जावा में जब कोई कोड रन करते हैं तब कुछ error आती है उसे एक्सेप्शन कहते हैं इस निम्न उदाहरण से समझा जा सकता है

```
/**
*Test Exception Application
*/
```

```
class TestException{
public static void main (String [] args){
String s = “Namaskar”;
System.out.println(s.charAt(10));
}
}
```

उपर्युक उदाहरण में स्ट्रिंग Namaskar में 10 अक्षर नहीं हैं और प्रोग्राम के हिसाब से 10 वें अक्षर को बताना है अतः प्रोग्राम रन करने पर निम्न मेसेज के साथ execution बंद हो जायेगा।

```
Exception in thread “main”
 java.lang.StringIndexOutOfBoundsException:
       -------------
            -----------
```

इस प्रकार की गलतिओं को जावा में एक्सेप्शन कहते हैं। जब हम बड़े बड़े प्रोग्राम लिखते हैं तब इस तरह की गलतिओं की संभावनाएं बड़े जाती है। इसके समाधान के रूप में try और catch स्टेटमेंट्स जावा में उपलब्द्ध हैं जिनका उपयोग निम्नानुसार करके गलतिओं की संभावनाओं को कम करते हैं।

```
/**
*Test Exception (Try-Catch) Application
*/

class TestException2{
public static void main (String [] args){
String s = “Namaskar”;
```

```
try {
System.out.println(s.charAt(10));
} catch (Exception e){
System.out.println("There is 10th position available");
}

}
}
```

इस प्रोग्राम का आउटपुट

```
There is 10th position available
```

इस उदाहरण से स्पष्ट है कि जब भी कोई एक्सेप्शन try ब्लाक में आता है तब कंट्रोल catch ब्लाक को चला जाता है और catch ब्लाक सभी एक्सेप्शन को रन करके चेक करता है। यदि हम कोई विशेष एक्सेप्शन चला चाहते है तब निम्नानुसार के लिखना होता है।

```
/**
*Test Exception (Try-Catch-2) Application
*/


class TestException3{

public static void main (String [] args){

String s = "Namaskar";

try {
System.out.println(s.charAt(10));
} catch (StringIndexOutOfBoundsException e){
System.out.println("There is 10th position available");
}

}
}
```

एक और सुविधा जावा में है वह है कि exeception ऑब्जेक्ट को भी मेसेज भेज सकते हैं। उदाहरण के लिए

```
/**
*Test Exception (Try-Catch-3) Application
*/
```

```
class TestException4{

public static void main (String [] args){

String s = “Namaskar”;

try {
System.out.println(s.charAt(10));
} catch (StringIndexOutOfBoundsException e){
System.out.println(“There is 10th position available”);
System.out.println(e.toString());
}

}
}
```

इसका आउटपुट निम्न आएगा

```
There is 10th position available
java.lang.StringIndexOutOfBoundsException:
String index out of range: 10
```

जावा में कुछ execeptions पहले से ही डिफाइन होते हैं और उनको पकड़ा जा सकता है। कुछ मामलों में catch से एक्सेप्शन पकड़ना अनिवार्य होता है।

इनपुट आउटपुट (Input Output):

जावा में Input Output का सिस्टम थोडा जटिल है और बहुत सारी classes हैं जिनका उपयोग read और write करने के लिए करना होता है। एक फायदा भी है कि Input Output को फाइल्स से ,डिवाइस से, मेमोरी से और वेबसाइट से सभी से एक ही तरह से प्रदर्शित किया जाता है।

जावा में Input Output सिस्टम java.io पैकेज में इम्प्लीमेंट किया हुआ है। यह stream के विचार पर अधारित है। Input stream (स्ट्रीम) एक डाटा सोर्स है जिसे

डाटा प्राप्त करने के लिए access किया जा सकता है। Output एक डाटा सिंक है जहां पर डाटा को लिखा जा सकता है। ये स्ट्रीम्स दो स्ट्रीम्स में में विभाजित हैं एक Byte streams एवं दूसरी character streams. Byte streams का उपयोग डाटा को छोटे टुकड़ों में जैसे byte, integers आदि में read और write किया जाता है। character streams का उपयोग character को read या write करने में किया जाता है।

अगले सेक्शन में डाटा के प्रकार के हिसाब से विभिन्न स्ट्रीम्स को उदाहरणों की सहायता से समझेंगे।

बाइट ओरिएंटेड स्ट्रीम (Byte Oriented Streams):

बाइट ओरिएंटेड स्ट्रीम के लिए दो प्रकार की स्ट्रीम्स उपयोग के लिए जावा में उपलब्द्ध हैं। एक class FileOutputStream: जिसे bytes को stream में write करने के लिए उपयोग होता है और दूसरी class FileInputStream: जिसका उपयोग bytes को stream के द्वारा read (पढने) के लिए होता है।

```
/**
*Write Bytes Application
*/

Import java.io.*;
 class WriteBytes{
public static void main (String [] args){
int data[] = {15,25,35,45,150,550};
FileOutputStream f;
try {
f = new FileOutputStream("f1.data");
for ( i=0;i<data.length;i++)
 f.write(data[i]);
f.close();
} catch (IOExceptopn e){
 System.out.println("Error with files:" + e.toString());
}
}
}
```

इस उदाहरण में FileOutputStream का इम्प्लीमेंट की प्रक्रिया बताई है। इसमें FileOutputStream का एक इंस्टांस new का उपयोग करके बनाया गया है इस इंस्टांस में फाइल का नाम f1.data आर्गुमेंट के रूप में पास किया गया। इसमें

इंटरनल ऑब्जेक्ट f एवं फाइल के बीच में सम्बन्ध इस तरह स्थापित किया गया कि जब भी write ऑपरेशन f पर किया जाता है तब डाटा फाइल में write होता है। फाइल f1.data में पूरा data, data[] array से फाइल में write मेसेज के द्वारा लिखा गया। अंत में close मेथड के द्वारा फाइल को close लिया गया। यहाँ पर ध्यान रखना होगा कि कोड को try – catch ब्लाक में लिखा गया है और यह आवश्यक है क्योंकि IOException आ सकता है और आने पर उसका समाधान हो सके।

```
/**
*Read Bytes Application
*/

import java.io.*;
 class ReadBytes{
public static void main (String [] args){
int data[] = {15,25,35,45,150,550};
FileInputStream f;
try {
f = new FileInputStream("f1.data");
int data;
while((data = f.read())!=-1)
System.out.println(data);
f.close();
} catch (IOExceptopn e){
 System.out.println("Error with files:" + e.toString());
}
}
}
```

इस उदाहरण में FileInputStream का इम्प्लीमेंट की प्रक्रिया बताई है। इसमें FileInputStream का एक इंस्टांस new का उपयोग करके बनाया गया है इस इंस्टांस को फाइल f1.data से सम्बन्ध स्थापित किया गया।

फाइल f1.data से पूरा data, read मेसेज के द्वारा पढ़ा जायेगा। यह फाइल के अंत में पहुँचने पर -1 की वैल्यू रिटर्न करेगा। अंत में close मेथड के द्वारा फाइल को close लिया गया। यहाँ पर ध्यान रखना होगा कि कोड को try – catch ब्लाक में लिखा गया है और यह आवश्यक है क्योंकि IOException आ सकता है और आने पर उसका समाधान हो सके।

इस उदाहरण का आउटपुट निम्न होगा

```
15
25
35
45
150
550
```

Write मेसेज के द्वारा पूरा array की bytes को फाइल में लिखा जा सकता है।

```
 /**
*Write Array Bytes Application
*/

import java.io.*;
class WriteArrayBytes{
public static void main (String [] args){
byte data[] = {15,25,35,45,150,550};
FileOutputStream f;

try {
f = new FileOutputStream("f1.data");
f.write (data,0,data.length);

f.close();
} catch (IOExceptopn e){
 System.out.println("Error with files:" + e.toString());
}
}
}
```

इस उदाहरण में मेसेज write bytes के array को arguments, पहले कॉम्पोनेन्ट के इंडेक्स को एवं कंपोनेंट्स की संख्या जिसको लिखना है को प्राप्त करती है। इस तरह से लिखा हुआ bytes डाटा को भी पिचले read के प्रोग्राम द्वारा पढ़ा जा सकता है। यहाँ पैर ध्यान देने वाली बात है कि write एक integer को आर्गुमेंट में expects करता है और मेथड read()integer के फॉर्म में रिटर्न करती है ना की बाइट के फॉर्म में। इसका कारण है कि नार्मल byte को read या write करने की वैल्यू -128 से 127 है जबकि bytes को फाइल से read या फाइल में write करने की रेंज 0 से 255 है।

Buffered Byte Oriented Streams:

संचार में सिन्क्रोनाइजेसन एवं ओवरहेड को काम करने के लिए बफर का उपयोग होता है। Buffered Byte Oriented Streams के लिए दो classes BufferedOutputStream एवं BufferedInputStream का उपयोग होता है।

```
/**
*Write Buffered Bytes Application
*/

import java.io.*;
class WriteBufferedBytes{
public static void main (String [] args){
byte data[] = {15,25,35,45,150,550};
FileOutputStream f;
BufferedOutputStream bf;
try {
f = new FileOutputStream("f1.data");
bf = new BufferedOutputStream (f);

for ( i=0;i<data.length;i++)
 bf.write(data[i]);
bf.close();
} catch (IOExceptopn e){
 System.out.println("Error with files:" + e.toString());
}
}
}
```

यहाँ पर Output Stream को arguments की तरह उपयोग करके Buffered Output Stream को बनाया गया और इस तरह हमने अपनी दिलचस्पी आउटपुट को buffering करने में दिखायी।

```
 /**
*Read Buffered Bytes Application
*/

import java.io.*;
class ReadBufferedBytes{
public static void main (String [] args){
FileInputStream f;
BufferedInputStream bf;
try {
f = new FileInputStream("f1.data");
```

```
bf = new BufferedInputStream (f);

int data;
while((data = f.read())!=-1)
System.out.println(data);
bf.close();
} catch (IOExceptopn e){
 System.out.println("Error with files:" + e.toString());
}
}
}
```

इसका आउटपुट निम्न होगा

```
15
25
35
45
150
550
```

डाटा बफर बाइट ओरिएंटेड स्ट्रीम्स Data Buffered Byte Oriented Streams): डाटा बफर बाइट ओरिएंटेड स्ट्रीम्स का उपयोग डाटा के छोटे छोटे टुकड़ों में जो primitive टाइप से मेल करती हैं। डाटा को read और write करने के लिए निम्न मेसेज का उपयोग कर सकते हैं :

Read message	Write message
readBoolean()	writeBoolean(boolean)
readByte()	writeByte(byte)
readShort()	writeShort(short)
readInt()	writeInt(int)
readLong()	writeLong(long)
readFloat()	writeFloat(float)
readDouble()	writeDouble(double)

अगले उदाहरण में डाटा को डाटा बफर बाइट ओरिएंटेड स्ट्रीम्स में स्टोर करने के लिए एक integer को जो की integer के array से मेल खाता है। उसके बाद double array और अंत में बूलियन वैल्यूज :

```
  /**
*Write Data class  Application
*/

import java.io.*;
```

```
class WriteData {
public static void main (String [] args){
byte data[] = {5.6,22.32,-5.1,2.44};
FileOutputStream f;
BufferedOutputStream bf;
DataOutputStream ds;

try {
f = new FileOutputStream(“f1.data”);
bf = new BufferedOutputStream (f);
ds = newDataOutputStream(bf);

ds.writeInt(data.length);

for ( i=0;i<data.length;i++)
 ds.writeDouble(data[i]);
ds.writeBoolean(true);
ds.close();
} catch (IOExceptopn e){
 System.out.println(“Error with files:” + e.toString());
}
}
}
```

उदाहरण में डाटा बफर बाइट ओरिएंटेड स्ट्रीम्स को तीन पदों में पूर्ण किया गया। सबसे पहले आउटपुट स्ट्रीम को बनाया गया उसके बाद इसे buffered किया गया और अंत में डाटा stream को बनाया गया। अगले उदाहरण में डाटा read का उपयोग बताया गया है :

```
 /**
*Read Data class  Application
*/

import java.io.*;
class ReadData{
public static void main (String [] args){
FileInputStream f;
BufferedInputStream bf;
DataInputStream ds;

try {
f = new FileInputStream(“f1.data”);
bf = new BufferedInputStream (f);
ds = new DataInputStream(bf);
```

```
int length = ds.readInt();
for( int i=0; i<length,i++)

System.out.println(ds.readDouble());
System.out.println(ds.readBoolean());

ds.close();
} catch (IOExceptopn e){
 System.out.println("Error with files:" + e.toString());
}
}
}
```

इस उदाहरण का आउटपुट

```
5.6
22.32
-5.1
2.44
```

कैरेक्टर ओरिएंटेड स्ट्रीम्स (Character Oriented Streams):

कैरेक्टर ओरिएंटेड स्ट्रीम्स का उपयोग characters को read और write करने के लिए होता है। आउटपुट टेक्स्ट स्ट्रीम्स को बनाने के लिए यह आवश्यक है कि एक इंस्टांस FileWriter और उसके बाद BufferedWriter का इंस्टांस बनाना है। इस प्रकार की स्ट्रीम्स को बनाने के लिए निम्न तीन मेथोड्स का उपयोग के सकते हैं।

Message
Write(String,int,int)
Write(char[], int,int)
newline()

इसमें सबसे मेसेज का पहला आर्गुमेंट के द्वारा प्राप्त स्ट्रिंग से Character लेकर दूसरे आर्गुमेंट से प्राप्त integer की वैल्यू उस Character का पद (पोजीशन) एवं तीसरे integer आर्गुमेंट से कितने Character हैं उसकी जानकारी मिलती है। दूसरा मेसेज भी पहले की तरह काम करता है दोनों में अन्तेर है कि दूसरे मेसेज में Character array के फॉर्म में होते हैं। तीसरा मेसेज आउटपुट stream में नयी लाइन (New Line) बनाते हैं। ध्यान देने वाली बात यह है कि ये मेसेज ऑपरेटिंग सिस्टम से भी

स्वतंत्र रूप से काम करते हैं। निम्न उदाहरण में कैरेक्टर ओरिएंटेड स्ट्रीम्स का उपयोग दर्शाया गया है।

```
 /**
* Write Text class Application
*/

import java.io.*;
class WriteText{
public static void main (String [] args){
FileWriter f;
BufferedWriter bf;

try {
f = new FileWriter("f1.text");
bf = new BufferedWriter(f);

String s = "Namaskar Ji"
bf.Writer(s,0,s.length());
bf.newLine();
bf.write ("Java is very Useful !#@",14,3);
bf.newLine();

bf.close();
} catch (IOExceptopn e){
 System.out.println("Error with files:" + e.toString());
}
}
}
```

इनपुट टेक्स्ट स्ट्रीम्स को बनाने के लिए यह आवश्यक है कि एक इंस्टांस FileReader और उसके बाद Buffered Reader का इंस्टांस बनाना है। इस प्रकार की स्ट्रीम्स को बनाने के लिए निम्न एक मेसेज readLine का उपयोग पूरी line को टेक्स्ट से पढने के लिए उपयोग कर सकते हैं। यह एक स्ट्रिंग, जो पुरे लाइन को स्टोर किये हुए है, के इंस्टांस को रिटर्न करता है। और फाइल के अंत में null को रिटर्न करता है। उदाहरण के लिए

```
 /**
*Read Text class  Application
*/

import java.io.*;
```

```
class ReadText{
public static void main (String [] args){
FileReader f;
BufferedReader bf;

try {
f = new FileReader(“f1.text”);
bf = new BufferedReader(f);

String s;

While ((s =bf.readLine()) != null)
System.out.println(ds.readDouble());
System.out.println(s);

ds.close();
} catch (IOExceptopn e){
 System.out.println(“Error with files:” + e.toString());
}
}
}
```

इसका आउटपुट निम्न

```
Namaskar Ji
!#@
```

स्टैण्डर्ड इनपुट (Standard Input): किसी समय में स्टैण्डर्ड इनपुट को पढने की आवश्यकता होती है और ऐसे समय में स्टैण्डर्ड इनपुट का उपयोग होता है। स्टैण्डर्ड इनपुट का रिफरेन्स जावा में एक वेरिएबल System.in के रूप में है। इसके द्वारा read करने के लिए InputStreamReader को BufferedReader को डिफाइन करना होता है।

```
 /**
*Standard Input class  Application
*/

import java.io.*;
class StandardInput{
public static void main (String [] args){
InputStreamReader isr;
BufferedReader br;

try {
```

```
isr = new InputStreamReader (System.in);
br = new BufferedReader(isr);

String line;

While ((line = br.readLine()).lenght()!0)
System.out.println(line);

} catch (IOExceptopn e){
 System.out.println("Error in standard Input);
}
}
}
```

इसमें readLine() मेथड एक पूरी लाइन को स्टैण्डर्ड इनपुट से एक स्ट्रिंग के रूप में रिटर्न करती है एक और मेथड length() को कॉल किया गया जिससे यह चेक किया जा सके कि स्टैण्डर्ड इनपुट close हो गया।

कभी कभी ऐसे उदाहरण भी मिले हैं जिसमें मेथड के द्वारा अभिव्यक्त किया जाता है कुछ एक्सेप्शन को catch के द्वारा ट्रैप न किया जा सके । अतः इसके समाधान के रूप में नया कीवर्ड throws का उपयोग करते हैं । इस केस में try-catch ब्लाक को डिफाइन करने की जरूरत नहीं होती है।

```
/**
*Standard Input class  Application-2
*/

import java.io.*;
class StandardInputWithThrows{
public static void main (String [] args) throws IOException
{
InputStreamReader isr;
BufferedReader br;

isr = new InputStreamReader (System.in);
br = new BufferedReader(isr);

String line;

While ((line = br.readLine()).lenght()!=0)
System.out.println(line);
}
}
```

थ्रेड्स (Threads):

जावा में अलग अलग tasks को साथ साथ रन करने के लिए थ्रेड्स का उपयोग होता है। प्रत्येक थ्रेड एक task को स्वतंत्र रूप से CPU द्वारा दिए गए टाइम के अनुसार रन करता है। अलग अलग थ्रेड आपस में बात भी हो सकती है और आपस में डाटा ट्रान्सफर को भी सिन्क्रोनाइज़ कर सकते हैं।

थ्रेड को डिफाइन करने के लिए थ्रेड class की एक subclass को बनाना होता है। थ्रेड class में एक abstract मेथड जिसका नाम run है इसे भी subclass में डिफाइन करना होता है। इसी मेथड में एक स्वतंत्र थ्रेड को चलने का कोड उपलब्ध है।

निम्न उदाहरण में CharTread जो कि थ्रेड की subclass है, को डिफाइन किया गया है :

```
/**
*Char Thread class application
*/

class CharThread extends Thread {
Char c;
CharThread (Char ac){
c = ac;
}
public void run(){
whilr(true) {
System.out.println( c);
try {
sleep (100);                    // यह टाइम डिले के लिए है
} catch (InterruptedException e){
      System.out.println( “Interrupted”);
}
}
}
}
```

एक class एक ऐसे character डाटा मेम्बर को डिफाइन करता है और जब class का इंस्टांस constructor की सहायता से बनता है तब ये इनिशियलाइज़ हो जाता है मेथड run में एक infinite लूप character को प्रिंट करता है। उसके बाद थ्रेड स्लीप मोड में 100 मिलिसेकंड के लिए चला जाता है। एक और बात ध्यान देने की है कि एक्सेप्शन स्लीप मोड में भी आ सकता है इसलिए कोड में try-catch मोड को डिफाइन किया गया है। अगले उदाहरण में दो इंस्टांस बनाये गए हैं और उन्हें अलग अलग character से इनिशियलाइज़ किया गया है। जब भी दोनों स्टार्ट होती हैं तब run मेथड दोनों इंस्टांस को साथ साथ चलाती है। उदाहरण के लिए :

```
/**
*Test Threads class Application
*/
class TestThread{
public static void main (String[]args){
CharThread t1 = new CharThread(‘a’);
CharThread t2 = new CharThread(‘b’);

t1.start();
```

```
t2.start();

}
}
```

इसमें CharThread के दो इंस्टांस एक constructor का उपयोग करके बनाये गए हैं। दोनों इंस्टांस एक साथ स्टार्ट हो रहे हैं।

इसका आउटपुट है

```
a
b
 a
b
a
b
a
b
a
b
.....
```

दोनों थ्रेड्स CPU के पास एक साथ जाते है और CPU से मिले टाइम के अनुसार वैल्यू देते हैं।

एक उदाहरण जिसमें एक से अधिक थ्रेड्स एक साथ काम कर रहे हैं। इसके लिए एक अच्छा उदाहरण निर्माता और उपभोक्ता का है इसमें विचार यह है कि यह एक उदाहरण है जिसमें दो प्रोसेस है और दोनों ही प्रोसेस एक ही बफर से interact इंटरैक्ट करती हैं। निर्माता द्वारा बनाये गए आइटम्स बफर में स्टोर होते है और उसी बफर से आइटम्स को उपभोक्ता द्वारा उपयोग में लाता है। इसमें सिन्क्रोनाइज़ करने की समस्या हो सकती है क्योंकि दोनों प्रोसेस एक ही बफर इंटरैक्ट करती हैं और इसलिए बफर एक क्रिटिकल रिसोर्सेज। एक और समस्या है कि निर्माता भरे हुए बफर में आइटम नहीं रख सकता और उपभोक्ता खली बफर में से आइटम नहीं ले सकता। इसलिए उदाहरण उपभोक्ता एवं निर्माता का थ्रेड्स को समझने के लिए लिया गया है।

```
/**
*Producer Consumer class
*/
```

```
class producerConsumer {

public static void main (String[] args){

Buffer buffer = new Buffer (30)   // 30 बफर का साइज़ है

Producer prod = new Producer(buffer);
Consumer cons = new Consumer(buffer);

Prod.start();
Cons.start();
}
}
```

इसमें एक बफर जो कि 30 आइटम्स को स्टोर कर सकता है, को बनाया गया है और इसके बाद एक इंस्टांस निर्माता (Producer) का और एक इंस्टांस उपभोक्ता(Consumer) का बनाया । दोनों ही इंस्टांस में एक argument buffer जिससे दो इंस्टांस एक दूसरे से कम्यूनिकेट कर सकें । और अंत में दोनों इंस्टांस को स्टार्ट किया । अब हम उपभोक्ता के लिए class बनायेंगे

```
/**
*Producer class Application
*/


class Producer extends Thread {

Buffer buffer;

public Producer (Buffer b){
buffer = b;
}

public void run(){
double value =0.0;

while (true){
buffer.insert(value);
value += 0.1;
}
}
}
```

उपर्युक्त producer class, थ्रेड्स class की subclass है और इसलिए इसमें run मेथड है जो एक स्वतंत्र थ्रेड चला सकती है इस class में एक डाटा मेम्बर जो एक

रिफरेन्स जो बफर constructor के arguments से पास हुआ है, को समाविष्ट करेगा।

यहाँ पर मेथड run एक double वैल्यू को बफर में इन्सर्ट करती है जिससे इनफिनिट लूप चलता है। इसलिए producer आइटम्स (double values) को produce करके कॉमन बफर में इन्सर्ट करता है।

```
/**
*Consumer class Application
*/
class Consumer extends Thread {

Buffer buffer;

public Consumer (Buffer b){
buffer = b;
}

public void run(){
while (true){
 System.out.println(buffer.delete());
}
}
}
```

class Consumer भी class थ्रेड की subclass है जो एक स्वतंत्र थ्रेड चला सकती है। इस class में एक डाटा मेम्बर जो एक रिफरेन्स जो बफर constructor के arguments से पास हुआ है, को समाविष्ट करेगा। यहाँ पर run मेथड डाटा बफर से निकल देती है। और उसे स्टैण्डर्ड आउटपुट में प्रिंट करती है।

अब इस उदाहरण को देखते हैं

इसमें बफर को सर्कुलर बफर में डिफाइन करेंगे और ये बफर एक array और दो पॉइंटर्स के साथ इम्प्लीमेंट किया गया है। एक पॉइंटर हेड पोजीशन के लिए और दूसरा टेल पोजीशन के लिए। टेल पोजीशन के पॉइंटर्स द्वारा डाटा को इन्सर्ट किया जा सकता है और हेड पोजीशन से डाटा को read किया जा सकता है। एक और डाटा मेम्बर है जो कितने आइटम्स बचे हैं उसकी सख्या बताएगा।

```
/**
*Buffer clasa Application
```

```
*/

class Buffer {
double buffer[];
int head =0;
int tail =0;
int size =0;

int numElements =0;

public Buffer (int s){
buffer = new double[s];
size =s;
numElement =0
}

Public void insert (double element){
Buffer[tail] = element;
Tail = (tail+1)% size;
numElement++;
}

public double delete(){
double value = buffer[head];
head = (head-1)%size;
numElement --;
return value;
}
}
```

यहाँ पर ऐसा प्रतीत होता है कि ये मेथड अच्छे से काम करेगी पर यह सही नहीं है इसको सही तरीके से काम करने में दो समस्याएँ हैं

- इसमें दो मेथोड्स हैं `insert()`और `delete()` दोनों एक ही स्ट्रक्चर में हैं। दोनों मेथोड्स एक साथ `execute` नहीं हो सकती हैं यदि दोनों का उपयोग करना है तब कोई एक मेथड को पहली मेथड जो run कर रही है उसके समाप्त होने का इंतजार करना होगा या इसका उल्टा भी हो सकता है।
- `insert()`मेथड ये भी चेक नहीं करती कि बफर में कोई स्लॉट खली है कि नहीं और इसी तरह `delete()` मेथड भी यह चेक नहीं करती कि बफर में एक भी डाटा है कि नहीं ।

अगले भाग में इन्ही दो समस्याओं का समाधान मिलेगा

सिन्क्रोनाइज़ मेथड (Syncronized method):

जावा में सिन्क्रोनाइज़ मेथड उपलब्ध हैं। इसमें दोनों मेथोड्स एक साथ नहीं चलती है। इसमें इंस्टांस में एक लॉक है जो सिन्क्रोनाइज़ करने के काम आता है। पहली समस्या का समाधान निम्न उदाहरम से मिलता है :

```
public synchronized void insert (double element){

Buffer[tail] = element;

Tail = (tail+1)% size;
numElement++;
}


public synchronized double delete(){
double value = buffer[head];
head = (head-1)%size;
numElement --;
return value;
}
```

वेट और नोटिफाई (Wait and notify):

subclass के द्वारा मेसेज के रूप में Wait और notify भेजा जाता है। ये मेसेज केवल सिन्क्रोनाइज़ मेथड (Syncronized method)के द्वारा ही भेजे जा सकते हैं। मेसेज Wait थ्रेड को स्लीप मोड में डाल देता है और लॉक को जारी कर देता है। हमारे उपर्युक्त उदाहरण में जब थ्रेड वैल्यू को बफर में इन्सर्ट करना चाहता है और बफर खाली नहीं है तब थ्रेड स्लीप मोड में चला जाता है। इसी तरह दूसरा थ्रेड जब बफर में से वैल्यू को निकालना चाहता है और बफर खाली है तब ये स्लीप मोड में चला जाता है।

एक थ्रेड जिसने अभी अभी डाटा को बफर में इन्सर्ट किया है उसको इसे notify करना होता है जिससे दूसरा थ्रेड जाग जाये। इसी तरह जब थ्रेड बफर में से डाटा निकालता है तब उसे भी notify करना होता है जिससे दूसरा थ्रेड जाग जाये। इस दोनों मेथोड्स को निम्न उदाहरण से समझाजा सकता है।

```
  public synchronized void insert (double element){
if(numElement == size){
```

```
try {

wait();}

catch (InterruptedException e){
     System.out.println( “Interrupted”);
}
}
Buffer[tail] = element;

Tail = (tail+1)% size;
numElement++;
notify();
}


public synchronized double delete(){

if(numElement == 0){

try {

wait();}

catch (InterruptedException e){
     System.out.println( “Interrupted”);
}
}

double value = buffer[head];
head = (head-1)%size;
numElement --;
notify();
return value;
}
```

यहाँ पर ध्यान देने वाली बात है कि Wait के साथ catch एक्सेप्शन का उपयोग होना ही है।

जार फाइल्स (JAR Files):

जब हमने निर्माता एवं उपभोक्ता वाला उदाहरण देखा था तब उसमें निम्न चार फाइल बनायीं गयी थीं।

```
Buffer.class
Producer.class
```

```
Consumer.class
ProducerConsumer.class
```

इस तरह के एप्लीकेशन में जहां एक से ज्यादा फाइल्स बनती है और सभी का उपयोग एप्लीकेशन के execution में होता है। सभी फाइल्स को एक पैक में compressed रूप में रखने के लिए जावा एक मैकेनिज्म उपलब्ध करता है जिसे JAR (Java ARchive) Files कहते हैं। JAR File को jar कमांड से बनाया जाता है। JAR Files बनाते समय एक और फाइल बनायीं जाती है जिसे manifest फाइल कहा जाता है ये फाइल उन सभी फाइल्स की इनफार्मेशन रखती है जिनको JAR Files में रखा जा रहा है।

jar कमांड एक डिफ़ॉल्ट manifest फाइल META-INF डायरेक्टरी में MANIFEST.MF के नाम से बनाता है जो current डायरेक्टरी के बाद होती है।

manifest फाइल में फाइल्स के अलावा कुछ लाइन्स को भी arguments के द्वारा जोड़ा जा सकता है। जैसे उस फाइल का नाम जिसमें लाइन्स लिखी गयी हैं। इनफार्मेशन को जोड़े (Key,value)के रूप में ही दिया जा सकता है।

निर्माता उपभोक्ता के उदाहरण में केवल एक जरूरी जोड़ा को जिसमें उस class का नाम जिसमें main function है को एक फाइल mylines.text में दिया जा रहा है।

```
# cat mylines.text
Main-class: ProducerConsumer
```

JAR File को निम्न प्रकार से बनाया जा सकता है।

```
# jar cmf mylines.text ProducerConsumer.jar
ProducerConsumer.class      Buffer.class      Producer.class
Consumer.class
```

इस cmf कमांड में c क्रिएशन ऑफ़ jar फाइल्स के लिए , m manifest फाइल में lines की text फाइल को जोड़ने के लिये एवं f जार फाइल का नाम आर्गुमेंट के द्वारा प्रदान करने के लिए उपयोग हो रहा है। ProducerConsumer.jar के नाम से जार फाइल बनानी है और अंत में उन सभी फाइल्स के नाम जो इस jar में रहेंगी।

निम्न कमांड से हम ProducerConsumer.jar में क्या क्या है उसे देख सकते हैं

```
# jar tf ProducerConsumer.jar
META-INF/
META-INF/MENIFEST.MF
```

```
ProducerConsumer.class
Buffer.class
Producer.class
Consumer.class
```

JAR फाइल में उपलब्ध एप्लीकेशन को निम्न प्रकार से execute कर सकते हैं

```
# java –jar ProducerConsumer.jar
```

जावा नेटिव इंटरफ़ेस (Java Native Interface JNI):

जावा नेटिव इंटरफ़ेस ऐसे फंक्शन को कॉल करने के लिए उपयोग होता है जो जावा के आलावा किसी और लैंग्वेज में लिखा हो। यह उन एप्लीकेशन के लिए बहुत ही उपयोगी है जिसमें जावा में उपलब्ध संसाधनों से पूरा समाधान नहीं निकलता है। इसके निम्न कारण हो सकते है :

- कोई ऐसा बहुत बड़ा टेस्टेड कोड हो जो किसी दूसरी लैंग्वेज में डेवेलप हुआ हो। और इसे दोबारा लिखने का कोई मतलब नहीं है।
- एप्लीकेशन को ऐसे सिस्टम की विशेषताओं या डिवाइस की आवश्यकता हो जो जावा में आज उपलब्ध नहीं हैं।
- Execution स्पीड अनिवार्य हो।

हम यह मान भी लें कि JNI उपर्युक्त समस्याओं का समाधान दे रहा हो फिर भी इसमें कुछ नुकसान भी हैं। पहला नुकसान है कि पोर्टेबिलिटी (Portability) समाप्त हो जाती है क्योंकि दूसरी लैंग्वेज में लिखा कोड किसी एक प्लेटफार्म पर चलेगा अतः हर

प्लेटफार्म पर नहीं चलेगा और पोर्टेबिलिटी (Portability) कम हो जाएगी। दूसरी समस्या है कि सुरक्षा (security) कम हो जाती है क्योंकि दूसरी लैंग्वेज में उस लेवल की सुरक्षा (security) उपलब्ध नहीं है जो जावा में है।

जावा में ऑब्जेक्ट का उपयोग जितना सरल है जावा नेटिव इंटरफ़ेस (Java Native Interface JNI)का उपयोग उतना ही कठिन है। क्योंकि जावा ऑब्जेक्ट का दूसरी लैंग्वेज के कोड के साथ काम करना बहुत ही उलझा हुआ काम है। JNI का उपयोग C लैंग्वेज में बनाये गए कोड के साथ उपयोग अगले भाग में समझाया गया है।

नेटिव मेथड की परिभाषा(The definition of native methods):

नेटिव मेथड के उपयोग के लिए निम्न पदों का पालन करना होगा

1. नेटिव कीवर्ड का उपयोग करके एक जावा class में फंक्शन को डिफाइन किया जाये।
2. जावा एप्लीकेशन को कम्पाइल करें
3. जावा में उपलब्ध यूटिलिटी javah का उपयोग करके C का header बनाया जाये।
4. JNI की गाइडलाइन्स का उपयोग करके C फंक्शन को डिफाइन करें
5. अंत में साझा लाइब्रेरी में C फंक्शन (और header फाइल) को कम्पाइल करें

निम्न उदहारण में class Car में एक सरल मेथड जो कि बिना arguments के static होगी एवं कोई भी वैल्यू रिटर्न नहीं करेगी, बनायेंगे। मेथड printDescription()को नेटिव मेथड (step1) डिफाइन निम्नानुसार किया जायेगा

```
class Car {
……..
public native static void printDescription();
}
```

यह मेथड पूर्व मेथोड्स में दो अंतर हैं पहला इस नेटिव मेथड की कोई बॉडी नहीं है और दूसरा native कीवर्ड का उपयोग किया गया है। यह native कीवर्ड कम्पाइलर को बता है कि यह मेथड किसी दूसरी लैंग्वेज में जावा के बाहर डिफाइन की गयी है।

अब class Car को सामान्य तरीके से कम्पाइल किया जाये(step2) :

```
# javac Car.java
```

यूटिलिटी javah का उपयोग करके एक header फाइल निम्नानुसार बनायेंगे(step3)

```
javah Car
```

इस कमांड के द्वारा एक header फाइल class के नाम से ही .h एक्सटेंशन के साथ बनेगी

Car.h

उपर्युक्त फाइल में निम्न कंटेंट होंगे:

```
# include <jni.h>
/* class Car ke liye header file */
#ifndef _Included_Car
# define _Included_Car
#ifdef _cplusplus
Extern “C”{
#endif
/*Inaccessible static:Owner*/
/*
*Class Car
*Method: printDescription
*Signatur: ()V
*/
JNIEXPORT void JNICALL Java_Car_ printDescription
(JNIEnv *,jclass);
```

इस फाइल का संपादन नहीं हो सकता है क्योंकि यह javah के द्वारा बनी है। जब भी इस तरह के प्रोग्राम चलेंगे यह मेथड पुनः बनायीं जाएगी अंत की दो लाइन में C का फंक्शन डिफाइन किया गया है जिसका नाम Java_Car_ printDescription है

```
JNIEXPORT void JNICALL Java_Car_ printDescription
(JNIEnv *,jclass);
```

इस फंक्शन का नाम Java_Car_ printDescriptionहै। यह नाम Java सेशुरू होगा उसके बाद अंडरस्कोर से अलग करके class का नाम होगा और अंत में मेथड का नाम होगा। इसमें रिटर्न वैल्यू का टाइप void है अतः ये कोई भी वैल्यू रिटर्न नहीं करेगी। इसको ध्यान से देखने पर पता चलता है कि ओरिजिनल जावा मेथड में कोई भी

आर्गुमेंट नहीं होने पर भी यह दो आर्गुमेंट को प्राप्त करती है। पहला आर्गुमेंट पॉइंटर्स के टेबल को पॉइंट करता है और यह C से जावा की फंक्शनालिटी को access करता है। दूसरा आर्गुमेंट उस क्लास को परिभाषित करता है जिसने मेसेज को भेजा था। साझा लाइब्रेरी के इम्प्लीमेंटेशन को Macros JNIEXPORT और JNICALL की सहायता से परिभाषित करते हैं।

अब javah के द्वारा बनाये गए प्रोटोटाइप का उपयोग करके एक फाइल Car.c में C फंक्शन को डिफाइन निम्नानुसार करेंगे। (step4)

```
# include <Car.h>
# include <studio.h>
JNIEXPORT void JNICALL Java_Car_ printDescription
(JNIEnv *env,jclass c1){
 Printf("It is a Car\n");
}
```

यहाँ पर ध्यान देने वाली बात है कि फाइल एक स्टैण्डर्ड C फाइल है और यह C की सभी फंक्शनालिटी का उपयोग हो सकता है। इसमें फंक्शन एक स्ट्रिंग को स्टैण्डर्ड आउटपुट पर प्रिंट करता है।

इस फंक्शन को साझा लाइब्रेरी की तरह कम्पाइल करना है। (step5)

यह फाइल C के कम्पाइलर और JNI फाइल सर्च करने के पाथ पर आश्रित है। निम्न कमांड का उपयोग किया जाता है :

```
# gcc –shared –o libCar.so Car.c
```

इसमें फ्लैग –shared यह बताता है कि Car.c को साझा लाइब्रेरी में कम्पाइल करा है –o आप्शन देता है कि इसका नाम libCar.so होगा।

यहाँ पर यह भी आवश्यक है Class Car को निर्देशित किया जाये कि जो मेथड native जिस Class में डिफाइन है उसे लोड किया जाये। सामान्यतः यह Class Car में static ब्लाक को इनिशियलाइज़ निम्न तरह से किया जाता है

```
 Class Car{
……
static{
System.loadLibrary("Car");
}
}
```

मेथड loadLibrary(), Class Car को निर्देशित करती है कि साझा लाइब्रेरी को लोड करे। निम्न उदाहरण से native मेथड को चेक करेंगे।

```
class TestCar {
public static void main(String[] args){
Car c1 = new Car("WagonR","Amit Kumar",18 );
c1. printDescription();
}
}
```

इस का आउटपुट है :

```
It is a Car
```

निउमेरिक पैरामीटर्स और रिटर्न वैल्यू (Numeric parameters and return values):

यहाँ पर हमको यह समझाना होगा कि एक और पॉइंट है जिससे native मेथड का इम्प्लीमेंटेसन और उलझा हुआ बन जाता है और वह है कि स्टैण्डर्ड टाइप जावा और C में अलग अलग हैं जावा में उपलब्ध सभी स्टैण्डर्ड टाइप एक विशेष साइज़ का होता है पर C में ऐसा नहीं है उदाहरण के लिए C में int का साइज़ नेचुरल है और यह प्लेटफार्म पर डिपेंडेंट है। यही कारण है कि JNI में आल टाइप्स को डिफाइन किया गया है इससे प्रत्येक टाइप को जावा और C के बीच डाटा पास करने के लिए उपयोग किया जा सके। निम्न सारिणी में विस्तृत जानकारी है।

Java	C	Size
boolean	Jboolean	1
byte	jbyte	1
char	jchar	2
short	jshort	2
int	jint	4
long	jlong	8
float	jfloat	4
double	jdouble	8

इसमें jboolean टाइप की वैल्यू JNI_TRUE और JNI_FALSE हैं जो क्रमशः 1 और 0 हैं।

निम्न उदाहरण में एक और native मेथड computeDistance()डिफाइन किया गया जो किसी Car के द्वारा नियत पेट्रोल में कितनी दूरी (Distance) तय की जाएगी की गणना करे। इसे C में डिफाइन किया गया है।

```
JNIEXPORT jfloat JNICALL Java_Car_ computeDistance)
(JNIEnv *env,jclass c1,jint petrolInLiter, jfloat average){
jfloat distance;

//compute the distance
distance = (petrolInLiter* average);
return distance }
}
```

इस मेथड में दो पैरामीटर्स एक jint और jfloat को डिफाइन किया गया है और रिटर्न टाइप भी jfloat है। इस मेथड की बॉडी द्वारा distance की गणनाकी गयी है और यहाँ पर ध्यान देने वाली बात है कि jint और jfloat दोनों क्रमशः int और float के साथ संयुक्त किये गए हैं। इस मेथड का उपयोग अगले एप्लीकेशन में दिखाया गया है

```
Class TestCar22{
 public static void main(String[] args){
float distance = Car. computeDistance(34.5,18.4);
System.out.println("Distance="+distance+"KM");

}
}
```

इसका आउटपुट निम्न आएगा

```
Distance = 353.2 KM
```

स्ट्रिंग का प्रयोग (Using String):

स्ट्रिंग को जावा मेथड और C के फंक्शन में JNI के द्वारा साझा करना और कठिन है क्योंकि स्ट्रिंग का उपयोग दोनों भाषाओँ में बहुत अलग है। C में स्ट्रिंग 1 बाइट character के अंत में null वैल्यू से समाप्त होती है और जावा में स्ट्रिंग दो बाइट character का होता है और अंत में UNICODE character होता है।

निम्न उदाहरण में Class Car में एक मेथड printModel() को डिफाइन किया गया है जो एक स्ट्रिंग को एक arguments से प्राप्त करेगी और इस स्ट्रिंग को स्टैण्डर्ड आउटपुट में प्रिंट करेगी।

```
public native static void printModel(String Model);
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
 JNIEXPORT void JNICALL Java_Car_printModel
(JNIEnv *env, jclass c1, jstring model){

char*str =(*env)-> GetStringUTFChars(env,model,null);
printf(str);

(*env)-> ReleaseStringUTFChars(env,model,str);
Printf("\n");
}
```

इस आर्गुमेंट model को jstring डिफाइन किया गया है जो की जावा का स्ट्रिंग टाइप है जिसे JNI में native मेथड में उपयोग के लिए डिफाइन किया गया है। यह फंक्शन जावा स्ट्रिंग को GetStringUTFChars() फंक्शन का उपयोग करके नार्मल C स्ट्रिंग में बदलता है और इसे str में स्टोर करता है। इसके बाद printf फंक्शन को कॉल करके str को प्रिंट करता है और अंत में जो मेमोरी आवंटन को ReleaseStringUTFChars() फंक्शन का उपयोग करके मुक्त करता है।
इसमें पॉइंटर env,पॉइंटर से फंक्शन की टेबल को पॉइंट करता है इसका उपयोग एक हुक की तरह से जावा फंक्शन को access करने में होता है। निम्न उदाहरण में उपर्युक्त मेथड का प्रयोग देख सकते हैं।

```
 class TestCar33 {
public static void main(String[] args){
Car c1 = new Car("WagonR","Amit Kumar",18 );
Car. printModel(c1.model);
}
}
```

इसका आउटपुट निम्न होगा
WagonR

निम्न उदाहरण में एक native मेथड getDescription() को डिफाइन किया गया है जो कोई भी आर्गुमेंट प्राप्त नहीं करती है और स्ट्रिंग रिटर्न करती है:

```
public native static void getDescription();
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
 JNIEXPORT jstring JNICALL Java_Car_printModel
(JNIEnv *env, jclass c1){
```

```
jstring jstr;
char desc[] = “It is WagonR Car”;
jstr = =(*env)-> NewStrinUTFChars(env,desc);
return jstr;
}
```

इसमें फंक्शन NewStrinUTFChars(), C के स्ट्रिंग desc को जावा स्ट्रिंग में बदलता है जो अंत में वैल्यू को रिटर्न करता है। निम्न उदाहरण में इसका उपयोग देख सकते हैं :

```
 class TestCar44 {
public static void main(String[] args){
Car c1 = new Car(“WagonR”,”Amit Kumar”,18 );
String str = c1.getDescription();
System.out.println(str);
}
}
```

इसका आउटपुट निम्न होगा

```
It is WagonR Car
```

नॉन स्टेटिक मेथड एवं नॉन स्टेटिक फ़ील्ड्स स्टेटिक का उपयोग (Using non static method and non static Fields):

अभी तक जितनी भी native मेथड का उपयोग हुआ है सभी स्टेटिक हैं। अब हम अगले उदाहरण में नॉन स्टेटिक मेथड जो फील्ड को ऑब्जेक्ट से access करेगी। यहाँ पर मेथड द्वारा फील्ड में increment करेंगे इसे प्योर जावा में निम्नानुसार करते हैं

```
public void AverageIncrement(int ave){
 average += ave;
}
```

इसका उपयोग निम्न प्रकार से किया जा सकता है

```
class TestCar55 {
public static void main(String[] args){
Car c1 = new Car(“WagonR”,”Amit Kumar”,18 );
C1. AverageIncrement(5);
System.out.println(c1.average);
}
}
```

इसका आउटपुट निम्न होगा

```
23
```

यही मेथड को हम native मेथड से निम्नानुसार लिख सकते हैं।

```
public native static void AverageIncrement(int ave);
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT jstring JNICALL Java_Car_ AverageIncrement
(JNIEnv *env, jobject obj_this, jint ave){
jclass class_Car =(*env)-> GetObjectClass(env, obj_this);
jfieldID    id_average    =    (*env    ->    GetFieldID(env,
class_Car,"Average","I");
jint    average    =    (*env    ->    GetIntField(env,obj_this
id_average);
average += ave;
(*env) -> SetIntField(env, object_this,id_average, average);
}
```

ये फंक्शन तीन पैरामीटर्स रखता है पहला वाला सामान्य environment और तीसरा एक integer है जो average से सम्बंधित है। दूसरा पैरामीटर एक नॉन static मेथड है उस ऑब्जेक्ट को रेफ़र करती है जो मेसेज का रिसेप्टर है। और दूसरे शब्दों में यह मेथड this की रिसेप्टर है। इसका मतलब हुआ कि static मेथड class का रिफरेन्स है और नॉन static मेथड this का रिफरेन्स है।
ऑब्जेक्ट की किसी भी फील्ड को access करने के लिए यह आवश्यक है कि पहले class का रिफरेन्स प्राप्त करें और उसके बाद फील्ड की पहचान करें। उपर्युक्त उदाहरण में class का रिफरेन्स निम्न प्रकार से लिया गया है

```
jclass class_Car =(*env)-> GetObjectClass(env, obj_this);
```

इसमें फंक्शन GetObjectClass() द्वारा उस ऑब्जेक्ट के रिफरेन्स class के लिए रिटर्न किया गया जो आर्गुमेंट से पास हुआ है। जो ऑब्जेक्ट आर्गुमेंट के द्वारा पास हुआ है वह this का रिफरेन्स है। इस स्टेटमेंट के execution के बाद वेरिएबल class_Car जो कि jclass टाइप की है वह class Car का रिफरेन्स रखेगी।
इसी तरह फिल्ड को निम्नानुसार identify किया जा सकता है।

```
jfieldID    id_average    =    (*env    ->    GetFieldID(env,
class_Car,"Average","I");
```

इसमें फंक्शन GetFieldID() ऑब्जेक्ट फिल्ड जो आर्गुमेंट से पास हुई है, के फिल्ड आइडेंटिफायर को रिटर्न करेगा। इसमें पहला आर्गुमेंट सामान्य environment है , दूसरा ऑब्जेक्ट की class बताता है तीसरा नाम एक स्ट्रिंग के रूप में बताता है और चौथा स्ट्रिंग है जो फील्ड का टाइप बताता है। यहाँ पर I का मतलब integer है।। इस स्टेटमेंट के execution के बाद वेरिएबल id_average जो कि jfieldID टाइप का है और Car.average ka फील्ड आइडेंटिफायर है।
इसमें फील्ड वैल्यू को निम्न स्तेमेंट से प्राप्त किया जा सकता है

```
jint average = (*env -> GetIntField(env,ओब्ज_this id_average);
```

इसमें आर्गुमेंट के द्वारा डिफाइन की गयी integer फील्ड की वैल्यू को फंक्शन GetIntField()के द्वारा रिटर्न किया जाता है। इसमें पहला आर्गुमेंट सामान्य environment है , दूसरा ऑब्जेक्ट का रिफरेन्स है (इस केस में this) और तीसरा आर्गुमेंट फील्ड आइडेंटिफायर है । इसमें रिटर्न वैल्यू integer टाइप है।

एवरेज (average) के increment के बाद नयी वैल्यू आने पर उसे सेट करने के लिए निम्न स्टेटमेंट का उपयोग हो रहा है।

```
(*env) -> SetIntField (env, object_this,id_average, average);
```

इसमें SetIntField() अपने arguments की विशिष्ट वैल्यू की सहायता से integer फिल्ड को identify करती है। इसमें पहला आर्गुमेंट सामान्य environment है , दूसरा ऑब्जेक्ट का रिफरेन्स है (इस केस में this) और तीसरा आर्गुमेंट फील्ड आइडेंटिफायर है और चौथा आर्गुमेंट jint टाइप की नयी वैल्यू है।

इस उदाहरण में हमने I का उपयोग integer के लिया किया है इसे signature कहा जाता है। निम्न सारिणी में विभिन्न signature को डिफाइन किया गया है जिनका उपयोग आवश्यकता के अनुसार किया जा सकता है।

Signature	Type name
B	Byte
C	Char
D	Double

F	Float
I	Int
J	Long
L classname	A class classname
S	Short
V	Void
Z	Boolean

JNI में Get--Field और Set--Field फंक्शन डिफाइंड हैं जिसमें -- की जगह पर आवश्यकता के अनुसार Boolean, Byte, Char, Short, Int, Long, Float, Double और Object में से कोई का भी उपयोग कर सकते हैं।

अब अगले उदाहरण में एक native मेथड getDistance() एक आर्गुमेंट जो कि average को प्राप्त करती है और Distance को रिटर्न करती है।

```
public native float getDistance(float petrolInLit)
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
 JNIEXPORT jfloat JNICALL Java_Car_ getDistance
(JNIEnv *env, jobject obj_this, jfloat petrolInLit){
jclass class_Car =(*env)-> GetObjectClass(env, obj_this);
jfieldID    id_average    =    (*env    ->    GetFieldID(env,
class_Car,"Average","I");
jint    average    =    (*env    ->    GetIntField(env,obj_this
id_average);
distance = (petrolInLiter* average);
return distance ;
}
```

निम्न एप्लीकेशन में उपर्युक्त मेथड का उपयोग दिखाया गया है।

```
class TestCar66{
 public static void main(String[] args){
Car c1 = new Car("WagonR","Amit Kumar",18 );
Float distance = c1.getdistance(33.5);
System.out.println("Distance = "+ distance+"KM");
}
}
```

इसका आउटपुट निम्न होगा

```
603 KM
```

एक्सेसिंग स्टेटिक फील्ड (Accessing Static field):

JNI static फील्ड को एक्सेस और संशोधित करने के लिए फंक्शन का सेट उपलब्ध करता है । अगले उदाहरण में एक native मेथड updateOwner() को डिफाइन करेंगे जो एक स्ट्रिंग आर्गुमेंट प्राप्त करती है और स्ट्रिंग को ही रिटर्न करती है । यह मेथड static फील्ड Owner को नए पास किये हुए स्ट्रिंग आर्गुमेंट को प्राप्त करती है

```
public native String updateOwner(String newOwner);
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT jfloat JNICALL Java_Car_ updateOwner
(JNIEnv *env, jobject obj_this, jString newOwner){

jclass class_Car =(*env)-> GetObjectClass(env, obj_this);

jfieldID id_Owner = (*env -> GetStaticFieldID(env,
class_Car,"owner","LJava/lang/String;");

jString owner = (*env -> GetStaticObjectField(env,obj_this
id_owner);
char *str = (*env)->GetStringUTFChars(env,owner,null);
printf("Old owner:%s\n",str);
(*env)->ReleaseStringUTFChars(env,owner,str);
(*env)->SetStaticObjectField(env,
obj_this,id_owner,newOwner);
return newOwner ;
}
```

ये फंक्शन स्ट्रिंग आर्गुमेंट के रूप में newOwner जो कि jString टाइप का है को प्राप्त करता है और जावा स्ट्रिंग रिटर्न करता है । ऑब्जेक्ट की static फील्ड को access करने के लिए यह आवश्यक है कि पहले class का रिफरेन्स को प्राप्त करें उसके बाद फील्ड आइडेंटिफायर को प्राप्त करें ।

class का रिफरेन्स को निम्न प्रकार से प्राप्त कर सकते हैं ।

```
jclass class_Car =(*env)-> GetObjectClass(env, obj_this);
```

फील्ड आइडेंटिफायर को को निम्न प्रकार से प्राप्त करें

```
jfieldID id_Owner = (*env -> GetStaticFieldID(env, class_Car,”owner”,”LJava/lang/String;”);
```

उपर्युक्त फंक्शन GetStaticFieldID() static ऑब्जेक्ट में आर्गुमेंट से वर्णित फील्ड का आइडेंटिफायर को रिटर्न करता है।
इसमें पहला आर्गुमेंट सामान्य environment है , दूसरा ऑब्जेक्ट की class तीसरा एक स्ट्रिंग है जो नाम को दर्शाता है चौथा आर्गुमेंट फील्ड के टाइप को identify करता है इस केस में class LJava/lang/String के इंस्टांस को फील्ड समाविष्ट करती है। यहाँ पर ध्यान देने वाली बात है कि class का नाम, पैकेज और class के नाम की कड़ी(Concatenation) के रूप में उपयोग किया गया है यहाँ पर class के नाम को / द्वारा अलग किया गया है। इसमें फील्ड वैल्यू को निम्न स्टेटमेंट से प्राप्त कर सकते हैं।

```
jString owner = (*env -> GetStaticObjectField(env,obj_this id_owner);
```

इसमें फंक्शन GetStaticObjectField() के द्वारा उस ऑब्जेक्ट के रिफरेन्स को रिटर्न करते है जिसको static फील्ड के arguments से वर्णितकिया गया है।
इसमें पहला आर्गुमेंट सामान्य environment है , दूसरा ऑब्जेक्ट का रिफरेन्स (this) तीसरा फील्ड आइडेंटिफायर है और jobject टाइप की वैल्यू को रिटर्न करता है जिसे सुरक्षित रूप से jString को असाइन करते हैं।
जावा स्ट्रिंग को प्रिंट करने के लिए यह आवश्यक है कि इसे नार्मल C स्ट्रिंग में बदला जाये और यह कार्य निम्न प्रकार से किया जाता है

```
char *str = (*env)->GetStringUTFChars(env,owner,null);
printf(“Old owner:%s\n”,str);
(*env)->ReleaseStringUTFChars(env,owner,str);
```

और अंत में नयी वैल्यू को सेट करने के लिए फंक्शन SetStaticObjectField() का उपयोग निम्नानुसार कर सकते हैं।

```
(*env)->SetStaticObjectField(env,
obj_this,id_owner,newOwner);
```

इसकी एप्लीकेशन को निम्न उदाहरण से समझा जा सकता है।

```
class TestCar77{
public static void main(String[] args){
Car c1 = new Car(“WagonR”,”Amit Kumar”,18 );

c1.SetOwner(“Amit Kuamr”);
System.out.println(“The owner is : ”+ c1.getOwner());

String newOwner = c1.updateOwner(“Anurita Khaskalam”)
Float distance = c1.getdistance(33.5);

System.out.println(“The new owner is ”+ newOwner);
System.out.println(“The owner is confirmed: ”+
c1.getOwner());
}
}
```

इस एप्लीकेशन का आउटपुट निम्न होगा

```
The Owner is : Amit kumar
The new Owner is : Anurita Khaskalam
The Owner is confirmed: Anurita Khaskalam
```

JNI SetStaticObjectField फंक्शन को ऑब्जेक्ट की वैल्यू सेट करने के लिए और GetStaticObjectField फंक्शन का उपयोग ऑब्जेक्ट की वैल्यू को प्राप्त करने के लिए और उपयोग किया जाता है । JNI में GetStatic--Field और SetStatic--Field फंक्शन डिफाइंड हैं जिसमें -- की जगह पर आवश्यकता के अनुसार Boolean, Byte, Char, Short, Int, Long ,Float, Double और Object में से कोई का भी उपयोग कर सकते हैं।

C से static जावा मेथड को कॉल करना (Calling non static Java methods from C):

इस सेक्शन में native फंक्शन की सहायता से नॉन static जावा मेथड को C से कॉल करने की प्रक्रिया को समझेंगे । इस उदाहरण में class Car की native मेथड printInitials()को जो कोई भी आर्गुमेंट प्राप्त नहीं करती है और न ही कोई वैल्यू रिटर्न करती है, को डिफाइन करेंगे । यह मेथड कॉल करने पर सिर्फ author के Initials को प्रिंट करती है।

```
Public native void printInitials();
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT void JNICALL Java_Car_ printInitials
(JNIEnv *env, jobject obj_this){

jclass class_Car =(*env)-> GetObjectClass(env, obj_this);

jfieldID id_Owner = (*env -> GetStaticFieldID(env,
class_Car,"owner","LJava/lang/String;");
// get the method ID

jmethodID id_getinitials = (*env -> GetMethodID(env,
class_Car,"getInitials","()LJava/lang/String;");

// call the method
Jstring in = (*env)->CallObjectMethod(env,obj_this,
id_getinitials);

// print Initials
char *str = (*env)->GetStringUTFChars(env,in,null);
printf("Initials of Owner:%s\n",str);
(*env)->ReleaseStringUTFChars(env,in,str);
}
```

सामान्य class प्राप्त करने के बाद यह आवश्यक है कि मेथड के आइडेंटिफायर को भी प्राप्त करें जिससे मेथड को कॉल किया जा सके।

```
jmethodID id_getinitials = (*env -> GetMethodID(env,
class_Car,"getInitials","()LJava/lang/String");
```

JNI फंक्शन GetMethodID(), आर्गुमेंट के द्वारा डिफाइन नॉन static मेथड के मेथड आइडेंटिफायर को रिटर्न करता है।

इसमें पहला आर्गुमेंट सामान्य environment है , दूसरा ऑब्जेक्ट की class तीसरा एक स्ट्रिंग है जो मेथड का नाम और चौथा signature या मेथड के प्रोटोटाइप को identify करता है। इसमें ()LJava/lang/String दर्शाता है कि मेथड कोई भी आर्गुमेंट प्राप्त नहीं कर कही है और Java.lang.String वैल्यू को रिटर्न कर रही है। मेथड getInitials() को निम्नानुसार कॉल किया जा सकता है

```
Jstring in = (*env)->CallObjectMethod(env,obj_this,
id_getinitials);
```

इसके द्वारा एक विशिष्ट ऑब्जेक्ट को मेथड आइडेंटिफायर से निर्दिष्ट मेसेज को फंक्शन CallObjectMethod() की सहायता से भेजा जाता है। यह फंक्शन मेथड को कॉल करता है और ऑब्जेक्ट के रिफरेन्स को रिटर्न करता है। इस केस में ये in है और ये मेथड getInitials() द्वारा रिटर्न जावा स्ट्रिंग को रेफ़र करेगा। अब स्ट्रिंग को सामान्य तरह से प्रिंट कर सकते है।

```
char *str = (*env)->GetStringUTFChars(env,in,null);
printf("Initials of Owner:%s\n",str);
(*env)->ReleaseStringUTFChars(env,in,str);
```

निम्न एप्लीकेशन से उपर्युक्त मेथड के उपयोग को समझ सकते है :

```
class TestCar88{
 public static void main(String[] args){
Car c1 = new Car("WagonR","Amit Kumar",18 );

c1.printInitials();
}
}
```

इसका आउटपुट होगा
A.K.

इस उदाहरण में CallObjectMethod() द्वारा मेथड को कॉल किया गया और यह मेथड ऑब्जेक्ट के इंस्टांस को रिटर्न करती है। JNI में दूसरे फंक्शन के द्वारा विभिन्न टाइप की वैल्यूज रिटर्न करना भी संभव है। इसके लिए JNI में फंक्शन का सेट call----method का उपलब्ध है और इससे ---- के स्थान पर विभिन्न टाइप जैसे Boolean, Byte, Char, Short, Int, Long , Float, Double और Object का उपयोग करके आवश्यकता के अनुसार रिटर्न वैल्यू प्राप्त की जा सकती है। निम्न सारिणी में कुछ मेथड और उसके Signature के उदाहरण देये गए हैं

Method	Signature
Int fun(inta,intb):	(II)I
boolean fun2(int a,char b,double c):	(ICD)Z
double fun3(String a,int b,float c):	(Ljava/lang/String:IF)D
String(int a,int b, Car c):	(IILCar)Ljava/lang/String:

इस सारिणी में Signature या prototype को डिफाइन किया गया है जहां पर कोष्टक के अन्दर आर्गुमेंट के Signature को और कोष्टक बहार रिटर्न वैल्यू के टाइप को डिफाइन किया गया है। अगले उदाहरण में class Car में एक native मेथड slower को डिफाइन किया गया है जो आर्गुमेंट के रूप में Car के रिफरेन्स को प्राप्त करती है और एक बूलियन वैल्यू रिटर्न करती है इस वैल्यू से यह पता चलता है कि रिसेप्टर प्राप्त Car, आर्गुमेंट से प्राप्त Car से slower है या नहीं :

```
public native boolean slower(Car other);
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT jboolean JNICALL Java_Car_ slower
(JNIEnv *env, jobject obj_this, jobject other){

jclass class_Car =(*env)-> GetObjectClass(env, obj_this);


jmethodID id_getDistance = (*env -> GetMethodID(env,
class_Car," getDistance","(F)F");

// call the method
Jfloat s1 = (*env)->CallFloatMethod(env,obj_this, id_
getDistance,500.0);
Jfloat s2 = (*env)->CallOFloatMethod(env,other, id_
getDistance,500.0);

if(s1<s2)
retarn JNI_True;
else
retarn JNI_False;
}
```

इसमें मेथड getDistance() को native मेथड के द्वारा दोनों इंस्टांस में कॉल किया गया है जहां पर ऑब्जेक्ट के रिसेप्टर को (this) और ऑब्जेक्ट के आर्गुमेंट के द्वारा other को कॉल किया गया है। इसमें ध्यान देने की बात है कि फंक्शन CallFloatMethod का उपयोग इसलिए किया गया है क्योंकि रिटर्न वैल्यू float टाइप की है।

निम्न एप्लीकेशन से उपर्युक्त मेथड के उपयोग को समझ सकते है :

```
class TestCar99{
public static void main(String[] args){
Car c1 = new Car(“WagonR”,”Amit Kumar”,18 );
Car c2 = new Car(“Alto”,”Anurag Kumar”,20 );
if(c1.slower(c2))
System.out.println(“Slower”);
else
System.out.println(“Faster”);
}
}
```

इसका आउटपुट निम्न होगा

```
Faster
```

C से static जावा मेथड की कालिंग(Calling Static Java Method from C):

हम यह जानते है कि कैसे static मेथड को जावा में कॉल किया जाता है अब हम देखेंगे कि native मेथड के द्वारा कैसे static मेथड को c के कॉल कर सकते हैं। अगले उदाहरण में class कार में एक native मेथड printDescription() को डिफाइन करेंगे जो न तो आर्गुमेंट को प्राप्त करती है और न ही कोइ वैल्यू रिटर्न करती है यह मेथड सिर्फ कार के डिस्क्रिप्शन को native मेथड printDescription() को कॉल करके प्रिंट करती है।

```
public native static void getDescription();
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT void JNICALL Java_Car_ printDescription2
(JNIEnv *env, jclass c1){

jclass class_Car =(*env)-> GetObjectClass(env, obj_this);


jmethodID id_desc = (*env -> GetStaticMethodID(env,
c1,”printDescription”,”()V”);
// call this method
(*env)->CallStaticVoidMethod(env,c1,id_desc);

}
```

यह आवश्यक है कि जिस मेथड को कॉल करना है उसका मेथड आइडेंटिफायर प्राप्त किया जाये उपर्युक्त उदाहरण में यह कार्य निम्नानुसार किया गया है

```
 jmethodID id_desc = (*env -> GetStaticMethodID(env, c1,"printDescription","()V");
```

इसमें पहला आर्गुमेंट सामान्य environment है , दूसरा ऑब्जेक्ट की class तीसरा एक स्ट्रिंग है जो मेथड का नाम और चौथा signature या मेथड के प्रोटोटाइप को identify करता है। इसमें "()V" दर्शाता है कि मेथड कोई भी आर्गुमेंट प्राप्त नहीं कर कही है और कोई भी वैल्यू को रिटर्न नहीं कर रही है।

अब निम्न स्टेटमेंट के द्वारा printDescription() मेथड को कॉल करेंगे

```
(*env)->CallStaticVoidMethod(env,c1,id_desc);
```

निम्न एप्लीकेशन से उपर्युक्त मेथड के उपयोग को समझ सकते है :

```
class TestCar101{
 public static void main(String[] args){
Car c1 = new Car("WagonR","Amit Kumar",18 );
C1. printDescription2();
}
}
```

इस उदाहरण में CallStaticVoidMethod () द्वारा मेथड को कॉल किया गया और यह मेथड कोई भी वैल्यू रिटर्न नहीं करती है। JNI में दूसरे फंक्शन के द्वारा विभिन्न पटाइप की वैल्यूज रिटर्न करना भी संभव है। इसके लिए JNI में फंक्शन का सेट callStatic----method का उपलब्ध है और इससे ---- के स्थान पर विभिन्न टाइप जैसे Boolean, Byte, Char, Short, Int, Long ,Float, Double और Object का उपयोग करके आवश्यकता के अनुसार रिटर्न वैल्यू प्राप्त की जा सकती है।

C से जावा constructor को कॉल करना (Calling Java Constructor from C):

एक native मेथड को डिफाइन करना है जो जावा constructor को काल करके class का ऑब्जेक्ट बनाये। निम्न उदाहरण में class Car में एक native मेथड createNewCar बनाया है जो model,owner एवं average को प्राप्त करके एक नयी Car को रिटर्न करेगी। यह मेथड class Car में उपलब्ध constructor में से

किसी एक को कॉल करेगी। हम यह जानते हैं की कौन सा constructor कॉल होगा यह इस बात पर निर्भर करता है कि कितने arguments हैं।

```
public native static Car creatNewCar(String mod, String own,
int ave);
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT jobject JNICALL Java_Car_ creatNewCar
(JNIEnv *env, jclass c1,jString mod,jString own,jint ave){

jclass class_Car =(*env)-> GetObjectClass(env, obj_this);


jmethodID id_constructor = (*env -> GetMethodID(env,
c1,"<init>","(LJava/lang/String; LJava/lang/String;I)V");


// bilt the object
jobject obj_new = (*env)->NewObject(env,c1,id_constructor,
mod,own,ave);
// Return it
return obj_new;
}
```

इसमें फंक्शन GetMethodID( ) के द्वारा constructor के लिए मेथड के आइडेंटिफायर को प्राप्त करते हैं।

```
jmethodID id_constructor = (*env -> GetMethodID(env,
c1,"<int>","(LJava/lang/String; LJava/lang/String;I)V");
```

इस मेथड का नाम init है जो की Java Virtual Machine(JVM) आतंरिक रूप से उपयोग होता है।

Class Car के पहले constructor का signature एक स्ट्रिंग "LJava/lang/String; LJava/lang/String;I)V" है।

constructor को निम्न प्रकार से कॉल कर सकते हैं

```
jobject obj_new = (*env)->NewObject(env,c1,id_constructor,
mod,own,ave);
```

constructor को कॉल करके फंक्शन NewObject द्वारा एक नया ऑब्जेक्ट बनाया जाता है। जिसका नाम तीसरे आर्गुमेंट से प्राप्त करता है। चौथे आर्गुमेंट से constructor के लिए पैरामीटर्स को प्राप्त करते हैं।

निम्न एप्लीकेशन से उपर्युक्त मेथड के उपयोग को समझ सकते है :

```
class TestCar102{
 public static void main(String[] args){
Car c1 = Car.createNewCar("This is xcent Car"," Avinash Kumar",19 );

System.out.println("Car:" + c1.mod +"," + c1.owner +"," + c1.average);
}
}
```

इसका आउटपुट होगा :

```
This is xcent Car, Avinash Kumar, 19
```

Arrays का उपयोग (Using Arrays) :

native मेथड के द्वारा array को भी access किया जा सकता है इस सेक्शन में हम इसी को समझेंगे। JNI में जावा array को c में उपयोग करने के लिए विभिन्न टाइप डिफाइन किये गए हैं।

Java Arrays	C Arrays
Byte []	jbyteArray
Char[]	jcharArray
Double []	jdoubleArray
Float[]	jfloatArray
Int[]	jintArray
Long[]	jlongArray
Object[]	jobjectArray
Short[]	jshortArray
Boolean[]	jbooleanArray

निम्न उदाहरण में Car class की एक native method को डिफाइन किया गया है यह मेथड arguments के द्वारा Car के array को प्राप्त करती है और उनमें से सबसे किफायती (affordable) Car को रिटर्न करती है। मेथड Car के एवरेज की

तुलना करती है और जिस Car का एवरेज ज्यादा होता है उसे किफायती बताकर रिटर्न करती है।

```
public native static Car creatNewCar(String mod, String own,
int ave);
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT jobject JNICALL Java_Car_affordable
(JNIEnv *env, jclass c1,jobjectArray Cars){

//Get the  FieldID
jfieldID    id_average    =    (*env    ->    GetFieldID(env,
c1,”average”,”I”);

// Get the Array Length

jsize length =  (*env -> GetArrayLength(env, Cars);

// transvers the array searching for the affordable Car

int I;
int affordable =0;
int newaverage =0;
for(i=o;i<length;i++)

// Get the ith Car
jobject Car = (*env -> GetObjectArrayElement(env, Cars,i);

// Get The Average of Car

jint average =(*env)-> GetIntField(env,Car, id_average);

// compare the two Cars

if(average > newaverage)
affordable =i;
newaverage = average;
}
}

// Return it

return (*env -> GetObjectArrayElement(env, Cars,affordable);
```

```
}
```

इसमें पैरामीटर Cars जो कि Cars के array को प्राप्त करती है , और उसको jobjectArray टाइप के द्वारा डिफाइन किया है।

```
jsize length =  (*env -> GetArrayLength(env, Cars);
```

फंक्शन GetArrayLength()एक वालू return करेगा जो jsize टाइप का होगा और array ली length को बताएगा।
array का i वां ऑब्जेक्ट निम्न प्रकार से प्राप्त किया जा सकता है :

```
jobject Car = (*env -> GetObjectArrayElement(env, Cars,i);
```

फंक्शन GetObjectArrayElement( ) तीन आर्गुमेंट के प्राप्त करता है। इसमें पहला आर्गुमेंट सामान्य environment है , दूसरा array का ऑब्जेक्ट और तीसरा पद है। यह फंक्शन ऐसे ऑब्जेक्ट के रिफरेन्स को रिटर्न करता है , जो array में विशिष्ट पद पर स्टोर है।
array के अन्दर Car का Average सामान्य तरीके से GetIntField( ) फंक्शन के द्वारा प्राप्त किया जा सकता है।

```
jint average =(*env)-> GetIntField(env,Car, id_average);
```

affordable Car रिफरेन्स को GetObjectArrayElement( ) फंक्शन के द्वारा रिटर्न किया जाता है।

```
return (*env -> GetObjectArrayElement(env, Cars,affordable);
```

निम्न एप्लीकेशन से उपर्युक्त मेथड के उपयोग को समझ सकते है :

```
class TestCar103{
 public static void main(String[] args){
Car c1 =  New Car(“xcent”,” Avinash Kumar”,19 );
Car c2 =  New Car(“WagonR”,” Amit Kumar”,18 );
Car c3 =  New Car(“Alto”,” Aurag Kumar”,21 );

Car[] cars = {c1,c2,c3};

Car affordable = Car.affordable(Cars);
```

```
System.out.println("Affordable Car is :" + c1.mod + affordable.model);
}
}
```

इस एप्लीकेशन का आउटपुट

```
Affordable Car is : Alto
```

JNI में array के एलेमेंट्स को संशोधित करने के लिए फंक्शन SetObjectArrayElement() का उपयोग किया जा सकता है। इस फंक्शन में एक अतिरिक्त पैरामीटर रिफरेन्स है जो array के विशिष्ट पद के लिए असाइन होता है। ये मेथोड्स का उपयोग जावा array के ऑब्जेक्ट के किसी भी एलिमेंट को access करने के लिए होता है। जावा के अन्य टाइप के array के बराबर C के array में बदलने के लिया JNI में फंक्शन का सेट उपलब्ध है। और इसी तरह से array की प्रक्रिया कुशल C instructions की सहायता से की जा सकती है। अगले उदाहरण में एक मेथड lower को डिफाइन किया गया है जो array main integer तत्वों को आर्गुमेंट के द्वारा प्राप्त करती है और कितनी Cars का एवरेज दिए हुए एवरेज से कम है उनकी संख्या बताती है। और इसमें ध्यान देने की बात यह है कि ये मेथड पहले जावा array को c array में बदलती है उसके बात प्रक्रिया करती है।

```
public native int lower (int[],numbers);
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT jint JNICALL Java_Car_lower
(JNIEnv *env, jobject,obj_this,jintArray numbers){

//Get the  FieldID
jfieldID    id_average    =    (*env    ->    GetFieldID(env, c1,"average","I");

// Get the class

jclass class_Car =  (*env -> GetObjectClass(env, obj_this);

// Get the Field ID
```

```
jfieldID id_average = (*env -> GetFieldID(env, class_Car,”average”.”I”);

// Get the Field Value
int average = (*env -> GetIntField(env,obj_this,id_average);

// Get The Size of Array

jsize length =  (*env -> GetArrayLength(env, numbers);


// Get the standard C Array

int *a = (*env)-> GetIntArrayElement(env,numbers,NULL);

// compute
int i,count=0;
for(i=0, i<length,i++){
if (a[i] < average)
count ++;
}

// Release the array
(*env)->ReleaseIntArrayElement(env,numbers,a,0);

// Return the value
return  count;
}
```

फंक्शन GetIntArrayElement() का उपयोग जावा के jint वैल्यूज के array को C के integer array में बदलने के लिए किया जा सकता है :

```
int *a = (*env)-> GetIntArrayElement(env,numbers,NULL);
```

assignment के बाद जावा array , वेरिएबल a सामान्य रूप से उपयोग किया जा सकते हैं। गतिशील (Dynamically) तरह से एसाइन्ड मेमोरी को निम्न फंक्शन ReleaseIntArrayElement() को कॉल करके मुक्त किया जाता है।

```
 (*env)->ReleaseIntArrayElement(env,numbers,a,0);
```

निम्न एप्लीकेशन से उपर्युक्त मेथड के उपयोग को समझ सकते है :

```
class TestCar104{
public static void main(String[] args){
```

```
Car c1 =  New Car(“Tiago”,” Aseem ”,21 );
int numbers = {16,17,18,19,20,21,22}
int num = c1.lower(numbers)

System.out.println(“Result:” + num);
}
}
```

इसका आउटपुट निम्न आएगा

```
Result: 4
```

इसके विभिन्न प्रकार के array के तत्वों पर प्रक्रिया करने के लिए JNI में फंक्शन का सेट Get----ArrayElement उपलब्ध है और इससे ---- के स्थान पर विभिन्न टाइप जैसे Boolean,Byte, Char, Short, Int, Long, Float, Double और Object का उपयोग करके आवश्यकता के अनुसार रिटर्न वैल्यू प्राप्त की जा सकती है।

निम्न उदाहरण में array ले तत्वों को कैसे संशोधित किया जाये इसका वर्णन है। इसमें एक मेथड getaverage को डिफाइन किया गया है जो कार के array और integer के array को आर्गुमेंट के द्वारा प्राप्त करती है और कोई भी वैल्यू रिटर्न नहीं करती। यह मेथड प्रत्येक कार के एवरेज को तदनुसार int array में विशिष्ट पद पर स्टोर करती है।

```
public native static void getaverage(Car[], cars, int[] numbers);
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT void JNICALL Java_Car_getaverage
(JNIEnv *env, jclass c1,jobjectArray cars,jintArray numbers){

//Get the  FieldID
jfieldID id_average = (*env -> GetFieldID(env, c1,”average”,”I”);

// Get the Array length

jsize length =  (*env -> GetArrayLength(env, cars);

// transvers the array searching for the affordable Car
```

```
int i;

for(i=o;i<length;i++){

// get the ith Car

jobject car= (*env)-> GetObjectArrayElement(env,cars,i);

// Get the average

int average = (*env -> GetIntField(env,car,id_average);

// store into array
a[i] = average;
}

// Release the array
(*env)->ReleaseIntArrayElement(env,numbers,a,0);
}
```

फंक्शन native cars के array को transvers करता है और एवरेज (average) को तदनुसार पद पर integer array a में स्टोर करता है । अंत में फंक्शन ReleaseIntArrayElement() को कॉल किया जाता है जो array के द्वारा उपयोग की गयी मेमोरी को मुक्त करता है और साथ साथ c array के तत्वों को जावा array में कॉपी भी करता है।

निम्न एप्लीकेशन से उपर्युक्त मेथड के उपयोग को समझ सकते है :

```
class TestCar105{
 public static void main(String[] args){
Car c1 =  New Car(“Tiago”,” Aseem ”,21 );
Car c2 =  New Car(“xcent”,” Avinash Kumar”,19 );
Car c3 =  New Car(“WagonR”,” Amit Kumar”,18 );
Car c4 =  New Car(“Alto”,” Aurag Kumar”,22 );

Car[] cars = {c1,c2,c3,c4};
int[] numbers = new int [4];

Car.getaverage(cars,numbers);

for(int i=0,i<4,i++)
System.out.println(“Car:” + i+”:” +numbers[i]);
}
```

```
}
```

इसका आउटपुट निम्न आएगा

```
Car 0 : 21
Car 1 : 19
Car 2 : 18
Car 3 : 22
```

अपवाद (Eceptions):

यह भी संभव है कि C native फंक्शन Eceptions को throw किया जाये और उन्हें जावा मेथड को प्रक्रिया करने के लिए भेजा जाये। निम्न उदाहरण में एक मेथड writeFile को डिफाइन किया गया है जो कि कोई भी आर्गुमेंट प्राप्त नहीं करती है और कोई भी वैल्यू रिटर्न भी नहीं करती है परन्तु ये मेथड IOException को throw कर सकती है। उपर्युक native मेथड Car के मॉडल को write करती है। और Eceptions को throw करती है कि कहीं कुछ गलत न हो जाए।

```
public native  void writeFile()throws Exception;
```

C में फंक्शन को निम्नानुसार डिफाइन करेंगे

```
JNIEXPORT void JNICALL Java_Car_writeFile
(JNIEnv *env, jobject, obj_this){

Void throws_IO_Exception (JNIEnv *);

// Get the class
jclass class_Car = (*env -> GetObjectClass(env, obj_this);

//Get the  Field ID
jfieldID  id_model  =  (*env  ->  GetFieldID(env,  class  Car,
“model”,”Ljava/lang/String;”);

// get the field value

jstring          model                    =(*env          ->
GetObjectField(env,obj_this,id_model);

char *str =  =(*env -> GetStringUTFChars(env,model,NULL);
FILE *fp;
if( fp= fopen(“car.text,”w”)) == NULL {

throws_IO_Exception (env);
```

```
(*env)->ReleaseStringUTFChars(env,model,str);
return;
}

if (fprintf(fp,"%s\n",str) != strlen(str)+1){
throws_IO_Exception (env);
(*env)->ReleaseStringUTFChars(env,model,str);
return;
}
(*env)->ReleaseStringUTFChars(env,model,str);

if(fclose(fp)){
throws_IO_Exception (env);
 return;
}
}
```

इसमें native फंक्शन स्ट्रिंग वैल्यू को फील्ड model से प्राप्त करता है फिर फाइल को ओपन टेक्स्ट करता है उसके बाद स्ट्रिंग को फाइल में लिखता है। और अंत में फाइल को close कर देता है । कोई भी error आने पर throws_IO_Exception (env)फंक्शन कॉल होता है और Exception को throw करता है। फंक्शन को निम्न प्रकार से डिफाइन कर सकते हैं।

```
void throws_IO_Exception (JNIEnv,*env){

// get the IO Exception class

jclass            class_IO_exc            =(*env            ->
FindClass(env,"java/io/IOException");

// get constructor method

jmethodID   id_IO_exc   =   =(*env   ->   GetMethodID(env,
class_IO_exc, "<init>","()V");
 // Create Object
jthrowable obj_exc = =(*env -> NewObject(env, class_IO_exc,
id_IO_exc);


// throw the Exception

(*env)-> Throw (env,obj_exc);
```

```
}
```

जब भी हूँ Exception को throw करना होता है तब class IOException के ऑब्जेक्ट को बनाना होता है। इसे करने के लिए उस class के constructor को कॉल करते हैं। सबसे पहले class Exception को प्राप्त करते हैं जो की निम्नानुसार किया जा सकता है

```
jclass class_IO_exc =(*env -> FindClass env,"
java/io/IOException");
```

इसके बाद फंक्शन, constructor की मेथड (<init>)आइडेंटिफायर को प्राप्त करता है।

```
jmethodID id_IO_exc = =(*env -> GetMethodID(env,
class_IO_exc, "<init>","()V");
```

Exception के ऑब्जेक्ट को फंक्शन NewObject()के द्वारा बनाते हैं।

```
jthrowable obj_exc = =(*env -> NewObject(env, class_IO_exc,
id_IO_exc);
```

और अंत में throw फंक्शन से Exception को throw करते हैं।

```
(*env)-> Throw (env,obj_exc);
```

इस फंक्शन के execution throw के बाद अपने आप native मेथड का समापन नहीं होता है और native मेथड के समापन के लिए ही execution के throw होने के बाद return का उपयोग किया गया है।

निम्न एप्लीकेशन से उपर्युक्त मेथड के उपयोग को समझ सकते है :

```
class TestCar106{
public static void main(String[] args){
Car c1 =  New Car("Tiago"," Aseem ",21 );

try{
ci.writefile();
System.out.println(" File Written ");
}catch (IOException e){
System.out.println(" Input/Output Error");
}
```

```
}
}
```

इसका आउटपुट निम्न होगा
```
File Written
```

और यदि कोई error आती है तब आउटपुट होगा
```
Input/Output Error
```

यह भी संभव है कि एक फंक्शन के सहायता से Exception को निम्न प्रकार से बनाया जा सकता है।

```
(*env)-> Thrownew (env, =(*env -> FindClass(env,
"java/io/IOException","A problem");
```

इसमें अंतिम आर्गुमेंट एक स्ट्रिंग है जो Exception के कारण को बताता है इसे जावा की मेथड toStringMessage() का उपयोग करके भी प्राप्त किया जा सकता है। JNI एक और फंक्शन ExceptionOcurred() को उपलब्ध करता है जो किसी विशिष्ट Exception को Throw करता है और फंक्शन ExceptionClear() का उपयोग लंबित Exception समाप्त करता है।

एन्ट (Ant):

Ant एक निर्माण उपकरण है जिसकी सहायता से जावा एप्लीकेशन को कम्पाइल(Compile), पैक (pack) , डिप्लॉय (Deploy) एवं डॉक्यूमेंटेशन किया जाता है। एक तरह से Ant की उपयोगिता make कमांड जैसी है परन्तु इनके द्रष्टिकोण में अंतर है। make कमांड के विशेष विवरण का संगठन एक shell कमांड्स का सेट है और Ant कमांड के विशेष विवरण XML सेंटेंस के सेट के रूप में पारिभाषित होते हैं। Build फाइल एक प्रोजेक्ट को परिभाषित करती है जिसमें कामों (tasks) का सेट होता है। बिल्डिंग प्रोसेस को execute करने के लिए tasks के लक्ष्य को कॉल करना होता है। यहाँ पर एक ऑब्जेक्ट के द्वारा एक task को रन करते हैं जो एक विशिष्ट task interface को इम्प्लीमेंट करते हैं। इस द्रष्टिकोण का मुख्य लाभ यह है कि बिल्ड फाइल्स ऑपरेटिंग सिस्टम से स्वतंत्र है इसके साथ ही एक हानि भी है कि इस प्रोसेस में shell कमांड्स का उपयोग नहीं ही सकता है।
Ant का विशिष्ट विवरण सामान्यतः build.xml फाइल में होता है।

Ant का पहला उदाहरण:

इस उदाहरण में एक फाइल HelloWorld.java को लेकर Ant का उपयोग समझेंगे। build.xml की एक संभव परिभाषा निम्न है

```
<?xml version = "1.0">
<!---- First build file ---->

<project name ="HelloWorld" default ="build" basedir=".">
       <target name="build">
                  <javac srcdir = "."/>
       </target>
</project>
```

पहली लाइन यह बताती है कि यह फाइल xml फाइल होगी। दूसरी लाइन बताती है कि यह लाइन और इस फाइल का पूरा कार्य एक प्रोजेक्ट के रूप में होगा जिसका नाम HelloWorld है और इसका डिफ़ॉल्ट टारगेट build है । सभी डायरेक्ट्रीज को basedir से रिलेटिव मन जाता है प्रोजेक्ट को execute करने के लिए एक टारगेट एक नाम build का उपयोग करते हैं

यह टारगेट एक task एलिमेंट को शामिल करता है जो वर्तमान डायरेक्टरी की सभी जावा सोर्स फाइल को कम्पाइल करता है। फाइल को कम्पाइल करने के लिए सिर्फ Ant कमांड से ही काम चल जाता है

```
# Ant
Buildfile:build.xml
Build:
       [javac] compiling 1 source file

BUILD SUCCESSFUL
TOTAL TILE 2 SEC
```

वस्तुनिष्ट प्रश्न

१. मोडुलस ऑपरेटर (%) निम्न में से किस पर लगा सकते हैं

   अ. इन्टिजर ब. फ्लोटिंग पॉइंट स. इन्टिजर और फ्लोटिंग पॉइंट दोनों पर

   द. इनमें से कोई नहीं

२. Decrement ऑपरेटर (--) वेरिएबल की वैल्यू कितनी कम करता है

   अ. 1 ब. 2 स. 3 द. 4

३. क्या 8 बाइट के long डाटा को ऑटोमेटिकली 4 बाइट float में बदला जा सकता है

अ. हाँ ब. नहीं

४. जावा में कितने प्रकार के primitive डाटा टाइप होते हैं

अ. 6 ब. 8 स. 16 द. 4

५. जावा में float और double का साइज़ क्या है

अ. 32 और 64 ब. 32 और 32 स. 64 और 64 द. 64 और 32

६. जब एक array मेथड में पास होता है तब मेथड क्या प्राप्त करती है

अ. array का रिफरेन्स ब. array की कॉपी स. array की लेंथ द. array के पहले अवयव की कॉपी

७. Array को डिक्लेअर एवं इनिशियलाइज़ करने का सही तरीका

अ. int[] A = {} ब. int[] A = {1,2,3,4} स. int[] A = (1,2,3,4)

द. int[] [] A = {1,2,3,4}

८. static मेथड पर निम्न में से कौन सा प्रतिबन्ध है

I. static मेथड केवल static डाटा ही एक्सेप्ट करती हैं

II. static मेथड केवल static मेथड को ही कॉल करती हैं

III. static मेथड this या super को रेफ़र नहीं करती हैं

अ. I और II ब. II और III स. III द. I,II और III

९. class String निम्न में से किस में उपलब्ध है

अ. java.lang ब. Java.awt स. Java.applet द. Java.string

१०. रिटर्न टाइप को पहचाने जिसमें मेथड कोई भी वैल्यू रिटर्न नहीं करती है

अ. int ब. float स. void द. कोई नहीं

११. निम्न में से कौन सा आइडेंटिफायर constructor के लिए उपयोग नहीं होता है

अ. public ब. protected स. private द. static

१२. डिफ़ॉल्ट constructor के प्रोटोटाइप को पहचाने

```
public class solution {}
```

अ. Solution(void) ब. public Solution(void)

स. Solution( ) द. public Solution( )

१३. जावा के कोड में एक्सेप्शन कब आता है

अ. run time ब. compilation time

स. किसी भी समय द. उपर्युक में से कोई नहीं

१४. निम्न में से कौन सा कीवर्ड एक्सेप्शन को मॉनिटर करने के लिए उपयोग होता है

अ. try ब. finally स. throw द. catch

१५. निम्न में से कौन सा bitwise ऑपरेटर नहीं है

अ. & ब. &= स. |= द. <=

१६. निम्न में से कौन सा राईट शिफ्ट ऑपरेटर sign को संरक्षित करता है

अ. >> ब. >>> स. <<= द. >>=

१७. निम्न में से कौन सा स्टेटमेंट केवल एकुलिटी चेक करता है

अ. if ब. switch स. if & switch द. इनमें से कोई नहीं

१८. निम्न में से कौन सा लूप कंडीशन false होने पर भी एक्सीक्यूट होता है

अ. do-while ब. while स. for द. कोई नहीं

१९. निम्न में से कौन से डाटा टाइप्स literals होसकते हैं

अ. integer ब. float स. boolean द. उपर्युक्त सभी

२०. एक class Box के ऑब्जेक्ट को डिक्लेअर करने का सही तरीका कौन सा है

अ. Box obj =new Box(); ब. Box obj =new Box;

स. obj =new Box(); द. new Box();

२१. ऑब्जेक्ट के लिए मेमोरी allocate करने के लिए कौनसे ऑपरेटर का उपयोग होता है

अ. malloc ब. alloc स. new द. give

२२. abstract class को परिभाषित करने के लिए कौनसे कीवर्ड का उपयोग होता है

अ. abst ब. abstract स. Abstract द. Abstract class

२३. किसी क्लास में एक abstract class को इन्हेरिट करके उसके सभी फंक्शन को परिभषित नहीं करते है तब वह class क्या कहलाएगी

अ. Abstract ब. A simple class स. static class द. इनमें से कोई नहीं

२४. abstract कीवर्ड निम्न पैकेज में से किस में पाया जाता है

अ. java.lang ब. java.util स. java.io द. Java.system

२५. जावा में किसी class को इन्हेरिट करने के लिए कौनसे कीवर्ड का उपयोग होता है

अ. super ब. this स. extent द. extends

२६. एक class मेम्बर को protected परिभाषित करें तब वह subclass में कैसा मेम्बर होगा

अ. public member ब. private member

स. protected member द. Static member

२७. यदि किसी एक्सप्रेशन में double, integer, float और long टाइप के डाटा है तब पूरी एक्सप्रेशन

का टाइप क्या होगा

अ. integer ब. float स. double द. long

२८. थ्रेड को शुरू करने के लिए निम्न में से कौनसी मेथड का उपयोग होता है

अ. run() ब. start() स. runThreads() द. startThreads()

२९. जावा में निम्न में से कौन सा पोलिमोर्फिस्म उपलब्ध है

अ. Compile time polymorphism ब. Execution time polymorphism

स. Multiple polymorphism द. Multilevel polymorphism

३०. जावा में मेथड ओवरलोडिंग कब निर्धारित होती है

अ. At run time ब. At compile time

स. At coding time द. At execution time

३१. एक class के अन्दर मेथोड्स और ऐट्रिब्यूट्स को परिभाषित करके जावा की कौन सी अवधारणा को प्रप्तिकया जाता है

अ. Encapsulation ब. Inheritance स. Polymorphism द. Abstraction

३२. निम्न में से कौनसा अवयव जावा प्रोग्राम को कम्पाइल , डिबग और एक्सीक्यूट करने

के लिए उपयोग होता है

अ. JVM ब. JDK स. JIT द. JRE

३३. निम्न में से कौनसा अवयव जावा प्रोग्राम को RUN करने के लिए उपयोग होता है

अ. JVM ब. JDK स. JIT द. JRE

३४. main() मेथड में निम्न में से कौनसे आइडेंटिफायर का उपयोग नहीं हो सकता है

अ. public ब. private स. static द. final

३५. जावा की कम्पाइल की हुई फाइल का एक्सटेंसन क्या होगा

अ. .class ब. .java स. .text द. .js

३६. रिलेशनल ऑपरेटर का आउटपुट क्या होगा

अ. Integer ब. Boolean स. Character द. Double

३७. जावा में कौनसे कीवर्ड से वेरिएबल की वैल्यू को बदलने से रोका जा सकता है

अ. final ब. last स. constant द. static

३८. निम्न में से किसको static डिक्लेअर नहीं कर सकते हैं

अ. class ब. object स. variable द. method

३९. abstract class का constructor नहीं हो सकता

अ. True ब. False

४०. JIT का फुलफॉर्म निम्न में से क्या है

अ.Just in Temporary ब. Just in Time

स. Jump in Time द. इनमें से कोई नहीं

सन्दर्भ :

1. Introduction to Java by Carlos Kavka
2. Object Oriented Programming Principles by Olexiy Tykhomyrov
3. https://www.interviewbit.com/java
4. https://www.sanfoundry.com/java

No Objection Certificate

(i) I, Dr. Amit Kumar Khaskalam *(name of the author/co-author)* represent and warrant that I am the sole owner of all copyright, trademark, and other intellectual property and proprietary rights in relation to the book/material.

(ii) I, Dr. Amit Kumar Khaskalam (name of the author/co-author) undertake that the Book/Material is not subject to any contract or arrangement which would conflict with my permission herein.

(iii) This is to certify that the undersigned hereby gives permission to UGC and also authorizes UGC to get it translated or published and uploaded on e-kumbh portal जावा प्रोग्रामिंग (name of the book). The book authored by the undersigned and translated using anuvadini tool has been properly vetted by me and it can be sent for publishing on e-kumbh portal.

(iv) UGC will have the full right to publish the book/text and is authorized to do any modifications, republication, or any other assistance related to the text if required.

(v) No legal action will be taken by the author in this regard.

Signature
Author/co-author

Date 18-07-2023

Place Bilaspur (C.G.)