**Java**

**Topic**

1. Identifier
2. Reserved Words
3. Datatypes
4. Literals
5. Arrays
6. Types of variables
7. Var-arg methods
8. Main method
9. Command line arguments
10. Java Coding standards
11. Operators and assignments
12. Flow controls 100
13. Declarations and Access modifiers 124
14. Import and static import
15. Packages 137
16. Abstract Class
17. Interfaces
18. Marker Interface

OOPS Concept

1. Data hiding
2. Abstraction
3. Tightly encapsulated class
4. Encapsulation
5. Inheritance
6. Overloading
7. Overriding
8. Polymorphism
9. Blocks
10. Constructors
11. Super and this
12. Final
13. Constructor Overloading
14. Type casting
15. Coupling and cohesion
16. Regular Expressions 164
17. StringTokenizer
18. Enum, enum Vs Enumeration
19. Create jar file
20. JDK , JRE and JVM
21. Classpath and Path
22. Garbage Collection
23. Memory Leak
24. Exception Handling
25. Inner Classes
26. Object Classes
27. String, StringBuffer and StringBuilder
28. Wrapper classes(Autoboxing and Unboxing)
29. Serialization 388
30. File handling 364
31. Generics 90
32. Collection 1
33. Multithreading 110

1)java is both a programming language &platform for application development.

2)java is object oriented,i.e. every java program requires the presence of atleast one class(an object oriented feature).

3)java is platform independent it means write once run anywhere because we can executes java file in any platform where JVM is available.

4)java is very simple compared with C & C++ because it much easier to write bug free code.half of the bugs in C & C++ program are related to memory allocation & de-allocation but java provides bug free code is automatic memory allocation & de-allocation.

5)java is safe because ,we can’t use pointer ,can’t access address in memory.automatically java will do memory allocation.

6)java is high performance cooz java use JIT compiler for compilation

7)java provides garbage collector for memory allocation & de-allocation.

8)java is written in editor,text editor or char editor.we can write in notepad also.

9)editor is a software which only takes text char.eg-notepad,editplus,notepad++,textpad etc.

10)a platform is an underlying computer system on which application programs can run.

11)platform can be of two types

->hw+sw based platform.eg.OS

->sw based platform.eg.JAVA,.NET platform

12)java is only software based platform.it runs on some specific hardware based OS like UNIX,Windowsetc.

13)like Java many languages are there like ada,ruby,python & javascript

14)like JDK ,koan software is for ruby,python interpreter for python .

**Three releases in java**

1)J2SE(java2 standard edition)

2)J2EE(java2 enterprise edition)

3)J2ME(java2 micro edition)

J2SE is only used for core java application.

J2EE is used for building web application.

J2ME is used for developing a mobile application.

**The Development Process**

It is done in 3 stages.

1)**development**->writing a java program is known as development stage.

->this java program is written in a file with extension .java.this file is also called source file.

2)**compilation**->converting source file into executable format is known as compilation.

->the java compiler converts the java program into class file.

->the class file made up of byte codes and it has extension of .class.

3)**execution**->the byte codes are executed by JVM.this stage is known as execution.

->the JVM is an interpreter which converts each line of byte code to the MLL & then executes it.

->a class file can be executed in any platform where JVM is available.

**Error**

1)syntax error->compile time errors

2)logical error->runtime errors

**Language**

* High level language(HLL)
* Low level language or machine level language(LLL)

Program(statements)-HLLTranslator(Instruction)-MLLCPU

**Translator**->converts HLL to MLL

* Compiler->takes all statements written at one short to convert HLL. programming language use compiler.java is a compiler based language.
* Interpreter->takes each statement one by one and convert in to MLL. scripting language use interpreter(Javascript)

JAVA FILE(program)(.java)JAVA COMPILERCLASS FILE(byte code)(.class)JVMMLLCPU

1)java compiler converts java program in to byte code.java file contains java program.java file extension is (.java).

2)java compiler generate a byte code in extension i.e. (.class).

3)JVM converts bytecode into specific platform CPU MLL.JVM acts as an interpreter.

|  |  |  |
| --- | --- | --- |
|  | JDK |  |
| Compiler | Java Application Launcher  Opens the JRE,loads the class file & invokes main method. | Applet View |
|  | JRE contains JVM+Java Packages classes+runtimes libraries |  |

**JRE**java runtime environment.JVM is available inside the JRE.

**JDK**java development kit.it provides both JRE & JVM.

**Java Versions**

Java 1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7

Java became popular with 1.2 version.

JDK5 represents JDK1.5.

java means coffee beans.java was first developed for embedded application i.e. electronic application.

at java 1.2 started enhancing web application.

development or implementation is a step where we write a java program.

then compilation->java program to bytecode or java file to class file.

execution means bytecode to MLL.

source file is a file with code.in case of java its java file.

java file is converted to class file or binary file or executed file.

.class file is executable format of java file.

**JDK Installation**

1)download JDK1.7 and install & save it in any location.

2)after installation get path from bin folder.

3)then add the in to environment variable path.

4)verify java installation is success in cmd “java –version”.

5)to change drive->drivename:

6)to change directory->cd foldername\foldername\

7)compilation->open the command prompt and go to the location where the java file is available.

->run the command with following syntax.

javac –d ..\bin filename.java

javac –d ..\bin com\jspider\practise\Demo1.java

->while compiling the .java extension is mandatory else compiler throws an error.

->after compilation the compiler generates .class file and stores in location where java file is available.

->this is default behavior of java compiler.

8)press enter no message is shown then class file generate successfully.

9)for execution->open command prompt,go to location where class file is available.

->execute class file by running java command with following syntax.

Cd ..\bin

javafilename

java com.jspider.practise.Demo1

->during execution .class is not required ,if provided then JVM will throw an error.

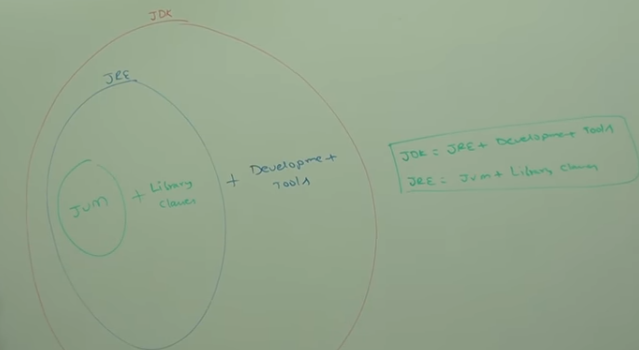
Note->we can develop and compile empty class In java.

->JVM executes only those class which contains main method.

->+ is a concatenation operator which join two strings or string & variable.

**JDK,JRE and JVM**

* JDK provide environment to develop and run java application . EX. Developer machine
* JRE provide environment to run the java application. EX. Client machine
* JVM is interpreter that is used to run the java program line by line.



**ClassPath and Path**

* Path where .class file is available that is called Classpath. This is required for compiler and JVM.
* Path location describes where binary executables are available. If we are calling javac command like >javac Test.java then internally it will call javac.exe file. So for that it will check the path location.

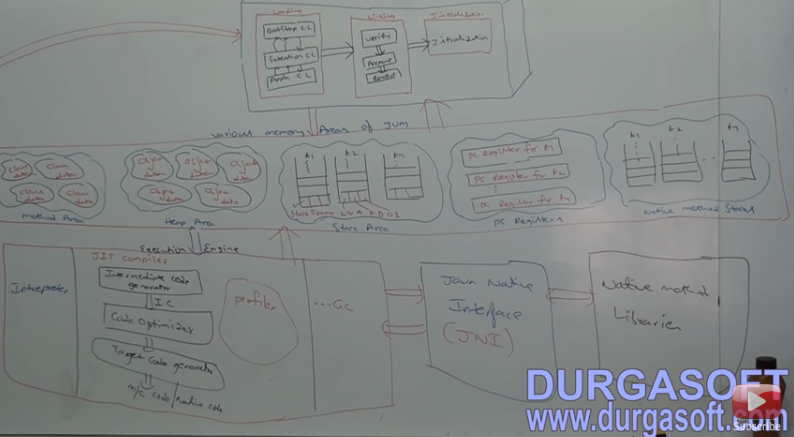
**Java, javaw and Javaws**

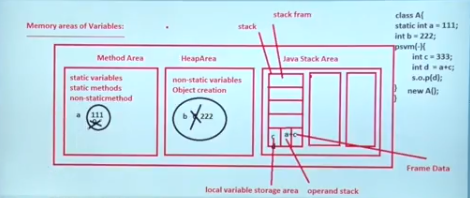
* Java is used to run the class file and print all like console output or file output.
* Javaw is used to display output without console.Ex.used in GUI application.
* Javaws(java web start utility) is used to download and launch the application.Ex.run jnlp file.

**How many ways to run a java program**

* Java Test;
* Java –jar test.jar
* Double click on jar file.(for creating jar file run the command > jar -cvfm demo.jar manifest.MF JarDemo.class JarDemo$1.class and for run the jar file >java –jar demo.jar)
* Double click on batch file(run too many commands so write into text file and save as .bat file).

**JVM Architecture**





**StackOverFlowError**

* This error will occur when our application uses recursive methods or large amount of data is fetched from server to client side and java object doesn’t maintain. Hence our stack memory will become full at run time and JVM will throw this error.
* **Solution** : inspect the stack trace and detect the repeating pattern of line numbers. Or increase the thread stack size using the [-Xss](http://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/optionX.html) flag with project configuration or via command line.

**OutOfMemoryError**

* When application try to add more data into heap area and no space is available the this issue will come.
* Heap size and PermGen(permanent generation) size is set during the JVM launch and can be customized by giving parameters –Xms, –Xmx and –XX:MaxPermSize.

**Memory Leak**

* memory leak in Java is a situation where some objects are not used by application any more, but GC fails to recognize them as unused.

**Reasons**

* if you have used some native objects and forgot to reclaim the memory explicitly then GC doesn’t delete that objects and cause memory leaks.
* using 'static' also can be one of the potential reasons for memory leaks. if we have not set to null then that will live for the entire life of an app unless they are explicitly set to 'null'.
* scope of an object to restrict its usage.

**Verbose**

* -verbose:class : used to display the information about classes that is loaded by JVM.
* -verbose : gc : Used to check garbage collection event information.
* -verbose:jni : Used to display native method.

**Memory**

Two types of memory->

1)**read/write memory**-write multiple & read multiple(facebook,files).types below.

* RAM->when power is goes off then data will be erased

->used for execution purpose

->fast memory

->called physical main memory

->designed by Intergated chips(IC’S)

* HARD DISK DRIVE->when power is goes off then data won’t be erased.

->used for storage purpose.

->slow memory

->virtual memory

->designed by magnetic tapes

2) **Read only memory**-write once & read multiple times(washing machine,refrigerator,tv/setup box,booting software.

**Memory utilization by Java using Static & Non Static**

When a java starts excuting a class then different process will happen.

* JVM will do Memory allocation.
* Splits memory-stack-execution purpose i.e. FIFO.

Heap-storage purpose

3)JVM gives a call to class loader.

4)class loader loads all static members in heap memory i.e called static pool.static member contains

only one copy in memory i.e. reffered by class thtsy known as class member.

5)for non static member,JVM creates an object in heap & that is loaded in object i.e. referred by

reference variable.this contain multiple copy in memory.called instance member & object member.

5)java calls main method & excutes all the statements.then its came out from stack..

6) then JVM calls garbage collector for clean the heap memory before terminating

**Programming**

**Language**

1)**Identifier**->A name in java program is called identifier. It is used for identification purpose. It can be method name or variable name or class name or label name.

**Rules for defining java identifiers**

1) Only allowed character in java identifiers are a to z,A to Z,0-9,$,\_.

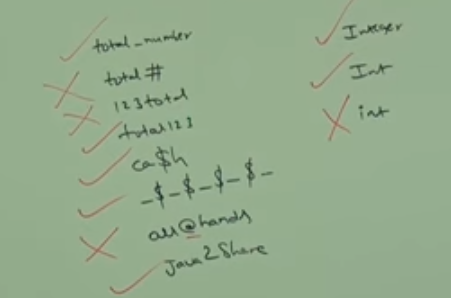
2) it shouldn’t start with a number.e.g.total123-variablename,classname,methodname

3) It is case sensitive.java language is treated as case sensitive programming language.

4) No length limit for identifiers but not recommended to take too lengthy identifiers.

5) can’t use keywords(reserved words) as identifier.

6) can use predefined class name or interface name as identifier. but not recommended.



1. **keywords**->In java some words are reserved words to represent some meaning or functionality,such type of words are called reserved words. There are 53 reserved words. case sensitive, must be in lower case.

**Keyword for Datatypes** : byte,short,int,long,float,double,Boolean,char(8)

**Keyword for Flow Control** : if,else,switch,case,default,while,do,for,break,continue,return(11)

**Keyword for Modifiers :** public,private,protected,static,final,abstract,synchronized,native,strictfp,transient,volatile(11)

**Keyword for Exception Handling :** try,catch,finally,throw,throws,assert(6)

**Keyword for Class Related :** class,interface,extends,implements,package,import(6)

**Object Related Keywords** : new,instanceof,super,this(4)

**Void Return Type Keyword** : void(1)return type is mandatory in java. but in c language return type is optional and default return type is int.

**Unused Keywords** : goto(usage of goto created several problems in old languages and hence sun people banned this keyword in java),const(use final instead of const)(2)

Note : goto and const are unused keywords and if we are trying to used we will get compile time error.

**Reserved Literals** : true,false(values for Boolean datatype),null(default value for object reference).(3)

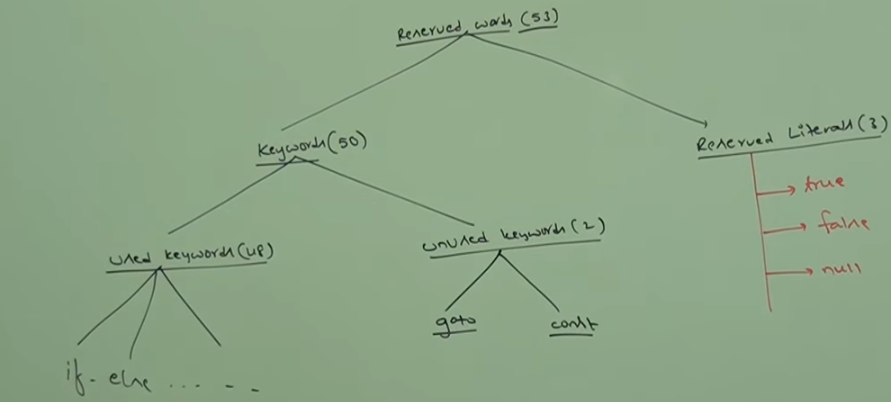
**Enum Keyword** : we can used enum to define a group of named constant.(1)

**Conclusion** :

->All 53 reserved words in java contains only lowercase alphabet symbol.

->in java we have only new keyword and there is no delete keyword because destruction of useless object is the responsibility of garbage collector.

->strictfp(1.2),assert(1.4),enum(1.5) are new keywords in java.



3) **Datatypes** :

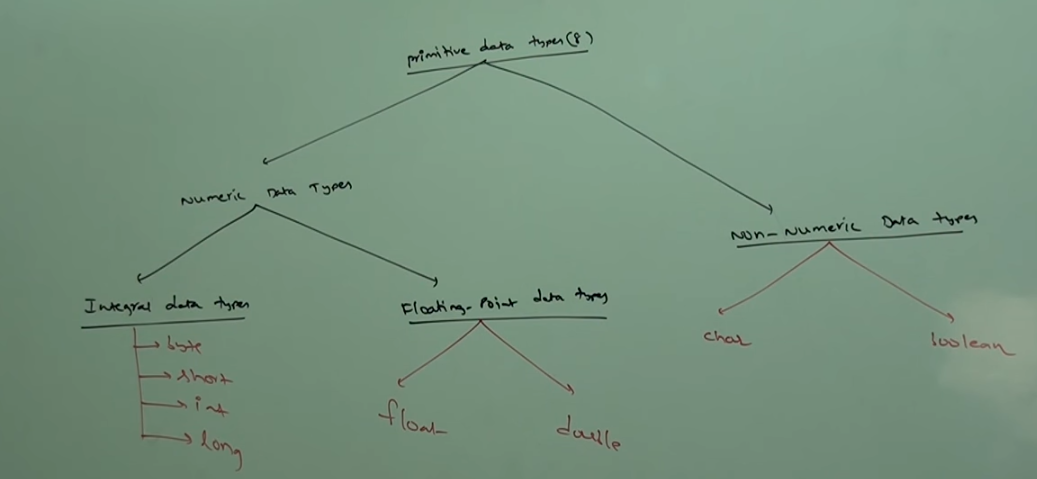
-> In java every variable and every expression has some type.each and every datatype is clearly defined.every assignment should be checked by compiler for type compatibility.

->because of above reason we can conclude java language is strongly typed programming language.

->java is not considered as pure object oriented programming language because several oops features are not satisfied by java(like operator overloading, multiple inheritance etc).moreover we are depending on primitive datatypes which are non objects.

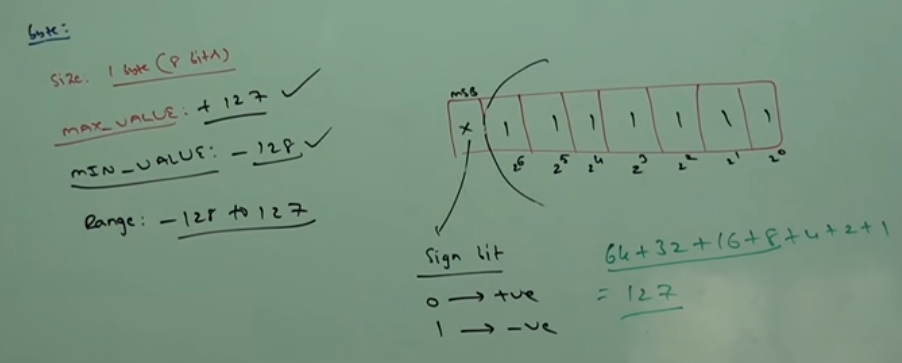
->primitives datatypes(8)

->except Boolean and char remaining datatype are considered as signed datatypes. Because we can represent both positive and negative numbers.



->**Numeric Datatype**

->**byte** : size 1 byte(8 bits).range -128 to 127



->byte is the best choice if we want to handle data in terms of streams either from the file or from network(file supported form or network supported form is byte).

->**short** : size 2 bytes(16 bits).range -2^15 to 2^15-1(-32768 to 32767)

->best suitable for 16 bit processors like 8085 but these processors are completely outdated and hence corresponding short datatype is also outdated datatype.

->most rarely used datatype in java.

->**int :** size 4 bytes(32 bits).-2^31 to 2^31-1(-2147483648 to 2147483647).

->most commonly used datatype in java is int.

->**long :** sometimes int may not enough to hold big values then we should go for long type.eg the amount of distance travelled by light in thousand days.to hold this value int may not enough then we should go for long datatype. long l=126000\*60\*60\*24\*1000 miles;

->eg2 : the no chars in a big file may excd int range hence the return type of length() method is long but not int. long l=f.length();

->size 8 bytes(64 bits).range -2^63 to 2^63-1

->note : all the above datatypes(byte,short,int,long) maint for representing integral values.if we want to represent floating point values then we should go for floating point datatypes.

->**Floating point datatype**

**->float :** if we want 5 to 6 accuracy then go for float. single precision.

->size 4bytes.range -1.7\*10^38 to 1.7\*10^38.

->**double :** if we want 14 to 15 accuracy then go for double. double precision.

->size 8 bytes.range -3.4\*10^308 to 3.4\*10^308.

->**Non Numeric Datatype**

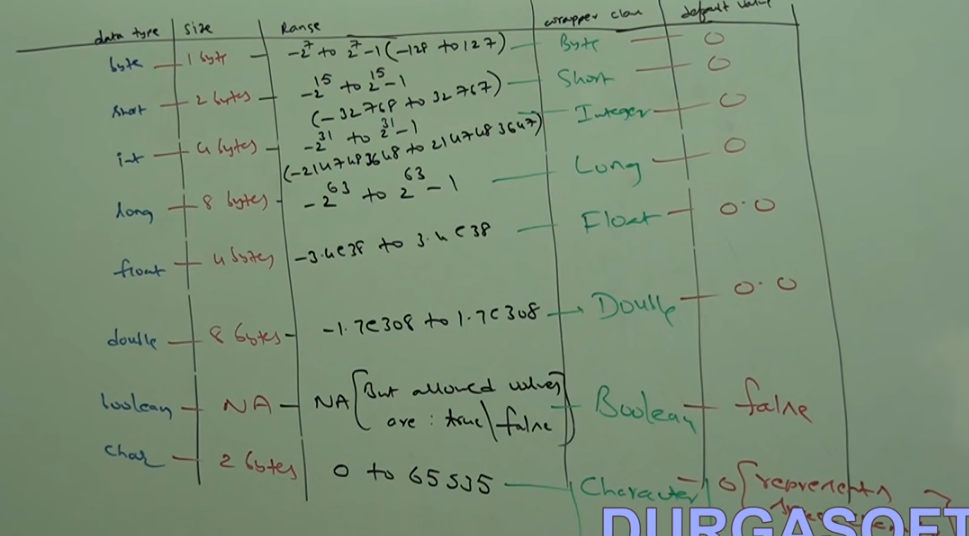
->**Boolean** : range,size is not applicable that is VM dependent. Value wil be true or false.

->char : size 2bytes.reange 0-65535

->old language(c/c++) ASCII code based and no of allowed different asci code characters are <=256. To represent these 256 characters 8bits are enough hence the size of char in old languages is 1 byte.

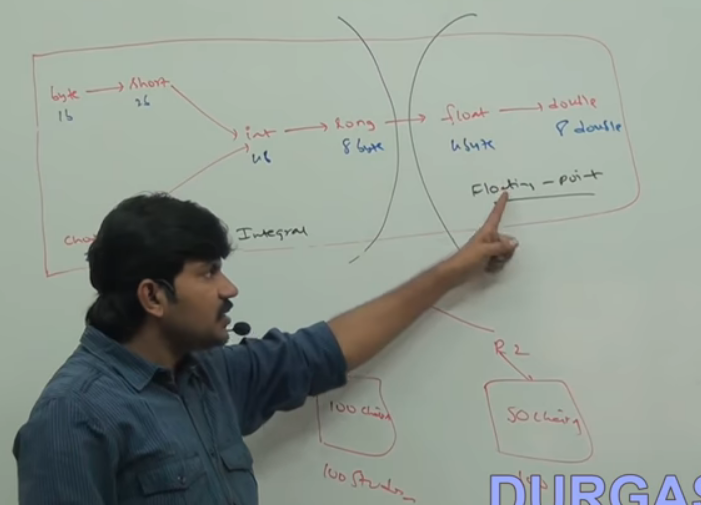
-> but java is Unicode based and the no if different Unicode characters are >256 and <=65536.to represent these many characters 8 bits may not enough hence compulsory we should go for 16 bits hence the size of char in java is 2 bytes.

->Note : null is the default value for object reference and we can’t allowed for primitive .if we are trying to use for primitive then we will get compile time error.



4)**Literals**->A constant value which can be assigned to the variable is called literals. e.g-

|  |  |  |
| --- | --- | --- |
| Int(keyword)  Class(keyword) | I(identifier)  Classname(identifier) | =10(literal) |

****

**Types of Variables**

There are two types of variables.

1)Primitive variables

Can be used to represent primitive values.

2) Reference Variables

**Unary Operator**

1)any opearator that works on a single operand is called unary operator.e.g-++,--

2)any unary operator works only on single operand.

3)the unary operator can be classified into post & pre.

4)in case of post operator the operator is specified after variable name & viceversa in case of pre increment.

5)in case of post or pre increment,the current value of variable gets incremented by ‘1’.

**Local & Global Variable**

|  |  |
| --- | --- |
| **Global Variable** | **Local Variable** |
| 1)initialized outside of the main method within the class body.  7)if we declare global variable but not initialized then compiler initialize default value( 0) to global variables.  4) it can be static or non-static.  2)accessed by any method in a program.  5)accessed through class name or directly.  6) stored in heap memory.  7)initialized in the beginning of the program & is deleted when the program shuts down.  8)peramanent use.  9)if both global & local variables are available in the class then **this** operator can use for global variable.  10)for JVM ,always global variable is the second priority.  11)we can assign a value to a global variable at the declaration time.int i=100;//correct  Int i;  I=20;//not correct  12)can use access specifier. | 1)initialized inside the method.  7)but local variable has to be declare & initialized in program.compiler not initialize any default value for local variable.  4) it can’t be static or non-static.  2)accessed within the method where it is initialized.  5)accessed directly within the method.  6) stored in stack memory.  7)initialized within the method & deleted when that method terminates.  8)temporary use.  9)but for local we can use directly variable name.  10)1st priority.  11)we can declare & initialize in diff line.  12)can’t use access specifier. |

**Static Member & Non-Static Member**

|  |  |
| --- | --- |
| **Static** | **Non-Static** |
| 1)create only one copy in a program.  2)accessed directly by both static & non-static context.  3)accessed by classname.  e.g. classname.staticmember  4)called class member.  5)class loader loads all static members of program.  6)at loading time,its loaded in heap memory i.e. called static pool.  7)static member of a superclass can’t be inherited by subclass.  8)starts from static keyword.  9)Inheritance& overriding doesn’t occour. | 1)create multiple copy by creating the object.  2)accessed directly by non-static context.  3)accessed by static context through reference variable.e.g.referencevariable.nonstaticmember.  4)called instance member or object member.  5)its load after object creation.  6)its load in object.  7)but non-static member of a super class is inherited by subclass.  8)starts without any keyword.  9)occour. |

Refer Demo3.java

**Method**

1)method is a block that starts from signature & contains set of instruction.

2)its same as function in ‘C’.

3)syntax of method->method signature.

4)the starting letter in the method name is lowercase.

5)if more than one word then from second words it must be uppercase.e.g.-simpleCompoundInterest

6)argument declared becomes local variable within the method.

7)return type & return value must match otherwise compiler throws an error.

8)arguments are not mandatory in methods.

9)main is a method in class which is mandatory for execution.

10)after return statement there should not be any statement.otherwise compiler throws error as unreachable statement.

11)void is a keyword in java where the method doesn’t return any value but performs the task given.

12)return statement is compulsory for any method but if void method is used the compiler itself add return statement by default.

13)the return value of a method can be stored in a variable,the variable should be declared with a type of method return type.

14)inside a method the return statement can have both post increment & pre increment.

15)in case of post increment operation the current value of the variable will be printed.the incremented value can’t be accessed since the control is returned back to the called place.

16)any statement in a program which java can’t execute in any condition is known as unreachable code.

|  |
| --- |
| Modifier returntype methodname(type arg0,type arg1)  { /\*method body  //write code to do a task  Return;//last statement of method body. \*/  } |
| Static int addNumber(int a,int b)  {  Int sum;  Sum=a+b;  Return sum  }  //function call or invoke  Int res=addNumber(10,20); |

**Class**

1)A class defn block consists of member,the member can be variables,methods & blocks.it can’t be declared as the static.

2)class can be either public or abstract or final.it will always non-static.

->starting letter of class name is upper case.it is a convention used in industry.

->class name must be same as name of file to be saved.

2)two types of members-**static**-accessed through class name(class name.membername) or directly

**Non static-accessed**throughreferencevariable(referencevariable.membername)

3)**variable**-**primitive variable**(int I,)

->primitive data type-byte,short,int, double,float,Boolean,char

->syntax of declaration-datatype variablename.e.g-int I;

->syntax of initialization-variable=value.e.g.-i=123;d=23.4;

**Reference variable**(classname referencevarname=new classname())

->programming language requires datatype to store data,but scripting language don’t requires datatype to store data.

->data can be stored directly.

Refer program Demo5.java

**Blocks**

1)Blocks doesn’t initialize by any signature(method name).its starts from open braces & end with close braces.

**Static Blocks & Non-Static Block**

|  |  |
| --- | --- |
| **Static Blocks** | **Non-Static Blocks** |
| 1)static bock starts from static keyword.  2)its is excuted before main method in sequential order.  3)SIB are used to initialize static variable of the class.  4)remain only one copy in the program.  5)excutes when class gets loaded.  6)It will executes without main method till 1.6. | 1)starts without keyword.  2)it is excuted after creating the object in sequential order.  3)NIB are used to initialize non-static variable of the class.  4)remain multiple copy in the program  5)executes when the class is instantiated. |

**Why JVM needs main method?**

1)when we run a program with the java interpreter,we specify the name of the class that we want to run.

2)the JVM invokes the main method defined within the class because main method controls the flow of the program & allocates whatever resources are needed & runs any other method that provide the functionality for the application.

3)main() method contains String[] args,to reads command line arguments.

>java Demo5 data1 data2 data3

**Multiple Calsses in one Java File**

1)java file can have multiple classe definition block.such cases file name can be any class name.

2)when a java compiler compiles a java source file which consists of multiple classes then compiler generates .class file for each class definition block.

3)JVM can run one class file at a time.

Source file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class A  { --- }  classB  {-----  } | Java compiler | Byte code  A.class  Byte code  B.class | JVM | CPU |

A.java or B.java

Refer Demo21.java

**Constructor& Constructor Overloading**

1)it is used for

-> creating the object of the class.

-> create an instance of class with the diff values.

its name as same as class name.e.g.-E(){-----}

2)when we are creating the object then we are using new operator.its provides the address of object in heap memory and send to constructor.then constructor creates the object on that particular address and store all non-static members of the class.

3)this should not use any static keyword.

4)we can’t declare any return type in constructor and also return keyword won’t use in the body.

5)it can be private,public & protected.

6)when we are invoking the constructor then constructor creates an object of the class loads non static members of the class in to the object and executes non-static block first of that class then executes constructors.(only time of object creation, execute non-static block first not initializing of the constructor.)

9)we can invoke a constructor of a class multiple times. Everytimes when we invoked an object of the class will be created.

10)the JVM requires a constructor in order to construct the object of the class.if class don’t have a constructor in the program then compiler provides a empty constructor i.e. called **default constructor**.it will not have any argument.

11)the default constructor automatically initializes all instance variable to zero(0) and value of instance variable always will be same.

12)ifwe want to initialize own value of instance variable then we create an arg constructor.

13)if we are using multiple constructor with diff signature in the class that is called **constructoroverloading**. Signature should change either in terms of argument type or in terms of argument no.

14)if developer creates any constructor in the class then compiler doesn’t provide any default constructor.

15)constructor is overloaded but not override.

**Difference between Constructor & Methods**

|  |  |
| --- | --- |
| **Constructor** | **Method** |
| 1)its name as same as class name.  2)it can’t contain static keyword ,abstract keyword & returntype with signature.  3)used to create the object & initialize the value of variable.  4)this is not static or non static member of the class.  5)we can use multiple constructor with diff signature i.e. called constructor overloading.  6)compiler provides default constructor.  7)we can’t override the constructor.  8)this() statement can use in constructor body to invokes the constructor of same class.  9)can’t inherits constructor of superclass in subclass. | 1)its name has given by users.  2)it contains access specifier,static,abstract & return type with signature.  3)used to perform different task.  4)this is static or non-static member of the class.  5)we can use multiple methods with diff signature i.e called method overloading.  6)does’t provide default method.  7)we can override the method.  8)this() can’t use in method body.  9)can inherit methods of superclass in subclass. |

**This Keyword**

1)inside argument constructor if we are initializing non static member(global member equal to local member) then NSM should be refer **“this”** keyword. this keyword indicates current class instance.

2)**this keyword** is helpful where to differentiate the global member and local member inside the context.

3)**this() starement** is used to invoke the constructor of the same class.that should be invoked in first line of the constructor body.

4)recursive call of constructor is not allowed in constructor.

Refer Demo27.java,Demo31.java,Demo33.java

**Method Overloading**

1)if we are using multiple method with different signature in the class i.e. called method overloading.signature should change with either in terms of argument type or no of arguments.

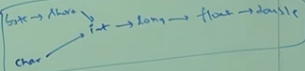
2)where developing a java class of same operation has to be perform with different data or arguments then we have to prefer method overloading.

3)the method overloading can be done on a static method or on a non-static method of a class.

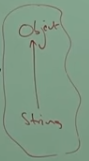
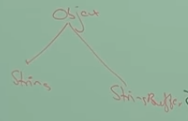
4)anon-static method of a superclass can be overloaded in the subclass.

5)if superclass contains static method then such method can’t be overloaded in the subclass because the static member of a superclass will not to be inherited to subclass.

6)In overloading, automatic promotion is applied by compiler.

****

7)child class have more priority than parent class.

** **

8)general method came in 1.0 version but var-arg method(call by any no of arguments) in 1.5version so general method will get chance . if general method will not present then var-arg method will get chance. This is exactly same as default in switch.

9)In overloading method resolution always takes care by compiler based on reference type. Runtime object won’t play any role.

**Method Overriding**

1)inheriting a method of a superclass and changing the method behavior in subclass is known as method overriding.

2)method overriding should obey the following rules.

->inheritance should happen.

->signature of the method should same as superclass.

->the subclass should provide different implementation to the method.

3)the static method of a superclass can’t be overrided in the subclass because it can’t be inherited in subclass. Also we can’t override private method.

4)when developing a class ,if class acquire the properties of a superclass and that properties has to be modified in that class then method overriding should be used.the original method should not be changed.

5)the method get overrided is known as overridden method and which method overrides the superclass method is known as overrided method.

6)a method of a superclass can be overrided as well as overloaded in the subclass.

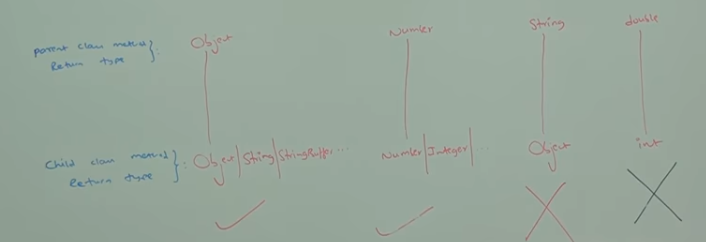
-> we can override no abstract as abstract. Advantage is subclass cann’t use the parent implementation.

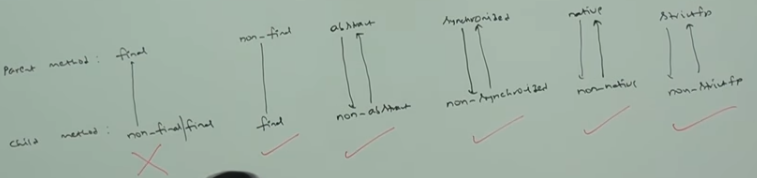
->in overriding we can’t decrease the scope of access modifier.

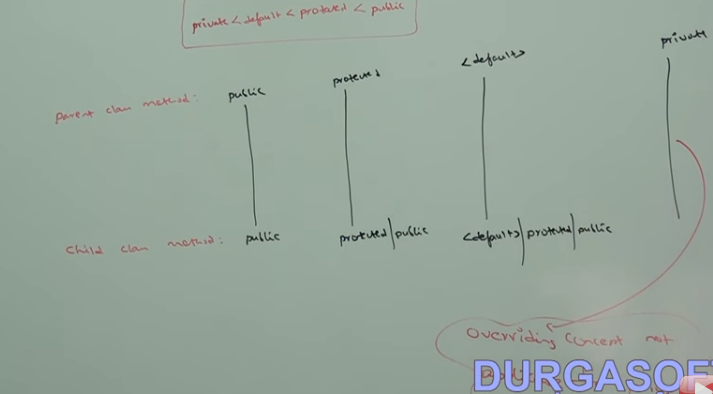
7)method overriding occurs only when the names and the type signature of both the methods are same.if they are not ,then the two methods are overloaded.

8)when we want to access method of superclass then we have to use “**super**” keyword.

9)co-varient return type is allowed after from jdk1.5.//>javac –source 1.4 demo.java







**Covariant Return Type**

1) The covariant return type specifies that the return type may vary in the same direction as the subclass.

2) Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type.

3) Let's take a simple example: Note: If you are beginner to java, skip this topic and return to it after OOPs concepts. Simple example of Covariant Return Type

class A{

A get(){return this;} }

class B1 extends A{

B1 get(){return this;}

void message(){System.out.println("welcome to covariant return type");}

public static void main(String args[]){

new B1().get().message(); }

}

**Difference betn Method Overloading & Overriding**

|  |  |
| --- | --- |
| **Methods Overloading** | **Method Overriding** |
| 1)when we use multiple methods in same class of diff signature either in terms of argumenttype & no of argument i.e. called method overloading.  2)its possible in either same class or subclass.  3)signature should be different.  4)static method of superclass is overloaded in same class but not in subclass.  5)for method overloading,inheritance is not mandatory.  6)interface doesn’t require method overloading.  7)we can use static methods or non-static method for overloading. | 1)when we inherits any method from superclass and change the behavior of method in subclass is known as method overriding.  2)its possible in subclass only.  3)signature should be same(return type also).  4)static member of superclass can’t override.  5)inheritance mandatory.  6)requires method overriding.  7)but only non-static methods participate in methods overriding. |

**Abstract Classes & Concrete Classes**

1)method is developed with the body is called concrete method or complete method. if a class contains complete methods is called concrete class.

2)method is developed without body(only method signature) is called abstract method or incomplete method.if a class contains abstract method then that class is called abstract class and that class should also declared abstract.

3)if a class are developed as an abstract class then it is not mandatory to write an abstract method.

4)an abstract class can’t be instantiated or an object of abstract class can’t be created.

5)if an abstract class contains complete non-static method then such method can’t be accessed because an object of abstract class can’t be created.

6)static member of an abstract class can be access because the static members are access with class reference.

7)if subclass inherits the superclass and superclass is abstract then subclass also be abstract then subclass is declared as abstract class.

8)if a subclass inherits the abstract method from superclass(abstract class) then by default subclass also become abstract class.if we provide body for abstract method in the subclass then subclass become concrete class.

9)then we can create the object of subclass to access all the members of subclass.

10)if we have to utilize concrete nonstatic method of an abstract class then we have to inherit superclass and give body to abstract method is known as overrided.

11)abstract static is an illegal combination in java class because whenever a method is declared as abstract such method should get the body in subclass or should be overrided in the subclass.

Abstract static void test();

12)static method can’t be overrided because it can’t be inherited to the subclass.

**Inheritance & Multi level Inheritance**

1)inheritance is the process in which one class inherits member of the other class. which class inherits i.e. called derived class & which class is inherited by other class i.e. called super or base class.

acquiring properties & behavior of a super class is known as inheritance. parent class is superclass & child class is subclass.

2)only the non-static member of superclass are inherited to the subclass.

the static member of the superclass is not to be inherited of subclass object because static member are not part of the objects and they are loaded in the static pool.

3)whenever an object of subclass is created then both the subclass member and superclass members are loaded into the object.

4)subclass object can access members of the superclass but superclass object can’t access members of the subclass.

5)when a subclass object is created then subclass constructor calls superclass constructor,superclasss constructor again calls its superclass constructor.this is known as constructor chain.inheritance can be achieved only through the chain of constructor.

6)the subclass constructor invokes superclass constructor implicitly by calling the default constructor of the superclass.a superclass constructor can be invoked by statement called **super();** statement.

7)if we have to invoke other constructor of a superclass(overloaded constructor) then explicitly super() statement should be written inside the constructor of the class.super() statement can take any arguments & always super() statement should be written in the first line of the constructor.

8)java doesn’t support the inheritance of multiple superclasses into a single subclass.but in C++,we can inherit multiple base class.

9)inheriting the feature more than one class at a time is known as **multiple inheritance**(Diamond problem).

10)in java multiple inheritance is not possible through class definition block because every class should inherit from the object class.multiple invocation of object class from same source is not allowed.

11)inside constructor of a subclass ,we can’t write two super() statement .

12)every class in java should extends from object class.Object class is the supermost class of all the classes in java.

13)Cat Is-A animal.Is-A means inheritance.cat is a subclass of animal.cat Has-A ear.Has-A means member.ear is a member of cat.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Animal |  |  |  |
|  | Mammel | Reptile | Bird |  |  |
| Cat | Dog | Snake | Lizard | Crow | Pigeon |

Cat Is-A mammal.

Cat Is-A animal.

Cat Is-A cat.

Cat Is-A object.

Is-A called inheritance.if Cat Has-A ear.this is called member.

Cat c=new Cat();

Animal a=c;

Mammel m=c;

Object 0=c;

Now cat class is called to polymorphic since this has multiple inheritance.

**Interface**

1)java supports four types of definition block.

->class type

->interface type

->enumeration type/enum type

->annotation type

2)an interface type definition block should start with the keyword “interface”.every interface should have a name.

3)the java compiler compiles interface and give bytecode of each interface.in general compiler gives bytecode for each type of definition block.

4)a jave file can have both class definition & interface definition block.

5)inside interface body we can develop only abstract method & start without abstract keyword.we can’t develop a concrete method inside interface.

6)every method are abstract in interface.by default the interface itself is an abstract.hence no need to declare the member or method using abstract keyword.

7)by default all the methods in interface are public.when we ll override the methods in subclass then public keyword should be there.

8)we can’t develop static methods in interface.

9)inside interface,variable should declare & initialize in the same line.by default all the variables are constant in interface& static.we can access anywhere through class name.

10)advantages->gives 100% abstraction & we can do multiple inheritance using interface.

11)final& abstract is illegal & static & abstract is illegal.

Refer Demo38.java

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1)if any class contains abstract methods or incomplete methods(without body) then that class is called abstract class.  2)we can use concrete method inside abstract class.  3)abstract keyword is used for abstract class.  4)not 100% abstraction.  5)multiple inheritance is not possible using abstract class.  6)in abstract class variable won’t be constant.  7)we can use static method in abstract class.  8)we can use diff. access specifier with method.  9)by default ,all the variable are not constant & static. | 1)in interface ,all the methods are abstract.  2)we can’t use concrete methods inside interface.  3)abstract keyword is not used.  4)100% abstraction.  5)multiple inheritance possible using interface.  6)variable will be always constant.we can use final keyword.  7)we can’t use static method in interface.  8)by default,all the methods are public.  9)by default,all the variables are constant & static. |

**Use of Final Keyword**

1)we can initialized local variable as final but in the same line that’s not mandatory.we can’t reinitialized the final local variable i.e that variable will be constant.

->we can initialized global variable as final but in the same line.that may be static or nonstatic.we can’t reinitialized the final global variable.

->final abstract keyword is illegal combination.

->a static method can’t be final.

->a static variable can be declared as final.

->by default every variables declared in interface are static and final.

final int SIZE=10;

final double PHI=3.14;

We can’t modify its size.variable name must be in uppercase.

2)using final to prevent overriding.

Class A{

final void show()

{System.out.println(“this is a final method”);

}}

Class B extends A{

Void show()//ERROR!cannot override{

System.out.println(“illegal”);

}}

3)Using final to prevent inheritance

Final class A{//…}

//The following class is illegal

Class B extends A//ERROR!cannot subclass A{//..}

**Type Casting & Type conversion**

1)converting one type to another type is known as type casting.the typecasting is classified into two types.

->primitive casting

->derived casting

2)**Primitive casting**converting a primitive datatype to another primitive datatype is known as primitive casting.classified into two types.

->auto widening

->explicit narrowing

1)Auto widening->the type conversion is that which automatically converts by compiler one data type into other.but we can convert a small data type into larger data type is called auto widening.e.g. we can store a int value into float cooz float is larger data type that int.

2)this is called promotion.e.g.

int a=6;

float f=a;

Byte ->short orchar -> int ->long -> float -> double

2)Explicit narrowing->when we are converting the large data type into small data type is called explicit narrowing or type conversion.that is called demotion.the narrowing should be explicitly specified in the code otherwise compiler throws precision loss error.e.g. (target\_type)value.

3)whenever narrowing is happening then there will be a loss of precision data.

double d=12.3;

int a=(int)d;

4)primitive casting can be done where

->initializing the variable.

->passing arguments through method.

->via returning the value of the method.

Char->

Byte ->Short->int->long->float->double

5)**Derived casting**-converting one object type to another object type is known as derived casting.

->an object can be casted only if the classes are related(inheritance).

->unrelated objects can not be converted.

6)two types ->auto upcasting

->explicit downcasting

7)**Auto upcasting->**converting a subclass object to superclass type is known as upcasting.if subclass obj is to assign to superclass type reference variable,compiler automatically converts the subclass obj to supertype hence it is als known as auto upcast.

8)whenever a subclass is upcasted then subclass member can’t be accessed.

9)**Explicit Downcasting->**converting superclass obj to any other subclass type is known as downcasting.

10)a new superclass object can’t be downcasted.the java compiler will not throw any error because syntax is correct.at run time JVM fails to convert because superclass obj doesn’t have the member of subclass.that time JVM throws an exception ClassCastException.

11)JVM can downcast only an upcasted object.

12)type promotion rule

->Byte & Short is to be promoted to int.

->if one operand is long ,whole expression is promoted to long.

->if one operand is float ,whole expression is promoted to float.

->if one operand is double ,whole expression is promoted to double.

**Polymorphism**

1)A method behaving differently at different stages is known as polymorphism.

In other words one object taking diff. form is known as polymorphism.

2)classified into two types.

->compile time polymorphism

->run time polymorphism

3)**compile time polymorphism ->**in compile time polymorphism,the compiler identifies the body of the method based on the signature,the identification is done at compilation time.hence it is called as compile time polymorphism & static polymorphism.e.g-method overloading & constructor overloading.

->static methods of the class can be participate in compile time polymorphism.

5)**run time polymorphism->**in run time polymorphism, method body are identified based on the object at run time ,hence it is known as run time polymorphism& dynamic polymorphism.e.g.method overriding.etc.

->static method can’t involved in run time polymorphism because static method can’t not be overrided.

->following types are mandatory for run time polymorphism.

#inheritance

#overriding

#upcasting

**Encapsulation**

1)+ encapsulation.e.g.class,interface,TVetc.

**Packages & Access levels**

1)a package is a grouping of related classes providing access protection & name space i.e classes can be defined inside a package that are not accessible by code outside that package.

2)related class are grouped into packages for the following reason:-

->to make classes & interfaces easier to find & use.

->to avoid naming conflicts.

->to control access

3)packages are stored on the disk in their own directories(folder) & subdirectories(subfolder).

4)packages names are written in all lowercase to avoid conflict with the names of classes.e.g.-java.lang,package pack1.graphics.Line

5)two types:-

->java API packages-built-in java class libraries.java API provides a large number of classes that are grouped into diff. packages according to their functionality.

1)java.lang-includes classes for –primitive types,strings,math functios,threads & exceptions.

2)java.util-includes classes for vectors,hash tables,random numbers,date etc.

3)java.io-include classes for the input & output of data.

4)java.awt-includes classes for GUI like windows,buttons,list menus etc.

5)java.net-includes classes for networking,communicating with local computer as well as with internet servers.

6)java.applet-classes for creating & implementing applets.

->user defined packages-we can also build a library that groups related classes. e.g-if we have class name called “Vector”.its name would crash with the Vector class from JDK.

6)in general,a java source file can contain any(or all)of the following internal parts in the following order.

->a single package statement(optional)

->any number of import statements(optional)

->a single public class declaration(mandatory)

->any number of classes private to the package(optional)

7)Importing packages-import p1.\*;

Import p1.Protection;

Import java.util.\*;

Class MyVector extends java.util.Vector

7)**class member acess**-modifier

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Private | Default | protected | Public |
| Same class | Yes | Yes | Yes | Yes |
| Same package subclass | No | Yes | Yes | Yes |
| Same package non-subclass | No | Yes | Yes | Yes |
| Diff. package subclass | No | No | Yes | Yes |
| Diff. package non-subclass | No | No | No | Yes |

1)**Private->**can’t access outside of the class.

->to access private member then class should provide getter & setter method.

->constructor can be declared as private then we can access within the class.we can’t create the obj. outside of the class.

->we can’t create the private class.

2)**Default->accessed** within the package.we can’t use access specifier.

->a class can beonly defaultor public then constructor is also default& public only..

->if a java file contain any public class(any type of defn block) then we have to save as public class name.only one class name should be public within the class.multiple public type is not allowed.

->if a file contain multiple default type defn block then file name can be any of the type defn.

3)**Protected->**it can accessed within the package & diff package subclass.

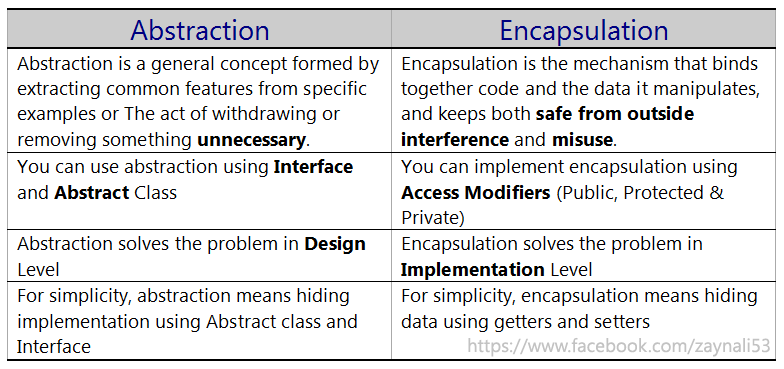
4)**Public->**can access everywhere.

5) **native** : Use to call library of other language from java.

6) **volatile** : Used to update any values of the variable into main memory so that all the threads will have the updated values.

**Inheritance ,Encapsulation,polymorphism& abstraction**

|  |  |  |  |
| --- | --- | --- | --- |
| **Inheritance** | **Polymorphism** | **Encapsulation** | **Abstraction** |
| 1)process by which objects of one class acquire features of object of another class.  Vehiclecarracing car | 1)ability to take more than one form.e.g. word “shape” has many forms as triangle,rectangle etc. | 1) Encapsulation is process of **wrapping code** and data together into a single unit.eg-TV  We can create fully encapsulated class by making the entire **data member will be private** and create getter, setter method to access that data member.  POJO class is the best example of fully encapsulated class.  Encapsulation is also known as "**Data hiding**" because they are protecting data which is prone to change.  Encapsulation in Java is achieved using access modifier **private, protected and public**.      - Singleton pattern in Java makes good use of Encapsulation. | 1) Abstraction is process of **hiding the implementation details**and showing only the functionality.eg-remote  Menhuman<--Women |



**Coupling**

1)By definition coupling is the degree to which one class has knowledge of another or in other words one class has a dependency upon another.

2)Tight coupling occurs when a dependent concrete class contains a pointer to another concrete class that provides the required behavior and should be avoided.

3)The problem here is that any changes to one class could impact the other and the person making the changes may be completely unaware of this and thus unknowingly break the class.

4) So how do we avoid this scenario? We design by contract by using an interface to specify an API for other classes to use as discussed in

the OO Concepts - Interfaces lesson or by using encapsulation as discussed in the OO Concepts - Encapsulation. The following code is an example of tight coupling and should be avoided:

5)Being low coupling would mean that changing something major in one class should not affect the other. High coupling would make your code difficult to make changes as well as to maintain it, as classes arecoupled closely together, making a change could mean an entire system revamp.

6)All good software design will go for high cohesion and low coupling.

/\*

Tight coupling example

\*/

class A {

int i;

B b = new B();

i = b.value; // No encapsulation of this variable in class B!

}

class B {

public int value; // Should be private and be accessed through public getters and setters

}

**Cohesion**

1)Cohesion is the degree to which components of a class belong together to fit a particular role. What we want to avoid is low cohesion where a class incorporates several different aspects. A class that tries to do many things comes with higher maintenance and lower reusability.

/\*

Low cohesion example

\*/

class AllInStaff {

void getStaffSalary();

void getStaffDetails();

void getStaffSalesReport();

}

/\*

High cohesion example

\*/

class Accounts {

void getStaffSalary();

...

}

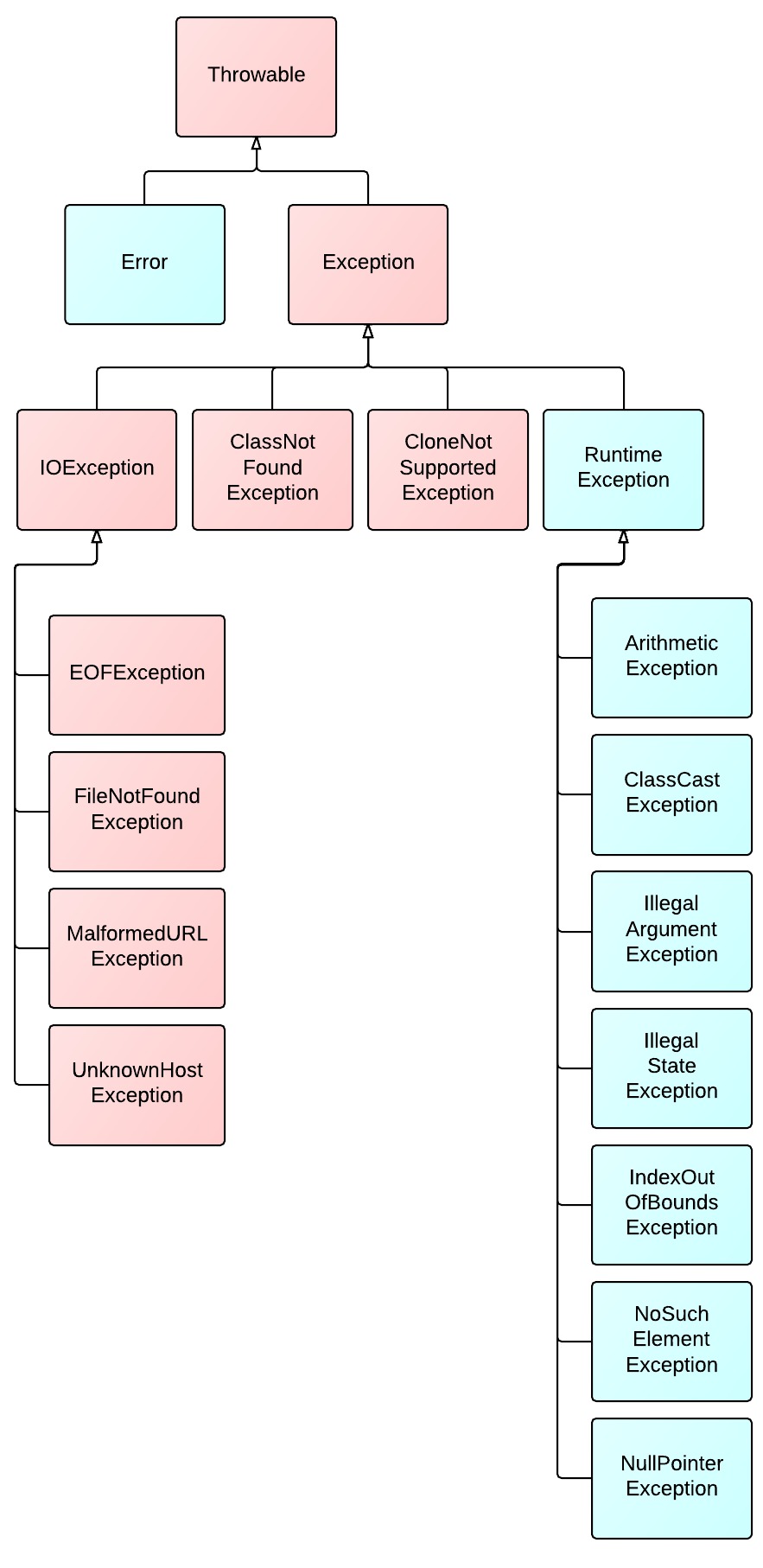
class Personnel {

void getStaffDetails();

...

}

**Exception Handling**



1)when we are using abnormal condition or abnormal statement then exception will occur then java does following steps:-

->crates an object of exception class(throwable type).

->add relevant message into the object.

->throws the object using **throw keyword.**

2)this process known as raise of exception.

3)whenever an exception raises the object should be handled in the program otherwise the java terminates the execution immediately.in such case the remaining portion of the program will not be executed.

4)in order to handle the exception,java provide try{}catch(){} block.the try{}block should have a statement which raises the exception.the exception raises handled by the catch block.the catch block should have an argument type **throwable.**

5)if an exception occur inside the try{} block,that exception will be handled by catch(){} block.the remaining portion of the try{} block will not be executed.

6)if there is no exception occur in try{} block then catch(){} block will never be executed.

7)in a program,a try{} block can have multiple catch(){}block.in such case only one catch block will be executed.if one of the catch(){} block arguments are declared as throwable class then other catch block become a unreachable blocks hence compiler throws an error.

-> Also we can catch error using throwable type in catch block.

8)nested try-catch block is allowed in program.if inner try exception is not catched by inner catch block then such exception **propagates** to the outer catch block. if outer catch block is handling the exception then outer catch block will be executed and continue the execution from outer catch block.

9)the exception are classified into two types.

->**Checked Exception**-checks the exceptions at compile time.handled by try-catch block or throws declaration.throws declaration is the keyword which is used with method signature and constructor signature except **run()** method.it is used after the signature with respective exception class name.

->**Unchecked Exception**-checks at run time.handled by try-catch block only.

10) If I am providing class name dynamically and runtime if .class is not present then we will get classnotfoundexception(checked).

If I am using hard coded class name and runtime if .class is not present then will get noclassdeffounderror(unchecked).

Test test=new Test();//noclassdeffounderror//This class was available at compile time but not available at run time

Class.forName(“oracle.jdbc.driver.OracleDriver”);//classnotfoundexception//jar file was not added

>java Test Student

10)**Stack unwinding-**

1)forcely removing incompleted methods or statements from the stack is known as **Stack Unwinding**.

2)java performs unwinding operation when exceptions are not properly handled.

3)if one of the stack member generates an exception and that exceptios is not handled by that member then exception propagates until it finds the corresponding handler in the stack.if no handler is found then finally the exception reaches to java and java terminates the execution.before terminating it unwinds the stack.it always good practice to handle the exception where it occurred.

-

Try

{

//block of code to monitor for errors

}

Catch(ExceptionType1 ex0b){

//Exception handler for exception type1}

10)**Finally Block-**

1)finally block is always associated with try-catch block.this block gets executed irrespective of the exception occurs or not occurs.

2)when developing any application if any mandatory code has to be executed then such code or statement are written inside finally block.

3)a try block should have either catch block or finally block.

4)when we use return keyword in either try block or catch block then finally block will get executed,all the statement will be unreachable but if we use System.exit(0) then finally block will not get executed.

e.g.when we are writing the java program to connect database to insert data.we have to finish following steps.

->connect to the database

->Prepare set of data

->Create SQL query

->run SQL query using java code

->close connection

If we are getting exeception before closing the connection then java terminates.but we have close the connection thatsy we have to keep close connection inside the finally block.

**Eclipse**

1)eclipse contain auto compile properties.

2)own output screen is called console.

3)**Meaning of System.out.println()-**System is the class that is available in java.lang package.out is the static reference variable point to out device.println() is the method of class System i.e accessing by reference variable out.

->we don’t need to import java.lang package in program.apart from that we have to import all packages belongs to the class.

4)**Scanner-**is the class to scan input device.it can’t contain static method.it doesn’t contain default constructor.it is available in java.util package.

5)**Next()-**next() method is available in Scanner class.used to read the data from input device until the press enter button.

**Object Classes**

1)Object is a the supermost class defined by java.all other classes are subclasses of object.object class is available in java.lang package.

2)every class inherits the members of object class.

3)Object is a superclass of all other classes.this means that a reference to a variable of type Object can refer to an object of any other class.

4)Object defines the following methods,which means that they are available in every object.

5)the method getClass(),wait(),notify,notifyAll() are declared as final.we can’t override.

6)the equals() method compares the contents of two objects.it returns true if the objects are equivalent and false otherwise.

7)thetoString() method returns a string that contains contains a description of the object on which it is called.also,this method is automatically called when an object is output using println().

8)finalize() & clone()will be protected methods.rest of the methods are public.

**Method of Object Class**

1)**toString()**

->returns the string representation of the object(address of object in string format with fully qualified class name @hexadecimal value).that is public method of object class.

->whenever a reference variable is printed,on the reference variable toString() method of object class is implicitly invoked hence output is string representation of the object.

->if an class,toString() method is overrided then whenever toString() method is invoked then overrided method will be executed.

->we can call toString() method implicit & explicit in both way.

->used to print all the members of the object.

2)**hashCode()**

**->**returns the hashcode value of object(i.e address of object in integer format).that is public method of object class.

->if hashCode() method is overrided then overrided method will be executed.

->can’t call implicitly.

3)**equals(Object obj)**

->determines whether one object is equal to another(address).

->public method of Object Class.

->if we don’t override then always its compare to address of the object.

->if we want to compare the value of the object then we have to override.

3)**clone**

->Process of creating exact duplicate copy is called cloning.

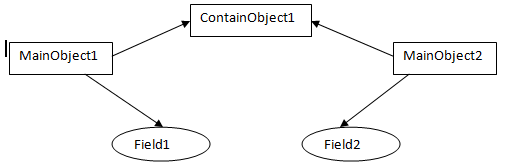
-> Used to maintain backup purpose and preserve the state of an object.

**Types**

**What is Shallow Copy?**

* For shallow copy just we have to use clone() method of object class and implements Cloneable interface.

Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

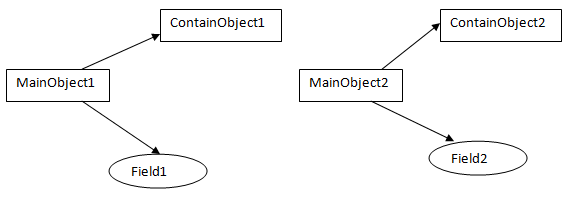


In this figure, the MainObject1 have fields "field1" of int type, and "ContainObject1" of ContainObject type. When you do a shallow copy of MainObject1, MainObject2 is created with "field3" containing the copied value of "field1" and still pointing to ContainObject1 itself. Observe here and you will find that since field1 is of primitive type, the values of it are copied to field3 but ContainedObject1 is an object, so MainObject2 is still pointing to ContainObject1. So any changes made to ContainObject1 in MainObject1 will reflect in MainObject2.

Now if this is shallow copy, lets see what's deep copy?

**What is Deep Copy?**

A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.



In this figure, the MainObject1 have fields "field1" of int type, and "ContainObject1" of ContainObject type. When you do a deep copy of MainObject1, MainObject2 is created with "field3" containing the copied value of "field1" and "ContainObject2" containing the copied value of ContainObject1.So any changes made to ContainObject1 in MainObject1 will not reflect in MainObject2.

* For deep copy, We have to override clone method in our class and create fresh object and return that object and implement clonable interface.
* If object contains only primitive variable then go for shallow clone otherwise deep clone.

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| Public String toString()  Public int hashCode()  Public boolean equals(Object object)  Public void wait()  Public void wait(long milliseconds)  Public void wait(long milliseconds,int nanoseconds)  Public void notify()  Public void notify All()  Public class getclass()  Protected native Object clone() throws CNSE  Protected void finalize() | Returns a string representation of the object in the format fully classified class name @hexadecimal no..  Returns the integer value for the object.  Determines whether one object is equal to another.  Waits on another thread of execution.  Resume execution of a thread waiting on the invoking object.  Resume execution of all thread waiting on the invoking object.  Returns the runtime class of this object.  Creates and returns a shallow copy of this object.  called by the garbage collector on an object when garbage collection determines that there no more references to the object. |

**String, StringBuffer and StringBuilder Concept**

1. **String class is an immutable class because if we want change the value of the object(using concat()) then new object will create but StringBuffer class is mutable because if we want to change the value of the object(using append()) then change will happen in the same object.**
2. **In String class equals() method is overrided for values comparison and == operator will do reference comparison but in StringBuffer class equals() method is not overrided so it will call from object class and will do reference comparison.**
3. **If we are creating object with new operator then it will create two object. One is in heap area and another one is SCP area(future purpose). And without new operator it will create only one object that is in SCP area.**
4. **In heap area duplicate objects are allowed but in SCP area duplicates not allowed.**
5. **If object will create at runtime and content has been changed(Ex. S1.concat(“durga”)) then it will create object in heap area but if content has not been changed(EX.s1.toLowerCase()) then duplicates not allowed in heap or SCP and for constant one object in in SCP area.**
6. **GC is not allowed to access SCP area, Only access heap area so if object doesn’t contain any reference in SCP area then also GC will not delete that object. SCP object will destroy after JVM shutdown.**
7. **There are five types of memory area in java. Stack, Heap, method(contains SCP), native method stack, piece register.**
8. **We can’t change the value of the object that is called Immutable. But if we want change the value of the object then it will create the new object else use current object.**
9. **Final variable and Immutable object is different concept.**
10. **If content is fixed then go for string and If content is not fixed then go for StringBuffer because for every change string will create new object.**
11. **We can use method chain also. here method will execute from left to right.**
12. **Intern() method will check the object in SCP. If available then return the object from string constant pool and if not available then create the object in the SCP. Used to create object of string with new operator in SCP area.**
13. **To create immutable class in java, you have to do following steps.**

* Declare the class as final so it can’t be extended.
* Make all fields private and final so that dirqect access is not allowed and it’s value can be assigned only once .
* Don’t provide setter methods for variables
* Initialize all the fields via a constructor performing deep copy.
* Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

**StringBuffer Constructor**

1. **new StringBuffer();//16,NC=(cc+1)\*2, new StringBuffer("vikash"), new StringBuffer(int initialCapacity)**
2. **In string size,length and capacity all are same hence we have only length() method. But In stringBuffer(default capacity=16,new capacity=(CC+1)\*2) we have two methods length() and capacity().**
3. **Every methods are synchronized so performance is low.**

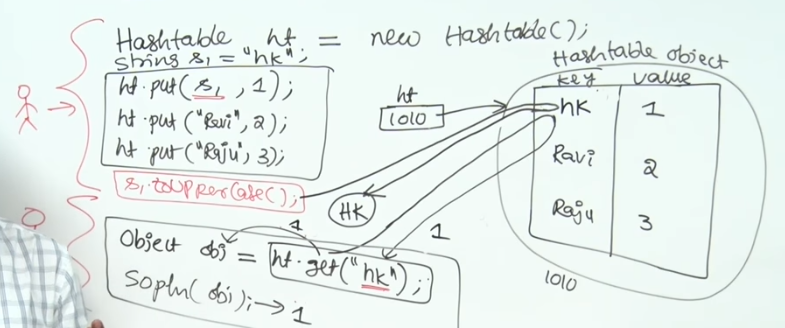
**StringBuilder**

1. **Excatly same as StringBuffer except this is not threadsafe so performance is high.**

----------------------------------------------------------------------------------  
                                  ***String***                ***StringBuffer***        ***StringBuilder***  
----------------------------------------------------------------------------------                   
**Storage Area** | Constant String Pool           Heap                       Heap   
**Modifiable**     |  No (immutable)            Yes( mutable )          Yes( mutable )  
**Thread Safe**   |           Yes                                  Yes                              No  
**Performance** |         Fast                                Very slow                    Fast  
-----------------------------------------------------------------------------------

**Why String class is Immutable?**

1. **Due to store as a key into map object.**

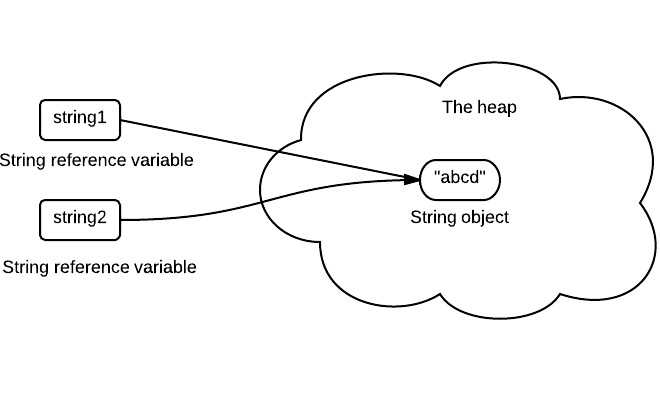
****

**1. Requirement of String Pool**

String pool (String intern pool) is a special storage area in [Method Area](http://www.programcreek.com/2013/04/jvm-run-time-data-areas/). When a string is created and if the string already exists in the pool, the reference of the existing string will be returned, instead of creating a new object and returning its reference.

The following code will create only one string object in the heap.

|  |
| --- |
| String string1 ="abcd";  String string2 ="abcd"; |

Here is how it looks:  
[](http://www.programcreek.com/wp-content/uploads/2013/07/java-string-pool.jpeg)

If a string is not immutable, changing the string with one reference will lead to the wrong value for the other references.

**2. Caching Hashcode**

The hashcode of the string is frequently used in Java. For example, in a HashMap. Being immutable guarantees that hashcode will always the same so that it can be cashed without worrying the changes.That means, there is no need to calculate hashcode every time it is used. This is more efficient.

In String class, it has the following code:

|  |
| --- |
| **Private int** hash;*//this is used to cache hash code.* |

**3. Facilitating the Use of Other Objects**

To make this concrete, consider the following program:

|  |
| --- |
| HashSet<String> set =**new** HashSet<String>();  set.add(**new**String("a"));  set.add(**new**String("b"));  set.add(**new**String("c"));    **for**(String a: set)  a.value="a"; |

In this example, if String is mutable, it's value can be changed which would violate the design of set (set contains unduplicated elements). This example is designed for simplicity sake, in the real String class there is no value field.

**4. Security**

String is widely used as a parameter for many java classes, e.g. network connection, opening files, etc. Were String not immutable, a connection or file would be changed and lead to a serious security threat. The method thought it was connecting to one machine, but was not. Mutable strings could cause a security problem in Reflection too, as the parameters are strings.

Here is a code example:

|  |
| --- |
| **boolean** connect(string s){  **if**(!isSecure(s)){  **thrownew**SecurityException();  }  *//here will cause problem, if s is changed before this by using other references.*  causeProblem(s);  } |

**5. Immutable objects are naturally thread-safe**

Because immutable objects can not be changed, they can be shared among multiple threads freely. This eliminates the requirements of doing synchronization.

In summary, String is designed to be immutable for the sake of efficiency and security. This is also the reason why immutable classes are preferred in general.

**Array**

1)arrays is a class not a datatype.collection of similar datatype.present in java.util package.

->primitive array-collection of similar primitive data type.for empty array, reference variable will print like [I@hexadecimal.default value will be zero.

->derived array-collection of similar object type.here for empty array will print like com.jspider.prac[tise.A@hexa](mailto:tise.A@hexa) decimal.by default the value will be null.

2)which stores a fixed-size sequential collection of elements of same data type.e.g. dataType[] arrayRefVar=new datatype[size].

Int [] a=new int[20];

3)declaration& initialization should be in same line.

4)traditional for loop-for(initializatiom;condition;incement){}

->auto increment for loop-for(type varname:arrayname){}

5)**derived array-**

1)declaration –classname[] arrayname=new classname[size]

2)we can’t create the object of abstract class but can create the reference variable of abstract class.

3)object[4]=12,it convert into an object and store.

**COMPARABLE & COMPARATOR**

1)if we want to sort a derived array of a class based on one value of object then class should be **comparable** type.for that we have to implement **Comparable** interface and override compareTo() method.

-available in jav.lang package.

2)if we want to sort a derived array of a class based on multiple value of objects then we have to create separate class and class should be **comparator** type.for that we have to implement **Comparator** interface and override compare() method.

->available in java.util package.

**Wrapper Class**

1)it is used to convert any data type into an object.primitive data types are not object.they donot belong to any class.they are defined in the language itself.

2)datastructure of JDK accept only object to store.a datatype is tobe converted into an object and then added to a stack or vector etc.for this conversion ,we use wrapper classes.here constructors are used.

3)to convert strings into datatypes (known as parsing operations),here methods of type parseXXX() are used.

4)all the wrapper classes are final class we can’t create the subclass of wrapper class.

5)in every wrapper class toString() method is overrided to return object value.

6)all wrapper classes which represents the numeric datatype inherits from number.

7)the number class is a abstract class.

8)converting a primitive type to an object is known as boxing operation.the boxing operation can be done by wrapper class.

9)from JDK1.5 onwards,boxing operation can be automatically done by compiler.

10)converting an object back to primitive type is known as unboxing operation.the unboxing operation also be done by using wrapper class.

11)java is not follow 100% OOPS concept.because we can not store any data type to object array.first we have to convert into an object then we can store.

12) Java uses a **pool** for Integers in the range from -128 to 127.

Integer a1=127;Integer a2=127;a1==a2(true).

Integer b1=128;Integer b2=128;//it will create the object.b1==b2(false).

13)when you tried to compare 127 with 127 there was only one object  
made and it worked right but when you tried with 128 it came out of  
the range and it created two objects so you can't compare them using  
== operator.

|  |  |  |
| --- | --- | --- |
| Primitive data types | Wrapper Class | Object |
| Byte  Short  Int  Long  Float  Double  Char  Boolean | Byte  Short  Integer  Long  Float  Double  Character  Boolean | Byte object  Short object  Integer object  Long object  Float object  Double object  Character object  Boolean object |

**OBJECT**

**NUMBER abstract class**

**Boolean**

**Character**

**Byte FC**

**Short FC**

**Integer fc**

**Double fc**

**Float fc**

**Long fc**

**Inner Class**

Display a class inside a class is known as inner class.

classifieds in following types.

1)Local Inner Class:-

->a local inner class is non-static inner class defined inside a method body.we can declare only non-static member.this class can be accessed only within the method where it is declared.

2)Static Inner Class:-

->it is defined with static keyword,outside of method block.inside static inner class,we can declare both static & non-static members.

->inside static inner class ,we can access any member of outer class.

->If we have to refer static inner class,it should be referred with outer class reference.

3)Non-Static Inner Class:-

->it is also known as instance inner class.it is defined outside of method block.inside non-static inner class,we can declare only non-static members.but can access any members of outer class.

->it can access through outer class object.

4)Anonymous Class:-

->it is declared inside the class.this class doesn’t have class name.

->it is created when creating the object of other class.

->it is used to override the method of a class only for that context,

->it is not used outside of the outer class & overrided method should be used only in the same class.

**Static Import**

The static import is used to access any static member of a class directly. There is no need to call by the class name.

import static java.lang.System.\*;

class StaticImportExample{

public static void main(String args[]){

out.println("Hello");//Now no need of System.out

out.println("Java");

}

}

**Ennumeration**

1)this is a type defn block,where we can declare only constant values.

2)it starts from enum keyword.

**Public enum** Day

{

*Monday*,*Tuesday*,*Wednesday*,*Thursday*,*Friday*,*Saterday*,*Sunday*;

**float**f=(**float**)3.14;

**publicstaticvoid** main(String[] args)

{

System.*out*.println("program starts");

System.*out*.println(Day.**class**);

System.*out*.println(Day.*Friday*);

Day.*values*();

System.*out*.println("program ends");

}

}

**File Handling**

**File**

1. If we have to save small amount of data then go for file otherwise go for database.
2. We can use java file object to represent file or directory.
3. In Unix, everything is treated as a file.Java file.io concept is implemented based on unix operating system. Hence java file object can be used to represent both files and directories.
4. File class is used to create file or directory.

**FileWriter**

1. Used to write character data into a file. Either it will override or append(append=true) with the existing data.
2. If specified file is not there then it will create file and write the character data.
3. Constructors and method.
4. Disadvantage : While writing the data by file writer we have to insert line separator(“\n”) manually which is vary from system to system. It’s difficult to programmer.

**FileReader**

1. Used to read character data from the file.
2. Read() method will return the int Unicode value characterwise. So we have to typecast to char at the time of printing. If next character is not available then it will return -1.
3. Read(Char[] char) will return the size of the array. So first get the no of character from file and fix the size.
4. Disadvantage : read characterwise. So performance is low.

**BufferedWriter**

1. To overcome above problem we should go for BufferedWriter and BufferedReader.
2. Same as FileWriter except line separator.
3. It can’t communicate directly with the file but communicate via any Writer object.
4. Whenever we are closing BufferedWriter then automatically internal writer will be closed. No need to close explicitly.

**BufferedReader**

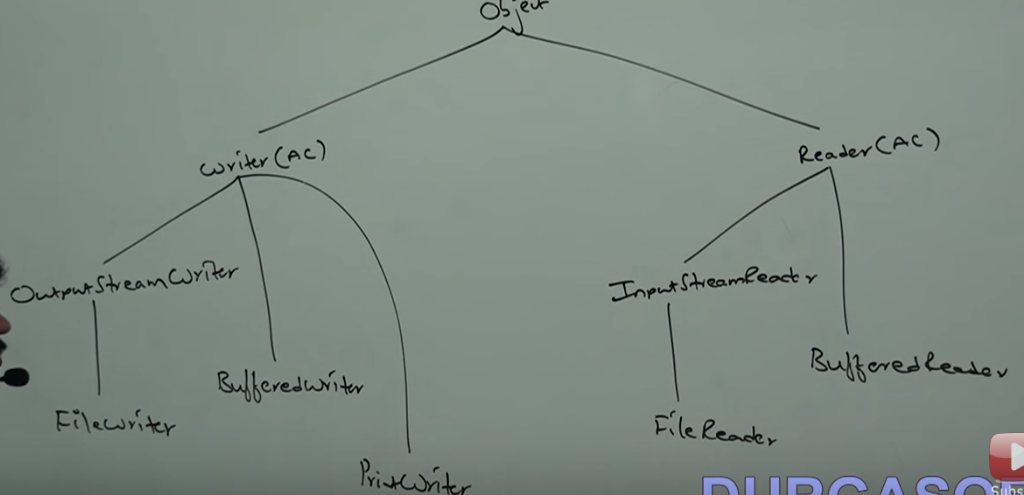
1. To read the data either character by character or line by line(advantage).
2. It can’t communicate directly with the file but communicate via any Reader object.
3. For empty file readLine() method will return null.
4. Whenever we are closing BufferedReader then automatically internal Reader will be closed. No need to close explicitly.
5. Most enhance reader to read character data from the file is BufferedReader.

**PrintWriter**

1. Through BufferedWriter or FileWriter we can only write int,char and string. If we have to write any type of primitive data then we can’t write directly but have to pass in string format. So our performance will goes down.
2. To inserting new line we have to call newLine() method again and again.
3. To overcome of this problem we have to use PrintWriter.Using printwriter we can write any type of primitive data directly usingprint() method.
4. For new line we have to use println() method.
5. It can communicate directly with the file and also communicate via any Writer object.
6. Diff between write(100) and print(100). Write(100) will add ‘d’ but print(100) will add 100 to the file directly.

**Conclusion** : the most enhance writer is PrintWriter and best reader is BufferedReader.

**Note** : In general we can use readers and writers to handle character data(text data), whereas we can use streams to handle binary data(like images, pdf files, video files, audio files etc..). We can use outputstream to write binary data to the file, inputstream to read binary data from the file.

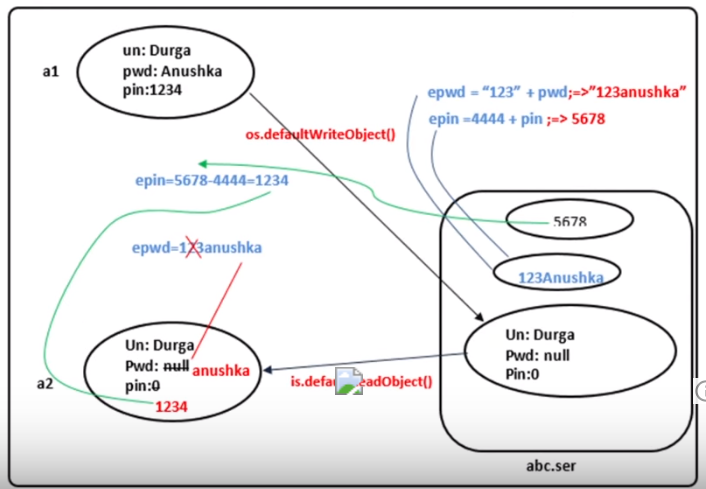


**Serialization**

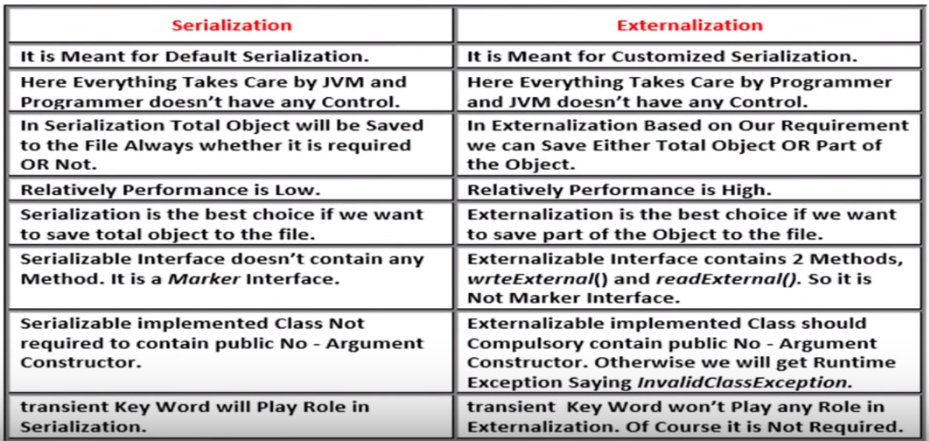
1. **Serialization** :Suppose we are writing state of the object into the file or we are converting an object from java supported form to file or network supported form that is called serialization.
2. For serialization we have to implement Serializable interface and use FileOutputStrea m or ObjectOutputStream. Then we have to call writeObject(emp) method.
3. Sender or receiver both have the values but updated state will transfer through the network.
4. **De-serialization**: When we are reading state of the object from the file or we are converting an object from file or network supported form to java supported form that is called deserialization.
5. For deserialization we have to implement Serializable interface and use FileInputStream or ObjectInputStream. Then we have to call readObject() method.
6. **transient**: Suppose if we don’t want to save the variable(password, pin etc..) due to security reason then we have to use transient keyword. Then JVM ignore original value and save default value into the file.
7. **Static**: Static variables are not the part of the object hence it won’t participate in serialization. If we are using static variable as transient then there is no use or impact.
8. **Final** : Final variables will be participated in serialization directly by their values. It will be replaced by value at compile time only. So if we are using final variable as transient then there is no use or impact.

|  |  |
| --- | --- |
| Declaration | Output |
| Int i=10;  Int j=20; | 10…20 |
| transient Int i=10;  Int j=20; | 0…20 |
| transient static Int i=10;  transient Int j=20; | 10…0 |
| transient Int i=10;  transient final Int j=20; | 0….20 |
| transient static Int i=10;  transient final Int j=20; | 10…20 |

1. We can serialize any number of objects into the file but in which order we serialized in the same order only we have to deserialize. Ex. oos.writeObject(Dog d1);oos.writeObject(Cat c1);oos.writeObject(Rat r1);(Dog)ois.readObject();(Cat)ois.readObject();(Rat)ois.readObject();
2. If we don’t know the order of the Objects then put if..else conditions.Ex. if(o instanceof Dog){Dog d2=(Dog)o;}else if(o instanceof Cat){Cat d2=(Cat)o;}
3. **Object Graph** : The group of objects which are reachable from the serialized object will be serialized automatically. This group of object is called object graph. For serializable, every group of object implements serializable interface.
4. **Need of Customized Serialization** : In default serialization, due to using transient keyword there may be a loss of information(password, pin etc..). To recover this loss of information we should go for customized serialization.
5. For that we have to override callback methods writeObject() or readObject() method. these methods will be executed automatically by JVM.



1. **Serialization with respect to Inheritance** : If parent class is serializable then by default every child is serializable even though child class doesn’t implement serializable or implement. That is serializable nature from parent to child. If Object class implement serializable then all the classes will by default serialized. GenericServlet implements serializable interface hence all the servlet class is serialized by default.
2. If parent class doesn’t implements serializable then also we can serialize child class object if child class implements serializable interface. At the time of serialization JVM will save default value for Non-serializable parent class into the file . At the time of deserialization JVM will check is any parent class is non-serializable or not. If available then JVM will execute Instance Control flow(Using default constructor mandatory otherwise InvalidClassException) or share it instance variable to the current.
3. To prevent subclass from being serialized we must implement writeObject() and readObject() method and need to throw NotSerializableException from these methods.
4. **Externalization :** In serialization total object will be saved always to the file and it is not possible to save part of the object which creates performance problem. To overcome of these problems we should go for externalization.
5. Through this we can save either total object or part of the object so that performance will be improved. This is the child interface of serializable and having two methods that is writeExternal() or readExternal().
6. At the time of serialization we have to use writeObject() method then it will call writeExternal() method. At the time of deserialization we have to use readObject() then JVM will create new object using public No-Argument constructor(manadatory). On that object readExternal() will be executed.
7. Transient keyword has no role in externalization because programmer have all control. So programmer doesn’t want to save then don’t save in writeExternal() method.



1. **Default Serial Version UIDProblems :** Both sender and receiver should use same JVM(Inix or windows) wrt to vendor and version otherwise receiver is unable to deserialize because of different serialVersionUID.
2. After serialization if we change .class file at receiver side then we can’t perform deserialization because of mismatch in serialVersionUID of local class of receiver and serialized object. So we get InvalidClassException.
3. To generate serialVersionUID internally JVM will use complex algorithm which may create performance problems.
4. We can solve above problems by configuring our own serialVersionUID.

**Marker Interface**

Looking carefully on marker interface in Java e.g. [Serializable](http://javarevisited.blogspot.com/2011/04/top-10-java-serialization-interview.html), Clonnable and Remote it looks they are used to indicate something to compiler or JVM. So if JVM sees a Class is Serializable it done some special operation on it, similar way if JVM sees one Class is implement Clonnable it performs some operation to support cloning. Same is true for RMI and Remote interface. So in short Marker interface indicate, signal or a command to Compiler or [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html).

**Generics**

1. Used to provide type safety and to resolve type casting problems.
2. **type safety** : Arrays are type safe because we can’t add the heterogeneous object at compile time but Collection is not type safe wrt to type because we can add the heterogeneous object at compile time but at runtime it will fail. Failing at compile time better than run time. EX. Create the name of the students.
3. **type casting :**in the case of array at the time of retrieval it’s not required to perform typecasting because there is a guarantee for the type of the elements present in array. But in the case of collection we have to perform typecasting at the time of retrieval because there is not guarantee for the type of the elements.
4. Main purpose of generics : we need not required fixed sized and need type safety and resolve typecasting problems then we need generics.
5. Using generics, At the time collection retrieval we have not required typecasting.
6. Polymorphism(usage of parent reference to hold child object) is only applicable for base type but not for parameter type(only class name or interface name not primitive).
7. In 1.5 Arraylist<T> where T is type parameter.We also can create our own generics class using type parameter <T>. We can take any valid java identifier but as a java convention we should take <T>.
8. There are two types of wildcards in generics bounded and unbounded.
9. **bounded**: Type can be upper bounded by using <T extends X>where all Types must be X type or child class of X or implementation class of X. Here X can be class or interface or both. class will come in first place then interface. We can extend only one class and multiple interface.
10. At the class level only extends <T extends X> is allowed. <T super X> is not allowed.
11. At the method level we have to use wild card. Ex. Test(ArrayList<String> list);Test (ArrayList<?> list);Test(ArrayList<? extends X> list);Test(ArrayList<?super x> list);
12. **Note1** : Test (ArrayList<?> list); we can call this method by passing list of anything but inside the method we can’t add anything except null. Because we don’t know the type exactly. null is allowed because it is the valid value for all datatype. this type of methods are best suitable for read only operation.
13. **Note2 :** Test(ArrayList<? extends X> list); we can call this method by passing list of X type or child class of X but inside the method we can’t add anything except null. Because we don’t know the type exactly. null is allowed because it is the valid value for all datatype. this type of methods are best suitable for read only operation.
14. **Note3 : lower bounded** : Test(ArrayList<?super x> list); we can call this method by passing list of X type or super class. if X is interface then X type or super class of implementation class of X. but inside the method we can add X type and null.
15. **Note4 :** ArrayList<?> lst1=new ArrayList<?>();we cann't write without bound at right side.
16. **unbounded**: Single <?>or <T>is called an unbounded wildcard in generic and it can represent any type, similar to Object in Java.
17. **Type Parameter** : We can declare this at two place either at class level(generic class ) or method level(generic method).We can defined bounded type at class level or method level.
18. If we send generic object to non-generic area then it start behaving like non-generic object and viceversa.
19. **At compile time** : type safety and to resolve type casting problems are applicable at compile time only. Hence generics concept is also applicable at compile time only.
20. At the time of compilation compiler will check the generics syntax. Then remove the generics syntax. Once again compiler will check the resultant code. hence for the JVM generics syntax won’t be available.

**JUNIT**

**Application Program Interface(API’S)**

1)Junit API-white box testing

2)ant API-build automation

Testing

3)log4g api-logging message of the application

White box testing

Black box testing

Gray box testing

4)**Black Box testing**-test the business program using application by tester.

5)**White Box Testing-**test the business program using code developed for the application by developer.also unit testing.

6)**Gray Box Testing-**sometimes developer tests the business program using code & sometimes tester test the business program using application.

Business Program/Business Class

-perform a task

-variables or properties

-methods or behavior

Test Program/Test Class

-verify all the variables

-verify all the methods behavior

7)for each business program we have to write one test program.

8)**Unit Testing Tools**

**Installation of Junit-4.11 for Testing**

1)install Junit from net then go to the project,right click then->properties->javabuildpath->libraies->addexternaljars->pathname of jarfile.

->Junit/TestNG-tools for white box testing

->nUnit

->plugUnit

9)**Junit**

1)a junit testcase is a program which consist of test method.

2)each test method should begin with test annotation.

3)a test annotation indicates that below method should be executed as a junit test.

4)if a junit test case doesn’t have any test method then the junit will not run the test case.

5)inside one junit test class we can develop multiple test methods.

6)after the execution of test cases,the junit report will be generated,this report gives the status of test cases executed.

7)in junit test case if we have to skip one of the test case during execution then such test case with begin with **@Ignore** annotation.

8)whenever junit test cases are executed only all test method are executed which doesn’t have **@Ignore**annotation.

9)**Rules of Test Method**

->all test method should be public

->all test method should have only void return type.

->all test method should be non static.

->the test methods should not be declared with parameters.

10)inside junit test case we can develop any method and that method can be invoked inside the test case.

11)**@Before** annotation method will be executed before the test cases.

12)**@After** annotation method will be executed after the test cases.

13)**@Before** and **@After** annotation method can be return anywhere in the program.

14)if any precondition are exist for executing the test method then its always advisable to write inside the before method.similarly all the post condition of test method should return inside after method.

15)@BeforeClass annotation is executed before @Before annotation.

16)@AfterClass annotation is executed after @After annotation.

17)after class and before class should be static method.

18)junit test case consists of test method.inside test method assertion method are used to verify the expected and actual value.if it is failed then it acts like a exception.after that it terminates the execution.all the assert methods are static and available inside the assert class.this assert methods are imported using static import.the static import imports only the static member of the package.

19)the assert() method compares expected and actual value.if both are same then junit report will be updated with pass status.if not matching then junit report will be updated with fail status.

15)**Step**

->@BeforeClass

->@Before

->@Test

->@Ignore

->@After

->@AfterClass

**Build Automation Tools**

1)use for compile the application or build the application.

2)two types of tools

->ANT-Apache,Another neet tool

->Maven

3)**Installation of Ant**

-**>**download apache-ant-1.8.4 from net.then st the path for ANT-HOME and bin folder like below

>set path=%path%;D:\Program Files\Java\jdk1.6.0\_05\binset the path for jdk

>set path=%path%;E:\SOFTWARES\javasoftware\apache-ant-1.8.4\bin

\*steps for set the path for ANT->

>set ANT\_HOME= E:\SOFTWARES\javasoftware\apache-ant-1.8.4

>set JAVA\_HOME= D:\Program Files\Java\jdk1.6.0\_05

>ant

Build file:build.xml does not exist!

Build fail

Now we have to create build.xml file

->in basedir we can write any path of src.

->we can create bat file(.bat) also to run the build.xml file.

->ANT software is used for building(compiling )all the java file from source file.

**Tracing Tools**

1)for tracing the project we have to use logger mechanism using log4j software or SLF4J.it contains the previous message in a file or console or database.it is called logging process & file is logger file.

2)when we want see all the process message of my computer then we can see through log file.

COMPUTER->RIGHTCLICK->MANAGE->EVENT VIEWER->WINDOWS LOG

Then we can all the problem of my computer.

3)windos OS is writing all the information in logger file.

4)for that we have to install lo4j software(.jar file) and add to eclipse through

project,right click then->properties->javabuildpath->libraies->addexternaljars->pathname of jarfile.

5)log4j will work,when we will create the properties file.we can save properties file(.properties) wherever we want but we have to give path in java file like("D:\\JAVA\\workspace\\JunitDemo\\log4j.properties”)

6)class contains the path of properties file and properties file contains the path of log file(sample.log) like D:\\JAVA\\workspace\\JunitDemo\\src\\com\\jspider\\log4j\\sample.log

7)

8)a log4j API’s is used for logging the information of a program during the execution.it provides six level of message which can be logged.

9)the log4j needs a properties file to log the messages.the properties file should contain following three information.

->the level of logs to be logged.we can get any level of log that has to be logged when execution.log level setting is common for all type of appender.

->Appender Types-the appender type indicates where the messages should be logged.log4j supports two types of appender.

-console appender & file appender

We can set multiple appender inside the properties.in such case the logger logs the message on each appender type.every appender should have a name.

->Pattern Layout-it specifies the format of a messages that has to be displayed in the specified appender.the format can be configured differently for fifferent types of appender.

10)write logger statement in a class.create an object of logger class by using getLogger() method of Logger class because Logger constructor is private in Logger class.

Static logger getLogger(string classname)

{

Return new Logger();

}

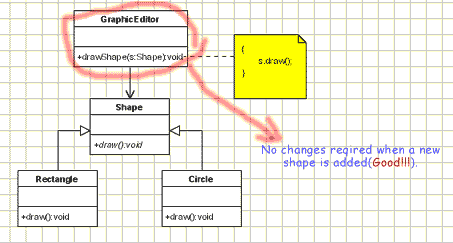
11)while invoking getLogger() method the class name should passed as an argument.

12)load the properties file in class by using a method configure of the class properties configurator.this method accepts the name of a properties file as a string.

13)write different level of log methods in the class.

|  |
| --- |
| **OPEN AND CLOSED PRINCIPLE** |

1)open for extension but closed for modifications.



**SINGLE RESPONSIBILITY PRINCIPLE**

* A class should have only one reason to change.

**Java custom annotation**

There are few points that should be remembered by the programmer.

1. Method should not have any throws clauses.
2. Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
3. Method should not have any parameter.
4. We should attach @ just before interface keyword to define annotation.
5. It may assign a default value to the method.

**Types of Annotation**

There are three types of annotations.

1. **Marker Annotation**

An annotation that has no method, is called marker annotation. For example:

@interface MyAnnotation{}

The @Override and @Deprecated are marker annotations.

1. **Single-Value Annotation**

An annotation that has one method, is called single-value annotation. For example:

@interface MyAnnotation{

int value();

}

We can provide the default value also. For example:

@interface MyAnnotation{

int value() default 0;

}

How to apply Single-Value Annotation

Let's see the code to apply the single value annotation.

@MyAnnotation(value=10)

The value can be anything.

1. **Multi-Value Annotation**

An annotation that has more than one method, is called Multi-Value annotation. For example:

@interface MyAnnotation{

int value1();

String value2();

String value3();

}

}

We can provide the default value also. For example:

@interface MyAnnotation{

int value1() default 1;

String value2() default "";

String value3() default "xyz";

}

How to apply Multi-Value Annotation

Let's see the code to apply the multi-value annotation.

@MyAnnotation(value1=10,value2="Arun Kumar",value3="Ghaziabad")

**Built-in Annotations used in custom annotations in java**

**@Target**

* @Target tag is used to specify at which type, the annotation is used.

The java.lang.annotation.ElementType enum declares many constants to specify the type of element where annotation is to be applied such as TYPE, METHOD, FIELD etc. Let's see the constants of ElementType enum:

Element Types Where the annotation can be applied

TYPE class, interface or enumeration

FIELD fields

METHOD methods

CONSTRUCTOR constructors

LOCAL\_VARIABLE local variables

ANNOTATION\_TYPE annotation type

PARAMETER parameter

Example to specify annoation for a class

@Target(ElementType.TYPE)

@interface MyAnnotation{

int value1();

String value2();

}

Example to specify annoation for a class, methods or fields

@Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})

@interface MyAnnotation{

int value1();

String value2();

}

**@Retention**

@Retention annotation is used to specify to what level annotation will be available.

RetentionPolicy Availability

RetentionPolicy.SOURCE refers to the source code, discarded during compilation. It will not be available in the compiled class.

RetentionPolicy.CLASS refers to the .class file, available to java compiler but not to JVM . It is included in the class file.

RetentionPolicy.RUNTIME refers to the runtime, available to java compiler and JVM .

Example to specify the RetentionPolicy

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE)

@interface MyAnnotation{

int value1();

String value2();

}

@Inherited

@Documented

@Target

**There are a few new features, but not many in java 1.5:**

For-each loop   
Static Import   
Autoboxing and Unboxing   
Enum   
Covariant Return Type   
Annotation   
Generics

**There are a few new features, but not many in java 1.6:**

1. @Override annotations on methods specified by an interface

NavigableSet, NavigableSet, Deque

2)jdbc enhancements:Automatic drver class

**There are a few new features, but not many in java 1.7:**

switch on String

1)Before JDK 7, only integral types can be used as selector for switch-case statement. In JDK 7, you can use a String object as the selector. For example,

String day = "SAT";

switch (day) {

case "MON": System.out.println("Monday"); break;

case "TUE": System.out.println("Tuesday"); break;

case "WED": System.out.println("Wednesday"); break;

case "THU": System.out.println("Thursday"); break;

case "FRI": System.out.println("Friday"); break;

case "SAT": System.out.println("Saturday"); break;

case "SUN": System.out.println("Sunday"); break;

default: System.out.println("Invalid");

}

2)Binary Literals with prefix "0b"

e.g.int number1 = 0b01010000101000101101000010100010;

3)Underscore for Numeric Literals

e.g.double aDouble = 3.1415\_9265;

4)Catching Multiple Exception Types

e.g.try {

......

} catch(ClassNotFoundException|SQLException ex) {

ex.printStackTrace();

}

5)The try-with-resources Statement

e.g. try (BufferedReader in = new BufferedReader(new FileReader("in.txt"));

BufferedWriter out = new BufferedWriter(new FileWriter("out.txt"))) {

int charRead;

while ((charRead = in.read()) != -1) {

System.out.printf("%c ", (char)charRead);

out.write(charRead);

}

} catch (IOException ex) {

ex.printStackTrace();

}

6)Type Inference for Generic Instance Creation

e.g. // Pre-JDK 7

List<String> lst1 = new ArrayList<String>();

// JDK 7 supports limited type inference for generic instance creation

List<String> lst2 = new ArrayList<>();

**There are a few new features, but not many in java 1.8:**

1)With static method:

You can call the methods with their names (references) with :: operator

class Test

{

public static int matchStringLength(String str1, String str2)

{

return str1.length() - str2.length();

}

}

public class Demo

{

public void printByStringLength()

{

String names[] = {"abcde","abc","ab","abcd", "a"};

Arrays.sort(names, Test::matchStringLength);

System.out.println(Arrays.toString(names));

}

public static void main(String args[])

{

new Demo().printByStringLength();

}

}

2)Parallel Sort

public class Demo

{

public static void main(String args[])

{

String fruits[] = {"guava","apple","banana"};

String fruits1[] = {"guava","apple","banana"};

// before JDK 1.8, using Arrays.sort()

Arrays.sort(fruits);

System.out.println(Arrays.toString(fruits));

// with JDK 1.8, using Arrays.parallelSort()

Arrays.parallelSort(fruits1);

System.out.println(Arrays.toString(fruits1));

}

}

parallelSort(char[] a) parallelSort(char[] a, int fromIndex, int toIndex)

parallelSort(byte[] a) parallelSort(byte[] a, int fromIndex, int toIndex)

3)Addition of Calendar.Builder

Before JDK 1.8, each date field is set separately with individual methods. Each set method added as a separate statement.

See the calendar instance in the following code.

public class Demo

{

public static void main(String args[])

{

Calendar calendar = Calendar.getInstance();

// populate calendar with individual set methods one-by-one as a separate statement

calendar.set(YEAR, 2013);

calendar.set(MONTH, APRIL);

calendar.set(DATE, 10);

calendar.set(HOUR, 8);

calendar.set(MINUTE, 56);

calendar.set(SECOND, 14);

calendar.set(AM\_PM, PM);

System.out.println(calendar.getTime());

// let us see the JFK 1.8 style

// all set methods are set as one statement

Calendar calendar1 = new Calendar.Builder()

.set(YEAR, 2013)

.set(MONTH, APRIL)

.set(DATE, 10)

.set(HOUR, 8)

.set(MINUTE, 56)

.set(SECOND, 14)

.set(AM\_PM, PM)

.build(); // one semicolon here

System.out.println(calendar1.getTime());

}

}

4)Introduction of Functional Interfaces

An interface containing only one abstract method is known as functional interface.

For example, the java.lang.Runnable interface is a functional interface as it contains only one abstract method run().

A new annotation, @FunctionalInterface, is introduced to raise compilation error if an interface marked as @FunctionalInterface contains more than one abstract method.

@FunctionalInterface

public interface Demo

{

public abstract void display();

}

5) Lambda Introduction

The strength of JDK 1.8 is introduction of Lambda function.

A) Lambdas with forEach loop

JDK 1.8 introduces a new method forEach() to use with lambda expression.

public class Demo

{

public static void main(String args[])

{

List<String> alphabets = Arrays.asList("A", "B", "C", "D");

System.out.println("Printing with earlier JDK 1.5 for loop:");

for(String str : alphabets)

{

System.out.print(str + "\t");

}

System.out.println("\nPrinting with latest forEach loop introduced with JDK 1.8 with lambda usage:");

alphabets.forEach(str ->

{

System.out.print(str + "\t");

}

);

}

}

public class Demo

{

public static void main(String args[])

{

List<String> cities = Arrays.asList("Delhi","Hyderabad","Chennai","Mumbai");

System.out.println("\n\nTo print the values earlier to Java 8:");

for(String str : cities)

{

System.out.print(str + "\t");

}

System.out.println("\n\nTo print using Lamda in Java 8 in more readable way using type:");

cities.forEach((String str) -> System.out.print(str + "\t"));

System.out.println("\n\nTo print using Lamda in Java 8 without using type but using \"value\":");

cities.forEach(value -> System.out.print(value + "\t"));

System.out.println("\n\nTo print using more concisely with :: operator:");

cities.forEach(System.out :: println);

}

}

**JDK Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1.5** | **1.6** | **1.7** | **1.8** |
| 1 | For-each loop | @override | switch on String | call the static methods with :: operator |
| 2 | Static Import | Jdbc enhancements : Automatic driver class | Binary Literals with prefix "0b" | Parallel Sort |
| 3 | Autoboxing and Unboxing |  | Underscore for Numeric Literals | Addition of Calendar.Builder |
| 4 | Enum |  | Catching Multiple Exception Types | Introduction of Functional Interfaces |
| 5 | Covariant Return Type |  | try-with-resources Statement | Lambdas with forEach loop |
| 6 | Annotation |  | for Generic Instance Creation |  |
| 7 | Generics |  |  |  |
| 8 | Linked List implements queue also. |  |  |  |

**Reflection API**

* API is a already developed program that is used by developer to develop own program.
* Group of classes that is used to reading a class at run time, loading its byte code,creating its object,calling the class member or modifying any fields at run time.
* E.g : javac,java,servlet,Hibernate
* **java.lang.Class** is the entry point for all the reflection operations. JVM creates immutable object of Class class that provides methods to read the runtime properties of the object and create new objects, call its method and get/set object fields.