JAVA INTRODUCTION

We chose Java with Selenium as it is popularly used language for Selenium **.**

**What is JAVA**?

Java is a high level programming Language. It is an object oriented language introduced by SUN MicroSystems in June 1995.

The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture.

**History of Java**

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.Java was originally designed for interactive television, but it was too advanced for the digital cable television industry at the time.The lanuage was initially called Oak after an oak tree that stood outside Gosling's office; it went by the name Green later, and was later renamed Java, from Java coffee, said to be consumed in large quantities by the language's creators. Gosling aimed to implement a virtual machine and a language that had a familiar C/C++ style of notation.

Sun Microsystems released the first public implementation as Java 1.0 in 1996. It promised "Write Once, Run Anywhere" (WORA), providing no-cost run-times on popular platforms. Fairly secure and featuring configurable security, it allowed network- and file-access restrictions.

Oracle Corporation acquired Sun Microsystems in 2009-2010(Jan 2010) and is currently owner of official implementation of Java SE platform.

**J2SE-> Java 2 Standard Edition**

Java started with 1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7(Latest)

(Note: 2 Never released)

**Features of JAVA**

Java Features :

The basic features that make Java a powerful and popular programming language are:

1)Compiled and Interpreted:-

It has both Compiled and Interpreter Feature .Program of java is First Compiled and then it is Interpreted .First of all The Program of java is Compiled then after Compilation it creates Byte Codes rather than Machine Language Then After Byte Codes are Converted into the Machine Language with the help of the Interpreter .

2)Platform Independent:-

Java Language is Platform Independent means program of java is Easily transferable because after Compilation of java program byte code will be created then we have to just transfer the Byte Code to another computer.

3)Object-Oriented:-

This is purely OOP Language, that is, all the Code of the Java Language is Written into the classes and Objects. This feature of java has made it Most Popular Language because it also Supports Code Reusability, Maintainability etc.

4)Robust and Secure:-

The Code of java is Robust ,Means it first checks the reliability of the code before Execution When We trying to Convert the Higher data type into the Lower Then it Checks the Demotion of the Code then It Will Warn the User Not to do this, hence it is Robust.

Secure :

* When We transfer the Code from One Machine to Another the First Check the Code either it is Effected by the Virus or not or it Checks the Safety of the Code if code contains the Virus then it will never Executed .
* The absence of pointers in Java makes it impossible for applications to gain access to memory locations without proper authorization as memory allocation and referencing model is completely opaque to the programmer and controlled entirely by the underlying run-time platform .

5)Simple, Small and Familiar:-

It is a simple Language Because it contains many features of other Languages like C and C++ and Java Removes Complexity because it doesn’t use pointers, Storage Classes and Go to Statements and java Doesn’t support Multiple Inheritance

6)Multithreaded and Interactive:-

Java uses Multithreaded Techniques For Execution Means Code is Divided into the Small Parts , those are Executed by java in Sequence and Timed Manner this is Called as Multithreaded .In this Program of java, it is divided into the Small parts & these are Executed by Compiler . Java is Called as Interactive because Code of java Supports CUI and Also GUI Programs.

7)Dynamic and Extensible Code:-

Java has Dynamic and Extensible Code Means With the Help of OOPS java Provides Inheritance and With the Help of Inheritance we Reuse the Code that is Pre-defined and also uses all the built in Functions of java and Classes

8)Distributed:-

Java is a distributed language which means that the program can be designed to run on computer networks. Java provides an extensive library of classes for communicating ,using TCP/IP protocols such as HTTP and FTP. This makes creating network connections much easier than in C/C++. You can read and write objects on the remote sites via URL with the same ease that programmers are used to when read and write data from and to a file. This helps the programmers at remote locations to work together on the same project.

**To work with Java**

**Download: jdk-6-windows-i586.exe**

**or**

**jdk-7-windows-i586.exe**

**URL:** [**http://www.oracle.com/technetwork/java/archive-139210.html**](http://www.oracle.com/technetwork/java/archive-139210.html)

or

**http://www.oracle.com/technetwork/java/javase/downloads/index.html**

**Or**

**Use Google + download Java**

Use any JDK with version greater than 1.5

Download JDK 1.6 or 1.7

**Note:**

-> JDK 7 is the latest

->To check if Java is installed, run Java in CMD prompt eg: c:\ Java

**Error: Java cannot be recognized as internal or external command**

**Reason: Java is not installed or path is not set**

**JDK**

Java Compiler

JRE (JVM) (‘JRE-Java Runtime Environment, JVM-Java Virtual Machine

Demo programs

Code Samples

Java DB(Don’t Install)

Public JRE (Don’t Install)

**Note**: Preferable ,don’t install in path containing spaces like c:\Program Files (It gives problem in J2EE). Install something like D:\Java\

* **Java Development Kit 7** means **Java 1.7**

**To Set Environment Path**

Environment variable PATH should be set to point to where the java binaries have been installed.

Open Java bin folder in the installed path, copy the path. Then go to My Computer, right click System Properties->Advanced->Environment Variables->Edit ,put ; paste it.

**->To check version of Java, run c:\java -version**

a)Use Editor s/w like Notepad,Notepad++,**TextPad,EditPlus**

b)IDE-Integrated Development Environment TOOLS

* JDeveloper
* NetBeans
* Eclipse
* IntelliJ IDEA

**1)What is a compiler?**

Compiler is a program that transforms source code written in a programming language into binary format or executable program.

**2)What is Interpreter?**

It executes the source code directly or translates the code one line at a time. These programs can run on computer having Interpreter.

Interpreter do generate binary code, but the code is never compiled instead interpreted each & every time the program executes

**3)HLL(High Level Language):** Programming Language which are more understandable by human and it is more natural language element., easier, making the process of developing programs simpler and understandable

**4)LLL(Low Level Language):** These languages which are close to m/c understandable language

**5)M/C Language:** This is the only language a microprocessor can process without any previous transformation

Assembly Language: It is also a low level language which uses assembler

Instruction Set or Binary Language

ALL(8085,8086,8051)

Mov X **Difficult to understand**

ADD AOX

cpu binary

CPU Memory

02

**\**

Translators converts HLL to LLL.Based on the way it is converted

* Compiler
* Interpreter

…………………………………

Source code

Executable Code

HLL Translator LLL(ALL)

**In Java,**

Translator

………………

…………………

Source code

Executable Code

(Class File)

BASICS

Interview Question:

**1)How do you say Java is platform independent?**

Compile into class file anywhere(any m/c) and run it on any m/c which has JVM

Source Code/File Executable

……………..

Sourcecode

Java Compiler

Byte Code

*XYZ.java javac XYZ.class*

ByteCode-Executable format of Java file.It is saved as filename.class

The programs written in Java has to be saved in a file with extension .java. This file is known as **source code**. **Source code(.java file)** is fed to java compiler, the java compiler converts source code into an executable format known as the **bytecode**. The bytecodes are saved in a file with extension **.class**.

The class files(.class) are automatically generated by the java compiler. The JVM understand only bytecode hence where ever JVM is available class files can be executed.

*Source Code/File Executable*

……………..

source code

Java Compiler

Byte Code

*XYZ.java JavaC XYZ.class*

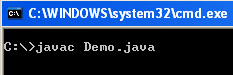
**Implementation/ Compilation Stage Execution Stage**

**Development Phase**

Javac->Compilation Eg: javac Demo.java

Java-> Execution Eg:java Demo

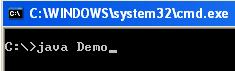
The source code written inside java file are converted into byte code(Class file) by using a command **javac**. The **javac** command is available in the bin folder where java is installed. The syntax is as below



Command Source file

After compilation ,the class files are generated in the default location where source code is available. The class files can be executed by using a command **java**. This command is also available in **Bin** folder

Syntax to run class file is as below



command Class file Name

Note: While running the class file .**class** extension should not be specified. Any java file will have,

class ClassName{

}

*Eg:*class Demo{

}

**Note:This Demo.java can be compiled but cannot be executed**

To execute any java file will have

class ClassName{

public static void main(String args[])

{

}

}

**Program**

1)Keywords(Reserve words)-public, static, final, int etc

2)Identifier -int A char A

Identifier

Keyword

3)Literals-Values assigned to the identifier

**Rules for saving a java file:**

1)Starting character of a class name should be uppercase (Industry Practice)

Eg: 1)classTestClass (Correct) 2)class testclass (wrong) 3)class Test (Correct)

It works even if you don’t use uppercase

2)Class Name & the file name should be same

**Interview Question:**

**1)What is the minimum lines to compile java program?**

Eg: class demo{}

Note: I can write & compile without using main() method. But cannot run without it.

**2)What is difference between JDK,JRE and JVM?**

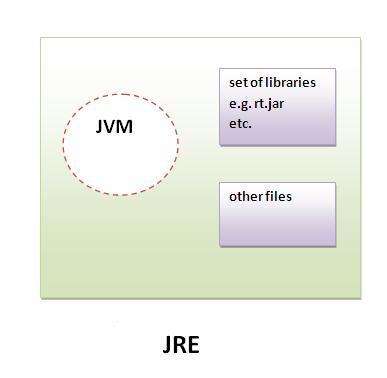
**JDK:**

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.



**JRE:**

JRE stands for Java Runtime Environment. It is this Environment where JVM executes the bytecode and contains class libraries & supporting files. It physically exists.



**JVM:**

JVM is an acronym for Java Virtual Machine, it is an abstract machine where java bytecode can be executed. JVMs are available for many hardware and software platforms (so JVM is platform dependent).

The JVM performs

* Loads code
* Verifies code
* Executes code

FYI:

JRE=JVM + Java Packages

JRE is targeted for execution of java files

JRE contains JVM, Class libraries and other supporting files. It does not contain any development tools such as compiler, debugger etc

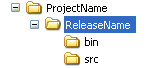
Actually JVM runs the program and it uses the class libraries and other supported files provided in JRE.

**General practice in industry** (bin or classes)(bin most commonly used)



.java .class

.java .java .class .class



If you organize this way compiling & running will be in different path. You have to cut & paste the class in **bin** folder. This becomes a problem. You can alternately use

**Eg: E:\Demo\src javac –d ..\bin Demo.java**

Instead of creating class file in the same path as Java file it generates class file in the path specified.

To run(go to bin dir)

cd ..\bin

Then

**E:\Demo\bin java Demo**

**class Name**

**{**

**………………….**

**…………………**

**Body**

**…………………..**

**}**

**Members ofClass**

* Methods(performs an operation)
* Variables (stores information)

**Note(imp):** To execute the class there should be atleast one method(**main method**)

public static void main(String[] args)

{

}

**S**ystem.out.println("Message"); // to print a message on the screen //S->Ucase

**Program 1:**

class Demo

{

// while taking notes we just use **psvm** in short

public static void main(String[] args)

{

System.out.println("Hi, Welcome to basics of Java"); // while taking notes use **Sop** in short

}

}

**Class:** Class can be defined as a blueprint / template to create an object.A class specifies the design of an object. It states what data an object can hold and the way it can behave.

**Members of Class:**

* **Fields**:These are properties of the class or object which we are going to create.***Ex:*** If we are creating a class called **Car** then they have property like model, cc,color,seats,category etc.. In a simplified way it can said as **variables** holds values.
* **Methods:** A method is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name.  Method can return a value.

**Program 2:**

class Demo2

{

public static void main(String[] args)

{

System.out.println("Hi” + “Welcome” +”to”+”Java class”);

}

}

**“+”** operator is known as **concatenation operator** which is used to join any two string or variable values.

**Note:**

1. Every Java program is a class.  The program starts with the name of the class.  **This name must be the same name as the .java file in your folder.** In this case the file is saved as Demo2.java.

2. Class names must begin with a letter, an underscore or a dollar sign.

Class names may contain only letters, digits, underscores and/or dollar signs.

Class names cannot use reserved words.

Class name cannot have space b/w words

3. Comments are used for identification purpose of the program & the programmer

a. // text - is the format for single line commenting.

b. /\* text\*/ - is the format for block commenting/multi line commenting

c. /\*\*documentation \*/ - documentation commenting.

Comment statements are ignored by the compiler.  It is a message to the reader of the code.

4. Every class,methods,blocks inside the class should start with “{“ and end with “} “

**Tokens:** The smallest individual entities in a Java program are called as Tokens.They are classified as

1.Reserved words(Key words)

2.Identifiers (classes,methods,variables,objects,packages and interfaces)

3. literals (values)

4.Operators + - ,++,-- etc

5.Separators {} () [] ; , ’

**Variables :** Variable is named memory location which can hold value and the value can change any no of times during execution. It is also an **identifier** in java.

1.The first character in a variable should not be a digit.It should start with a letter or a $ or \_ .

2.A variable name may consist of alphabets, digits , underscore and the dollar character.

3. A keyword should not be used a variable name

4. There should be no space in a variable name.

**Legal:**

int \_a;

int a\_very\_long\_variable\_name;

**Illegal:**

int :a;

int –d;

int e#;

int .f;

int 7g;

1. int i; // Declaration

i=10; // initialization

1. int i=10; // both Declaration and initialization

**Note:**

**1)**Every statement should end with ;

2)If variable is not initialized before using java throws error(local variable)

3)If you are not using the variable which is not initialized , it does not throw any error.

1)int i;

i=10;

2)int i=10; All works

3)System.out.println(i=10);

Note:

1. Whenever a variable is declared within a method it is a local variable
2. Whenever a variable is declared outside a method it is Global Variable/Member
3. Global Variables have default values, local variables should be initialized before using.
4. The initialization can be done at the time of declaration or separately or at the time of usage
5. If the global variable has to be initialized, then it can be done at global level like

static int i=10; //correct

static int i;

i=10; //compiler error

class Demo3

{

public static void main(String[] args)

{

int i; // declaration of local variable i

i=10; //local variables should be initialized before using

System.out.println("main starts");

System.out.println(i);

System.out.println("main ends");

}

}

o/p



class Demo4

{

static int i; // Global variable declared

public static void main(String[] args)

{

System.out.println("main starts!");

System.out.println(i); // printing value of i

System.out.println("main ends!");

}

}

o/p



**Note:**

1)Any variable declared in the class definition block is known as Global variable

2)The Global variables are initialized automatically by Java compiler at the time of compilation if not initialized

3)The default value for number data types i.e. integer, long, float, double is 0(Zero) where as for String it is Null.

**Datatype** can be classified as

* Primitive datatype - byte,short,int,long,float,double,char,boolean
* Derive Datatype – String

**Variable** are based on the above types

* Primitive variables
* Reference variables

class Demo5

{

static byte b;

static short s;

static int i;

static long l;

static float f;

static double d;

static char c;

static boolean flag;

public static void main(String[] args)

{

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

System.out.println("default value of byte is"+b);

System.out.println("default value of short is"+s);

System.out.println("default value of int is"+i);

System.out.println("default value of long is"+l);

System.out.println("default value of float is"+f);

System.out.println("default value of double is"+d);

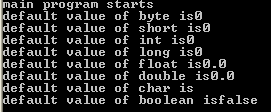
System.out.println("default value of char is"+c);

System.out.println("default value of boolean is"+flag);

}

}

o/p

 Default value of char is ‘\u0000’ or called as null character.

class Demo6

{

static String str; //String variable

public static void main(String[] args)

{

System.out.println("null");

System.out.println(str);

}

}

o/p



OPERATORS

An operator is a symbol that allows a programmer to perform certain operation like (arithmetic or logical) on data and variables.

1)Arithmetic (\*,/,%,+,-,++,--)

2) Assignment(=,+=,-=,\*=,/=,%=)

3)Relational (==,!=,<,>,<=,>=)

4)Instance of (instanceof)

5)Logical( &,|,^,!,&&,||)

6)Conditional(?:)

Depending on the no of operands, Operators can be of the following three types.

* Unary operator: It takes one operand, such as ++x,y--. Here x and y are variables and ++ and – are operators
* Binary operator: It takes two operands , such as x + y, x>y. Here x and y are variables while + and > are operators
* Ternary operator: It takes three operands,z=x>y?x:y, here x& y and z are variables, while ?: is an operator

***1.Arithmetic Operator***

|  |  |
| --- | --- |
| **Arithmetic operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| -- | Decrement |

class Demo7{

public static void main(String[] args) {

int i,j,k;

i=30;

j=20;

k=i+j;

System.out.println("Sum of i and j is " + k);

k=i-j;

System.out.println("Difference of i and j is " + k);

k=i\*j;

System.out.println("Product of i and j is " + k);

k=i/j;

System.out.println("Division of i and j is " + k);

}

}



class Demo8 {

public static void main(String[] args) {

int num1=18;

int num2=4;

int quotient=num1/num2;

int remainder=num1%num2;

System.out.println("num1 / num2 =" + quotient);

System.out.println("num1 % num2 =" + remainder);

}

}

****

***2.Assignment Operators***

The equal (=) symbol is known as the assignment operator. The assignment(=) operator is used to store or assign a value to a variable.

**Eg:**

x=10; // x stores 10

x=x+10; //stores 20

x=x+5 \* 20 //x stores 120

In **compound assignment operator** we have

+=,-=,\*=,/= and %=

**Eg:**

i=10; // i is initialized with the value 10

i+=15; // i is compound assigned with 25

i-=5; // i is compound assigned with 20

class Demo9{

public static void main(String[] args) {

int i=2,j=3,k=4,l=25,m=7;

i+=5;

j\*=6;

k/=2;

l-=10;

m%=5;

System.out.println(i);

System.out.println(j);

System.out.println(k);

System.out.println(l);

System.out.println(m);

}

}

o/p->



a\*=5 ;

a=a\*5;

a\*=5 + 4;

a=(a\*5) + 4;

***3.Relational Operators***

|  |  |
| --- | --- |
| Relational Operator | Description |
| == | Equal to |
| != | Not equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |

**Eg1**:

public class Demo10 {

public static void main(String[] args) {

int x=5;

int y=8;

int z=10;

boolean res;

res= x==y;

System.out.println(res);

res= x!=y;

System.out.println(res);

res= x>y;

System.out.println(res);

res= x<y;

System.out.println(res);

res= x>=y;

System.out.println(res);

res= x<=y;

System.out.println(res);

res= x>=y && y >=z;

System.out.println(res);

}

}



class Demo11 {

static int val1=10;

static int val2=8;

public static void main(String[] args) {

if(val1==val2)

{

System.out.println(val1 + " and " + val2 + " are equal");

}

if(val1!=val2)

{

System.out.println(val1 + " and " + val2 + " are not equal");

}

if(val1<val2)

{

System.out.println(val1 + " is less than " + val2 );

}

if(val1>val2)

{

System.out.println(val1 + " is greater than " + val2 );

}

}

}

****

**Object Equality**

The relational equality operators(==) and (!=) can also be used o compare the references of the two objects

Obj1==Obj2 //mean that Obj1 and Obj2 are equal and have same object reference

Obj1!=Obj2 //mean that Obj1 and Obj2 are not equal and does not have same object

//reference

**instanceof Operator**

The instanceof operator is a binary operator that checks whether an object is of a particu

lar type(here, type can be class, interface or an array). It is used for object reference variables only.

You can’t use the instanceof operator to test across two different class hierarchies.

class Demo12{

public static void main(String args[])

{

String s=new String("Hello");

if(s instanceof String)

{

System.out.println("s is an instance of String");

}

}

}

****

***4.Logical Operators***

|  |  |
| --- | --- |
| Category | Operators |
| AND | & |
| OR | | |
| NOT | ! |
| OR(Short circuit) | || |
| AND(Short circuit) | && |

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | a & b | a|b |
| True | True | True | True |
| True | False | False | True |
| False | True | False | True |
| False | False | False | False |

The NOT(!) operator returns the opposite of the current value of a Boolean operand.

The operators || and && evaluate only Boolean values

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | a && b | a||b |
| True | True | True | True |
| True | False | False | True |
| False | True | False | True |
| False | False | False | False |

**Interview Question:**

1.What is the difference b/w | and || , & and && ?

***5.Conditional Operator***

The Conditional Operator (?:) is a ternary operator that takes three operands. It works similar to if-else statement.

**Syntax:**

Operand1? Operand2:Operand3

**Ex:**

int i=20,j=25;

boolean test=false;

test=i>j ? true : false;

public class Demo14 {

public static void main(String[] args) {

int i=20,j=25;

boolean test=false;

test=i>j ? true :false;

System.*out*.println("The value of test " + test);

}

}

***6.Unary Operator***

++ Increment Operator

-- Decrement Operator

Post

Pre

Post Increment

Post Decrement

Pre Increment

Pre Decrement

class Demo15

{

public static void main(String[] args)

{

System.out.println("main starts");

int i=0;

int j;

j=i++; // Use first the current value and then increment

System.out.println("i="+i);

System.out.println("j="+j);

System.out.println("main ends.....");

}

}

o/p

****

class Demo16

{

public static void main(String[] args)

{

System.out.println("main starts");

int i=0;

System.out.println("i="+i);

i++; // Use first the current value and then increment

System.out.println("i="+i++);

System.out.println("main ends.....");

}

}

****

class Demo17

{

public static void main(String[] args)

{

System.out.println("main starts");

int i=0;

int j=0;

j=i + i++ + i + i++;

//0 0 1 1 //usage value of i

System.out.println("i="+i);

System.out.println("j="+j);

System.out.println("main Ends.....");

}

}

****

class Demo18

{

public static void main(String[] args)

{

System.out.println("main starts");

int i=0;

int j=0;

j=++i; //First increment and then use the value

System.out.println("i="+i);

System.out.println("j="+j);

System.out.println("main Ends.....");

}

}

****

class Demo19

{

public static void main(String[] args)

{

System.out.println("main starts");

int i=0;

int j=1;

int k=i + j++ + ++i + ++j + i++;

//0 1 1 3 1

System.out.println("i="+i);

System.out.println("j="+j);

System.out.println("k="+k);

System.out.println("main ends.....");

}

}

****

Methods:

Starts with lowercase

returntype methodName(datatype arg1,datatype arg2)

{

………………

…………………. //body of method

………………….

return value;

}

* If it is not returning anything use void
* In the method not returning anything you can skip return or use just **return;**
* To call the method, use **method name**
* Write methods in class definition block
* If **return type** is mentioned compulsorily it should **return the value** of **type mentioned**
* If return type is mentioned as void & returns something from method , it will return error

class Demo20

{

public static void main(String[] args)

{

System.out.println("main starts");

int i=test();

System.out.println("i="+i);

System.out.println("main Ends.....");

}

static int test()

{

System.out.println("Inside test method...");

return 100;

}

}

****

**Assignment:**

1.Write a java program (using method) which takes two number & return the sum

2. Write a java program (using method) which takes two number & return the product

3.Write a java program (using method) which takes two number & return the difference

class Demo21

{

public static void main(String[] args)

{

System.out.println("main starts");

int i=0;

int j;

j=test(i) + ++i + i++;

//2 1 1

System.out.println("i="+i);

System.out.println("j="+j);

System.out.println("main Ends.....");

}

static int test(int a)

{

System.out.println("a=" + a++);

return ++a;

}

}

****

Decision statements / Branch Statements

These statements allow you to create programs where decisions are taken based on the expression specified in the statements.

1. if statement
2. if….else statement
3. switch statement

If expression>

Execute if block

continues

false

True

***Basic structure of the if statement***

“if” block is executed only when the if expression is true. Otherwise , the “if” block is skipped. We must use Boolean expression in the if statement to make decisions.

public class Demo22 {

public static void main(String[] args) {

int age;

age=16;

if(age<18)

{

System.out.println("Not eligible to vote");

}

}

}

//Multiple statements in if block

public class Demo23 {

public static void main(String[] args) {

int age;

age=16;

if(age<18)

System.out.println("Not eligible to vote");

age=age+2;

System.out.println("Age of the person is " + age);

}

}

o/p



Note: In the above case only statement next to if is executed if the expression is true. The other statements gets executed as usual .

**Ex:** If we change age to 18 or more the next statement will not be executed.

In order to overcome the confusion it is recommended to use curly braces enclosing the if block statements.

if(age<18)

{

System.out.println("Not eligible to vote");

age=age+2;

System.out.println("Age of the person is " + age);

}

If <xpresion>

Execute if block

continues

Execute else block

true false

***Basic structure of the if..else statement***

public class Demo24 {

public static void main(String[] args) {

int a=10,b=20;

if(a>b)

{

System.out.println("a is greater than b");

}

else

{

System.out.println("b is greater than a");

}

}

}

//multiple if …else statement

public class Demo25

{

public static void main(String[] args)

{

int a=10,b=20,c=30;

if(a>b && a>c)

{

System.out.println("a is greater than b and c");

}

else if (b>a && b>c)

{

System.out.println("b is greater than a and c");

}

else

{

System.out.println("c is greater than a and b");

}

}

}

**Note**:

1. An if statement can be used without an else statement
2. Multiple if….else statements can be used in a program
3. Once an if …else statement causes an action in a program , then the remaining if…else statements will be ignored.

**switch …case Statement**

The switch …case statement is used to select an action from a given set of actions , based on a specified expression.

**Syntax:**

switch (expression/variable)

{

case value1: statement1;

case value2: statement2;

case value3: statement3;

[default: default\_statement;]

}

The expression /variable in the preceding code snippet can be any expression depicting a char,byte,int or enum variable. The switch case also support some wrapper classes like Integer,Byte ,Short

(In JDK 1.7 , we can also use String in values.)

public class Demo26 {

public static void main(String[] args) {

char country=’i’;

switch (country)

{

case‘b’: System.out.println(“Brazil”);

break;

case ‘n’:System.out.println(“New Zealand”);

break;

case ‘i’:System.out.println(“India”);

break;

default: System.out.println(“Invalid”);

}

}

}



**Note:**

a. If you do not use break statement then all the statements in the various case blocks following the matching expression are executed.

b. A case constant must be unique..duplicate values are not allowed

c. A case constant cannot be smaller than the width of the variable passed in switch

**When can we use if …else and switch …case?**

* if statements are used to evaluate Boolean expression.
* A *switch* statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.
* When we use Range of values, Relational operators or Logical operators then we use if…else.
* If we are using only values (list of values) the we go for switch..case

***for loop***

Initialize the loop

Check ( expression)

Loop Terminate

Execute for block

Execute Increment or Decrement expression

false

**true**

Basic structure of the Loop

For.. loop is initialized first and then the boolean expression is checked. If the expression evaluates to true, then the for block is executed, otherwise, the loop terminates. If the for block is executed, then the increment or decrement expression is updated to continue the loop.

**syntax:**

for (initialization;expression;increment/decrement expression)

{

//body

}

The 3 parts of the for loop –initialization,expression and increment/decrement-must be separated by semicolon or else the program will lead to a compile –error.

public class Demo28 {

public static void main(String[] args) {

for(int i=1;i<=20;i++)

{

System.out.println(i);

}

}

}

a)

for(i=1,j=1;i<=20;i++)

{

System.out.println(i);

System.out.println(j); /it is illegal to use it this way

}

b)

for(i=1,j=1;i<=5 && j>1;i++,j++)//it is legal

{

System.out.println(i);

System.out.println(j);

}

c)

for(i=1,j=1;i<=5, j>1;i++,j++)/ /multiple expressions (compiler error)

{

System.out.println(i);

System.out.println(j);

}

d) int i=1,j;

for(j=1;i<=20;)

{

System.out.println(i);

System.out.println(j);

i++;

}

//above code will work but it acts more like while loop

e) **for**(**int** i=10;i>5;i--)

{

System.*out*.println(i);

}

// above code is an example for decrementing for loop

***WHILE LOOP***

while(expression)

Execute whileblock

Loop terminates

False

True

**Basic structure of the while Loop**

public class Demo29{

public static void main(String[] args) {

int i=1;

while(i<=20)

{

System.out.println(i);// display i

i=i +2;

}

}

}

Note: While loop can be used with non numeric conditions also. Like checking a character in a variable or checking for a String value in a variable

**Ex:**

char ch;

while(ch!=’y’)

{

----- body

}

***Do-while loop:***

Check (expression)

Execute do-while block

Loop terminates

true

false

**Basic structure of the do… while Loop**

do……while loop works similar to while loop, but the difference is do…while loop block executes at least once irrespective of whether the expression evaluates to true or false.

**What is the difference b/w for loop and while/do while ?**

for loop checks for numeric conditions and have definite iterations.

while or do while loop can have both numeric and non numeric conditions. It has definite iterations with numeric condition and not finite iteration with non numeric conditions

public class Demo30{

public static void main(String[] args) {

int i=1;

do

{

System.out.println(i);// display i

i++;

}while(i<=20);

}

}

**//Demo for taking char input from user**

**//Demo for not finite iteration**

import java.util.Scanner;

public class Demo31 {

public static void main(String[] args) {

char ch='y';

Scanner input;

String s;

do{

System.out.println("your choice");

input = new Scanner(System.in);

s=input.nextLine();

ch=s.charAt(0);

}while(ch!='y');

}

}

**//Printing alphabets**

class PrintAlpha{

public static void main(String args[])

{

for(int i='A';i<='Z';i++)

{

System.out.println(i);

}

for(char ch='A';ch<='Z';ch++)

{

System.out.println(ch);

}

}

}

***Implementing the for….each Loop(enhanced for loop)***

The for…each loop is used to iterate over arrays and collections. The for …each loop has only two parts, unlike the traditional which has 3 parts.

The two parts of the for…each loop are variable and collection.

Syntax:

for(var:collection)

{

//body

}

public class Demo32 {

public static void main(String[] args) {

int arr[]={1,3,5,7,9};

System.out.println("Array elements are");

//enhanced for loop

for(int i:arr)

{

System.out.println(i);

}

//traditional for loop

for(int j=0;j<arr.length;j++)

{

System.out.println(arr[j]);

}

}

}

Implementing Jump Statements

1. **break**
2. **continue**
3. **return**

**Break statement**

It helps to terminate the loop and transfer the control in decisional and iteration statements.

public class Demo33 {

public static void main(String[] args) {

int j=0;

while(true)

{

j=j+2;

if (j==6)

{

break;

}

System.out.println(j);

}

}

}

In the above code control breaks out of the loop when decisional if statement becomes true. break can be used in all loops

**Labeled break statement:**

It is used only in nested ,decisional, and iteration statements. A label in a labeled break statement is similar to an identifier , which is used before the loop. The identifier is always followed by a colon.

public class Demo34 {

public static void main(String[] args) {

int j,k;

demo:

for(j=0;j<5;j++)

{

System.out.println("Value of J is =" + j);

for (k=2;k<5;k++)

{

System.out.println("value of k is " + k);

if (j==k)

{

break demo;

}

}

}

System.out.println("for exited");

}

}

**continue statement :**

In unlabeled statements , the continue statement causes the current iteration of a loop to stop and continue with the next iteration.

However, in labeled statements, the continue statement stops the current iteration of the outer loop and continue with the next iteration

**public class** Demo35 {

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

**for**(**int**j=0;j<10;j+=2)

{

**if**(j==6)

{

**continue**;

}

System.*out*.println(j);

}

}

}

**o/p**

0

2

4

8

**Using the Labeled Continue**

public class Demo36 {

public static void main(String[] args) {

int j,k;

demo:

for(j=0;j<5;j++)

{

for (k=2;k<5;k++)

{

if (j==k)

{

continue demo;

}

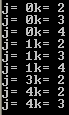
System.out.println("j= " + j + "k= " +k);

}

}

}

}



**return statement**

It causes the control of execution to return the caller of a method. While using the return statement, you must be careful about the return type of the method. If the method is of the void type then you should not use return or use just return.

LOCAL AND GLOBAL

**Global Local**

400

30

**i i**

**J**ava always gives **preference to local variable** when same variable is present in both local & global.

* Any executable statement should be part of a method
* **“.”**operator is used to access the Global member or Global variable specifically

**Note:** If local variable is not same as global variable, then you can directly access the global variable

Whenever we use Global variable always use **className.MemberName** . Don’t use it directly.

class Demo37

{

static int i;

public static void main(String[] args)

{

System.out.println("main method starts");

System.out.println("i="+i);

i=20;

System.out.println("i="+i);

i=30;

System.out.println("i="+i);

int i=400;

System.out.println("i="+i);

i=500;

System.out.println("i="+i);

System.out.println("i="+Demo37.i);

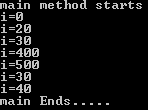
Demo37.i=40;

System.out.println("i="+Demo37.i);

System.out.println("main Ends.....");

}

}

****

STATIC AND NON-STATIC INTRODUCTION

**IQ: What is the difference b/w static members & non static members?**

A static member has only one copy of variables that share among all the objects of the class whereas a non-static member has its own copy of instance variable.

**Note: Using Class Name and .operator you can access static members**

class Demo38

{

static int i=20;

public static void main(String[] args)

{

System.out.println("main starts");

Demo38.test();

System.out.println("main ends.....");

}

static void test()

{

System.out.println("i="+ Demo38.i);

}

}



**Note:**

If we try to access non static members(variables/methods) we have to always use **reference.**

1. Primitive Data types 🡪 All primitive data types are declared using data type name.

Like int i;

double d;

1. Reference variable 🡪 To create reference variable use class name (Derived types)

String s1;

int i; //i is a variable of type int

short s; //s is a variable of type short

boolean b; b is a variable of type Boolean

A a1

B b1

C c1

Rama r1;

Sita s1;

Demo d1;

Reference variables Reference variables

int i=10;

boolean b=true;

double d=10.20;

We can store the values because we know the data type as we have declared it.

What to store for a1? What to store for b1?

-Demo3 d3; // d3 is a variable of type Demo3

-a1 is a variable of type A and can hold something which is of type A.

a1=new A();

b1= new B();

s1= new Sita();

r1=new Rama();

d1=new Demo1();

Demo3 d1= new Demo3();

**Note:**

* If we are accessing non static members we should use **reference variables**.
* If we are accessing static members we should use **ClassName and .operator**

class Demo39

{

int i=10;

public static void main(String[] args)

{

System.out.println("main starts");

Demo39 d39=new Demo39();

System.out.println("i="+ d39.i);

d39.test();

System.out.println("main ends");

}

void test()

{

System.out.println("test() method: i="+ i);

}

}

**O/p**

****

**Note:**

* Non static members can be accessed by nonstatic methods.
* Static members can be accessed anywhere in the program, otherwise we have to use reference variables.

**A a1=new A()**

a1 is reference variable of type A and we are storing type of A.

Memory location

Stack Heap Storage purpose

Stack is for execution purpose

Linkers

Loaders Program in compiler to link necessary libraries

Memory location

A

main

a1

Java

Command

Class Loader

Stack Heap

**When we run Class Loader it loads main() method as it is static. main has to get executed so it has to come to stack for execution**

A a1=new A();

a1 points to class A(object)

**Until we create object we cannot access non static members**

a1.i=10;

A

main

a1

Java

Stack Heap

A a2=new A();

a2.i=20

A

A

main

a1 a2

Java

Stack Heap

**When main method is done java calls, garbage collector before terminating, to clean the memory.**

**Note:**

* static will be **only one copy** in memory, non static can have many copies
* static members are also called as **class members**,
* non static members are called **reference members**, **object members** or **instance members.**

Static block

* It is a block { code} with keyword static which gets executed **before the main** method and is used for initializing the static variables.
* If there is static block, java first execute the static block then the main method
* You can have **multiple static blocks**.
* Always it executes in **top to bottom sequential order**..
* static block is used to **initialize static variables**.
* By default static variables are initialized to **default values** of 0.
* Inside static block **only static variable** can be used.
* If we use non static variables it throws **error**.
* This block is called **static initialization block**.

**static**

**{**

**……………**

**………….. Static Block**

**}**

class Demo40

{

static

{

System.out.println("3: I am the first to execute");

}

static

{

System.out.println("1: I am the first to execute");

}

public static void main(String[] args)

{

System.out.println("main starts");

System.out.println("main ends.....");

}

static

{

System.out.println("2: I am the first to execute");

}

}



class Demo41

{

static

{

System.out.println("SIB2");

i=30;

}

static int i=10;

public static void main(String[] args)

{

System.out.println("main starts");

System.out.println("i="+ Demo41.i);

System.out.println("main Ends.....");

}

static

{

System.out.println("SIB1");

i=20;

}

}

**o/p**



The static variable will have value which was **last assigned.**

A class contains two types of members

1. **static members**
2. **Non static members**

**Static members:**

1. The static members are declared with the help of keyword static
2. static members are loaded into the memory while loading the class
3. Throughout execution of java program only 1 copy of static member is available in the memory
4. The static members are also known as class members
5. All static members can be accessed by using . Operator on the class name.
6. Java language provides separate block to initialize static members of the program. This block is known as static initialization block
7. Before executing main method, static initialization block will be executed first
8. A program can have any number of static initialization block
9. The execution of the block is sequential (Top to bottom)
10. The static variables will have the recent values initialized by last static initialization block

class Demo42{

int i,j;

public static void main(String args[])

{

System.out.println("main starts");

Demo42 d1=new Demo42();

System.out.println("i="+ d1.i);

System.out.println("j="+ d1.j);

System.out.println("............");

Demo42 d2=new Demo42();

System.out.println("i="+ d2.i);

System.out.println("j="+ d2.j);

System.out.println("............");

d1.i=123;

d1.j=456;

System.out.println("i="+ d2.i);

System.out.println("j="+ d2.j);

System.out.println("............");

System.out.println("i="+ d1.i);

System.out.println("j="+ d1.j);

System.out.println("main ends");

}

{

System.out.println("1: Non static block");

i=20;

j=30;

}

{

System.out.println("2: Non static block");

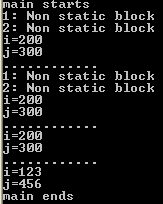
i=200;

j=300;

}

}

o/p



non – static block

* Non static block is also called as **instance initialization block(IIB)** used to initialize non static members.
* It gets **executed whenever an object of the class is created**.
* **IIB** can be called ( **multiple times** ) whenever the object is created.
* **SIB is created only once**.

class Demo43{

static int i;

int j;

public static void main(String args[])

{

System.out.println("main starts");

System.out.println("i= " +Demo43.i);

Demo43 d1= new Demo43();

System.out.println("j= " +d1.j);

d1.j=40;

Demo43 d2=new Demo43();

//value is of d2 object is printed

System.out.println("j= " +d2.j);

System.out.println("main ends");

}

static

{

System.out.println("SIB");

i=20;

}

{

System.out.println("IIB");

j=30;

}

}



**Non static members :**

The non static members are declared without using static keyword. The non static members are also known as **object members** or **instance members.**

1. The non static members are loaded into memory while creating an **object of the class**
2. The non static members can be accessed only by using **reference variable**
3. Based on the number of objects created ,copy of non static members are loaded into the memory
4. Java provides a separate block to initialize non static members. That block is known as **instance initialization block(IIB)**
5. The instance initialization block gets executed whenever an object is created
6. A program can have multiple instance initialization block and order of execution is always sequential(**Top to Bottom**)
7. A program can have both static initialization block(SIB) & instance initialization block(IIB). In such case static initialization block gets executed first & then for every object created ,non static initialization block will get executed

multiple class in java file

…………..

………….

…………….

Class1

{

……………..

……………….

}

Class2

{

……………..

……………….

}

Class3

{

……………..

……………….

}

Class1.class

……………….

………………..

…………………

Class2.class

………………..

……………

……………..

Class3.java

Class3.class

* If a class file has 1 public class file, that class name should be used as filename.java
* In the above class any class name can be used as classname.java. First, compiler checks the class by the same name as filename and compiles it and if there are other classes present it will also be compiled. For each class in .java file it will create separate class file.

class A

{

static

{

System.out.println("SIB of A class");

}

{

System.out.println("IIB of A class");

}

}

class Demo44

{

static

{

System.out.println("SIB of Demo44 class");

}

public static void main(String[] args)

{

System.out.println("main starts");

A a1=new A();

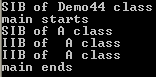
A a2=new A();

System.out.println("main ends");

}

}

o/p



**Note:**

1.static block gets executed only once when the class is loaded.

2.non static block get executed whenever an object of that class is created.

3.when an object is created all the non static member gets loaded

class B

{

int i=200;

int j=300;

int getIVal()

{

System.out.println("Entering getIVal()");

return i;

}

int getJVal()

{

System.out.println("Entering getJVal");

return j;

}

}

class Demo45

{

public static void main(String[] args)

{

System.out.println("main starts");

B b1=new B();

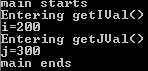
System.out.println(“i=” + b1.getIVal());

System.out.println(“j=” + b1.getJVal());

System.out.println("main Ends.....");

}

}



B

main

b1

Java

Also getIVal, getJVal

Stack Heap

Constructors

B b1=new B();

Constructor( This helps to construct)

* **new** operator calls the constructor.
* It constructs the object of the class mentioned. You cannot create an object without a class (new is a keyword).
* Every class you need a constructor otherwise you cannot create object.
* when there is no constructor in class, compiler will write the constructor. The constructor created by java compiler is called **default constructor**.
* The name of the constructor & class name should be same

C c1=New C();

**Constructors:**

* Used to create objects
* Used to initialize non static variables

class C

{

int i;

int j;

C()

{

i=10;

j=20;

}

}

class Demo46

{

public static void main(String[] args)

{

System.out.println("main starts");

C c1=new C();

System.out.println("i="+c1.i);

System.out.println("j="+c1.j);

System.out.println("main ends");

}

}

**o/p**



**Note:** **class or classes in a java file**

* A java file can contain any number of definition blocks.
* If one of the classes is public then filename should be the name of the public class. If the file contains any number of non public classes any class name can be used to name the file
* A java file cannot have more than 1 public class.
* Whenever a java file which contains multiple classes is compiled separate class file will be created for each class.

**Constructors:**

* **Constructors are definition blocks in java which is used to build or construct an object of the class.**
* The Constructor name should be same as class name
* Every class in java should contain at least one constructor. If not, the compiler writes constructor in the class which is known as default constructor.
* First **static**, then **IIB** & then **Constructor** part(actually constructor call the IIBs, then completes itself)

class A

{

static

{

System.out.println("SIB of class A");

}

A()

{

System.out.println ("Constructor of class A");

}

{

System.out.println("IIB of class A");

}

}

class Demo47

{

public static void main(String[] args)

{

System.out.println("main starts");

A a1=new A();

System.out.println("main ends");

}

}



class B

{

int i;

B(int a)

{

System.out.println("B constructor");

i=a;

}

}

class Demo48

{

public static void main(String[] args)

{

System.out.println("main starts");

B b1=new B(30);

System.out.println("i="+b1.i);

System.out.println("..............");

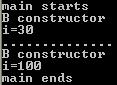
B b2=new B(100);

System.out.println("i="+b2.i);

System.out.println("main ends");

}

}



B

B

main

b1 b2

Java

Stack Heap

**Note:**

Whenever you write any Constructor, the compiler will not create any constructor. If no constructor is created by user then compiler creates a default constructor.

**Ex:** If we have created a constructor with single argument then we should use that constructor with single argument to construct the object. If we want constructor with no argument also ,like what compiler does ,then we should only create a no argument constructor.

Below is the example

class C

{

int i;

C(int a)

{

i=a;

}

C()

{

i=234;

}

}

class Demo49

{

public static void main(String args[])

{

System.out.println("main method starts");

C c1=new C();

System.out.println(c1.i);

C c2=new C(100);

System.out.println(c2.i);

System.out.println("main method ends");

}

}

o/p



**//Demo for 2 single arguments constructor like int and long**

**//when you pass whole number it is taken as int, to make it specific as long then we should use**

**‘l’ along with the number like 200l**

class C{

int i;

long l;

C(int a)

{

i=a;

}

C()

{

i=234;

}

C(long b)

{

l=b;

}

}

class Demo49{

public static void main(String args[])

{

C c1=new C();

System.out.println(c1.i);

C c2=new C(100);

System.out.println(c2.i);

C c3=new C(200l);

System.out.println(c3.l);

}

}



Inheritance

Class A

{

**int** i;

**int** j;

**void** print(){

System.out.println("i="+i);

System.out.println("j="+j);

}

}

**class** B **extends** A

{

**int** a;

**int** b;

**void** disp(){

System.***out***.println("a="+a);

System.***out***.println("b="+b);

}

}

**class** Demo50

{

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

{

System.out.println("main method starts....");

System.out.println("using reference variable of classA");

A a1=**new** A(); // USUAL OBJECT CREATION AND USING VARIABLES

a1.print();

a1.i=100;

a1.j=200;

a1.print();

System.out.println("..............");

System.out.println("using reference variable of classB");

B b1=**new** B();

b1.disp();

b1.print();

b1.i=1000;

b1.j=2000;

b1.a=1500;

b1.b=2500;

b1.print(); //INHERITANCE

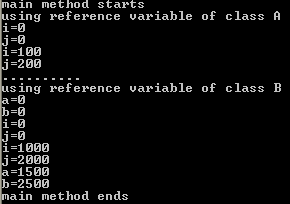
b1.disp();

System.out.println("main method ends...");

}

}

}

****

class C

{

int i;

int j;

C(int i,int j)

{

**this.i=i;**

**this.j=j;**

}

void print(){

System.out.println("i="+i);

System.out.println("j="+j);

}

}

class D extends C

{

int x;

int y;

D(int x,int y)

{

**super(x,y);**

**this.x=x;**

**this.y=y;**

}

void disp(){

System.out.println("x="+x);

System.out.println("y="+y);

}

}

class E extends D

{

E(int a,int b)

{

super(a,b);

}

}

class Demo51

{

public static void main(String[] args)

{

System.out.println("main method starts....");

E e1=new E(10,20);

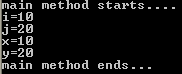
e1.print();

e1.disp();

System.out.println("main method ends...");

}

}



**Inheritance**: Inheriting the members of parent class to the child class is known as inheritance. The parent class is called a super class where as the child class is known as **subclass**.

* All the non static members of a super class are inherited to the object of sub class
* The inheritance can be achieved by using keyword **extends.** The subclass should extend the superclass
* With the reference of subclass object we can access both the inherited member as well as sub class members
* Multilevel inheritance are supported in java
* Multiple inheritance are not allowed in java through classes
* Every class we create in java will be sub class to the Object class. **Object class** is the super most class in java. If any class is not extending super class by default will always extend Object class
* **While inheriting object from super class, chain of constructors will be involved. The constructor of a sub class calls constructor of the super class. The constructor of the super class in turn calls constructor of its super class. This is known as chain of constructors.**
* **The java compiler makes an implicit call to the super class constructor by using default constructor of super class.**
* **If default constructors are not available then the overloaded constructor should be called explicitly using super statement.**
* The super statement should be always the first statement in the constructor
* Using super statement we can call any of the super class constructor.
* Whenever the java compiler makes a call to the super class, it uses super statement implicitly.

A

i :int

j: int

print():void

A

B Inherited members

B

a:int

b: int

disp():void

Defined

members

Java supports two types of relationship

1. **IS A relationship**
2. **HAS A relationship**

* The IS A relationship defines the inheritance of a superclass to the subclass
* where HAS A relationship defines composition of a class

SNAKE

DOG

(Head, Legs, Tail)

CAT

(Head, Legs, Tail)

HUMAN

(Head, Legs, Hands)

REPTILE

MAMMAL

ANIMAL

**Inheritance**

Dog HAS A head √

Dog HAS A tail √

Dog HAS A legs √

Human HAS A tail ×

Snake IS A Mammal ×

Snake IS A Animal √

Mammal IS A Animal √

Cat IS A Mammal √

Cat IS A Animal √

**Assignment:**

Create Student class with following attributes.

Student\_ID, Student\_Name, Student\_PhoneNumber, Student\_MailId, Student\_Address, Degree & YOP (year of passing).

* Class should allow to create a student with

1. Student\_Id ,Student\_Name & Student\_PhoneNumber
2. Student\_Id,Student\_Name & Student\_MailId

* Class should have methods to print every attribute values
* The student class should provide an option to modify phoneno, MailId & Address
* Provide method to print details of the student

1. Student Class
2. Attributes-Id, StudentName, StudentNumber,StudentMailId,StudentAddress,Degree & YOP
3. Constructors
4. StudentId,StudentName & StudentPhoneNumber
5. StudentId,StudentName & StudentMailId
6. Method to get each & every attribute values
7. Modify StudentPhoneNo,MailId

**class** Student {

// attributes

**private** **int** studentId;

**private** String studentName = **null**;

**private** **long** studentPhoneNumber;

**private** String studentAddress = **null**;

**private** String studentMailId = **null**;

**private** String studentDegree = **null**;

**private** **int** YOP;

// Constructors

Student(**int** studentId, String studentName, **long** StudentPhoneNumber) {

**this**.studentId = studentId;

**this**.studentName = studentName;

**this**.studentPhoneNumber = studentPhoneNumber;

}

Student(**int** studentId, String studentName, String StudentMailId) {

**this**.studentId = studentId;

**this**.studentName = studentName;

**this**.studentMailId = studentMailId;

}

**int** getStudentId() {

**return** studentId;

}

String getStudentName() {

**return** studentName;

}

**long** getStudentPhoneNumber() {

**return** studentPhoneNumber;

}

String getStudentMailId() {

**return** studentMailId;

}

String getStudentAddress() {

**return** studentAddress;

}

String getStudentDegree() {

**return** studentDegree;

}

**int** getStudentYOP() {

**return** YOP;

}

**void** setStudentPhoneNumber(**long** studentPhoneNumber) {

**this**.studentPhoneNumber = studentPhoneNumber;

}

**void** setStudentMailId(String studentMailId) {

**this**.studentMailId = studentMailId;

}

**void** setStudentAddress(String studentAddress) {

**this**.studentAddress = studentAddress;

}

**void** printStudentDetails() {

System.***out***.println("Student Id:" + studentId);

System.***out***.println("Student Name:" + studentName);

System.***out***.println("Student PhoneNo:" + studentPhoneNumber);

System.***out***.println("Student MailId:" + studentMailId);

System.***out***.println("Student Address:" + studentAddress);

System.***out***.println("Student Degree:" + studentDegree);

System.***out***.println("Student YOP:" + YOP);

}

}

**class** CreateStudent {

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

System.***out***.println("Creation of student starts...");

Student st1 = **new** Student(101, "Kishore", 90000000);

System.***out***.println(st1.getStudentName());

st1.setStudentPhoneNumber(80000000);

System.***out***.println(st1.getStudentPhoneNumber());

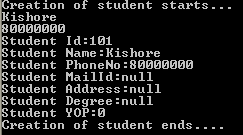
st1.printStudentDetails();

System.***out***.println("Creation of student ends....");

}

}

o/p



**sellPoint order management system**

Develop a class which is able to create a customer

SellPoint

Customer

Order

Customer class will have

CustomerId

CustomerName

CustomerCellPhoneNo

CustomerOfficeNo

CustomerAddress

The address should have Street,City,State & Customer should be able to create by using Country,Id,Name. The class should provide a method to print phone numbers & to print the address . Customer should be able to update change in phonenumbers as well as address . Whenever we need details of the Customer we should be able to get the details.

order Class OrderNumber

Quantity Availability

Quantity required

getQuantityAvailable()

getOrderNumber()

Boolean CreateOrder(int QuantityRequired)

Packages

A *package* is a grouping of related classes /interfaces/enumerations or annotation types providing access protection and name space management.

This helps in

* + Grouping of related types like classes and/or interfaces
  + avoiding name conflicts with classes created in same or other projects in the same company
  + You can allow types within the package to have unrestricted access to one another yet still restrict access for types outside the package.

com.qspiders.pack1 com.qspiders.pack2

B

A B

package com.qspiders.pack1;

public class A

{

int i;

public void print()

{

System.out.println("i="+i);

}

public static void main(String[] args)

{

System.out.println("class A of package com.qspiders.pack1");

}

}

For compilation



For execution



//separate java file B.java

package com.qspiders.pack1;

public class B

{

public static void main(String[] args)

{

System.out.println("Class B of package com.qspiders.pack1");

A a1=new A();

a1.print();

B b1=new B();

}

}

o/p



**Access Specifier(Visibility)**

1. private-Cannot access private members outside the class

class cannot be declared private

1. default -class can be public or default
2. protected
3. public

package com.qspiders.app1;

**class** A {

**private** **int** i = 10;

**void** print() {

System.***out***.println("i=" + i);

}

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

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

A a1 = **new** A();

a1.print();

a1.i = 20;

a1.print();

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

}

}

//separate file Run1.java

package com.qspiders.app1;

**class** Run1{

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

{

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

A a1=**new** A();

a1.print();

a1.i=100;

a1.print();

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

}

}



package com.qspiders.app1;

**public** **class** B {

**int** i = 100;

**private** B() {

}

**public** **void** print() {

System.***out***.println("i=" + i);

}

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

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

B b1 = **new** B();

b1.print();

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

}

}



package com.qspiders.app1;

**class** Run2

{

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

{

System.***out***.println("main method starts");

B b1=**new** B();

b1.print();// can’t create class as constructor is private

System.***out***.println("main method ends...");

}

}



While developing any application, classes related to similar tasks/operators are stored in a package. The java packages consists of classes which can be used in other packages.

* The java package naming is always done with the company domain name(URL)
* In one package duplicate classes are not allowed whereas same class name can exist in multiple packages.
* It is good practice to address the classname with the help of fully qualified name. Fully qualified classname consists of package name & the classname.
* A class in one package can access class from other package by using either fully qualified classname or import statement.

Access Specifiers :

The access specifiers in java indicates the visibility of the members of

the class. Java supports following 4 specifiers

1. private
2. default
3. protected
4. public

**Private:**

* Private members has a visibility with in the same class
* Private members cannot get access outside the class or outside the package
* Private can be used either for variable declaration/method declaration / for constructor
* If constructor is declared as private an instance of the class cannot be created using private constructor in another class.
* A class cannot be declared as private(java does not allow)

com.qspiders.pack1 com.qspiders.pack2

Run2

A Run1

i

print

* If class is defined as default then these will have visibility inside package
* If we want to access classes in another package we should use public class

package com.qspiders.pack1;

**class** C {

**int** i = 100;

**void** print() {

System.***out***.println("i=" + i);

}

}

**class** Run1 {

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

System.***out***.println("main method starts");

C c1 = **new** C();

c1.print();

c1.i = 200;

c1.print();

System.***out***.println("main method ends...");

}

}

//if writing both classes in same java file ,Save the above code with public classname .java ex: C.java (if any of the class is public)

//else if writing both in separate java files then write the “ package com.qspiders.pack1;” in the first line

**package** com.qspiders.pack2;

**import** com.qspiders.pack1.C;

**class** Run2

{

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

{

System.***out***.println("Main method starts");

C c1=**new** C();

c1.print(); //error,default member can't be accessed

c1.i=200; //error, default member can't be accessed

c1.print(); // error, default member can't be accessed

System.***out***.println("Main method ends...");

}

}

**Note:**

1. *while saving save with the class name containing public class.*
2. *You cannot have two(2) public class in a single java file.*

Constructor access depends on the class access specifier. If the class is public, constructor that java creates will be public. If we write our own constructor then we have to give public access to use the class outside the package.

package com.qspiders.pack1;

public class D

{

public D()

{

System.out.println("running constructor");

}

public int j=100;

public void print()

{

System.out.println("j="+j);

}

}

Note:Save it as public classname.java

package com.qspiders.pack2;

import com.qspiders.pack1.D;

class Run3

{

public static void main(String[] args)

{

System.out.println("main method starts");

D d1=new D();

d1.print();

d1.j=200;

d1.print();

System.out.println("main method ends...");

}

}



**Default Members:**

* default members has a visibility outside the class but within the package. But default members cannot access outside the package.
* default class and default members are declared without a keyword
* The public members has a visibility within the package as well as outside the package(another package)
* public members can be accessed from any where
* Whenever compiler finds default constructor(created by compiler) , the access specifier will always will go with the class access specifier
* If the class is public and access specifier is default(for constructor) we cannot create an instance of the class outside the package.

public class constructor (default access specifier)

public class D

{

D()

{

System.out.println("running constructor");

}

public int j=100;

}

package com.qspiders.pack1;

public class F

{ public int j=0;

protected int i=100;

protected void print()

{

System.out.println("i="+i);

}

}

class Run3

{

public static void main(String[] args)

{

System.out.println("main method starts");

F f1=new F();

f1.print();

f1.i=200;

f1.print();

System.out.println("main method ends...");

}

}

// Save this as F.java

//Compile & run , the above program Run3.class will work because protected members can be accessed from Same /package.

package com.qspiders.pack2;

import com.qspiders.pack1.F;

class Run4

{

public static void main(String[] args)

{

System.out.println("main method starts");

F f1=new F();

f1.print(); // protected members can't be accessed

f1.i=200; // without inheritance from other package

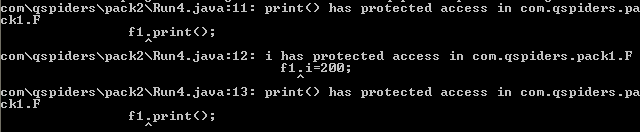
f1.print();

System.out.println("main method ends");

}

}





package com.qspiders.pack2;

import com.qspiders.pack1.F;

class Run5 extends F

{

public static void main(String[] args)

{

System.out.println("main method starts");

Run5 r1=new Run5();

r1.print(); // print() and i are protected members

r1.i=200; // class C, can be accessed only through

r1.print(); //inheritance, becomes private to this class

System.out.println("main method ends");

}

}



**Protected**

* The protected members have the visibility outside the class and also outside the package. But protected members can be accessed by other package only through inheritance.
* After inheriting protected members it behaves as a private member to the class

package com.qspiders.pack1;

class A

{

void test(){

System.out.println("test() of class A");

}

String test(String str){

System.out.println("test() of class A");

return "done";

}

String test(String str,int i){

System.out.println("test() of class A");

return "done";

}

}

Note: The above program can be compiled, but cannot be run as it does not have main() method

**Method overloading:**

Developing multiple methods with same method name but different method signatures is known as **method overloading**. The signature should change either in terms of number of arguments or its datatypes or return type of the method.

Imp:Method can be overloaded in the same class(usually) or in the sub class.

**Method overriding:**

Sub class providing a new implementation of already defined method in the super class.For overriding methods, there should be IS-A relationship b/w classes or inheritance.

Method overriding is applicable for non static methods.

because only non static methods are inherited.

**Rules for method overriding:**

1. method must have same name as in the super class
2. method must have same parameters in the sub class which is in the super class and in the same order.

Static methods are not overridden in the sub class but it is hidden.

Animal

speak()

Mammal

speak()

Cat

speak()

Dog

speak()

Tiger

speak()

**class** Animal {

**void** speak() {

System.***out***.println("Animal speaking");

}

}

**class** Mammal **extends** Animal {

**void** speak() {

System.***out***.println("Mammal speaking");

}

}

**class** Dog **extends** Mammal {

**void** speak() {

System.***out***.println("Bowbow");

}

}

**class** Run4 {

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

Mammal m1 = **new** Mammal();

m1.speak();

Animal a1 = **new** Animal();

a1.speak();

Dog d1 = **new** Dog();

d1.speak();

}

}



**class** A {

**void** test() {

System.***out***.println("test() method of class A");

}

}

**class** B **extends** A {

**void** test() {

System.***out***.println("test() method of class B");

}

}

**class** C **extends** B {

**void** test() {

System.***out***.println("test() method of class C");

}

}

**class** Run1 {

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

System.***out***.println("main method starts");

B b1 = **new** B();

b1.test();

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

A a1 = **new** A();

a1.test();

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

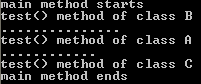
C c1 = **new** C();

c1.test();

System.***out***.println("main method ends");

}

}



**class** MummyMD {

**void** CreateMD() {

System.***out***.println("Heat tava");

System.***out***.println("put oil on tava");

System.***out***.println("pour dosa batter on tava");

System.***out***.println("roll it over after 1 min");

System.***out***.println("put potato masala");

}

}

**class** MyMD **extends** MummyMD {

**void** CreateMD() {

**super**.CreateMD();

System.***out***.println("put chutney");

System.***out***.println("put ghee on dosa");

}

}

**class** Run7 {

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

MummyMD m1 = **new** MummyMD();

m1.CreateMD();

System.***out***.println("--------------");

MyMD m2 = **new** MyMD();

m2.CreateMD();

}

}

**Note:**

Above examples are for method overriding

Can we override static method?

No, static method cannot be overridden

Why can’t we override static method?

Static method is bound to the class and non static method is bound to the object.

**class** A {

**static** **void** speak() {

System.***out***.println("Bow Bow");

}

}

**public** **class** MethodDemo **extends** A {

**static** **void** speak() {

System.***out***.println("Bow Bow Bow");

}

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

MethodDemo.*speak*();

// if we want A class speak()

// then

A.*speak*();

}

}

**Method Overriding:**

* Developing a method with the same name & signature as in the Super class but different implementation is known as method overriding.
* In order to override any method the class should extend the super class. The super class method can be changed in the sub class by using method overriding.
* Constructors cannot be overridden since constructors cannot inherit to sub class.
* If you want the super class implementation of the method in the sub class and also want to add your own implementation then you can use super.methodname in the sub class method

**Class**

Abstract class

Concrete class

**abstract** **class** D {

**void** test1() {

// concrete method

}

**abstract** **void** test(); // abstract

}

**Note:**

* To declare an abstract method we need to use abstract keyword i.e. abstract method has no body & it is incomplete.
* If a class has abstract method then class should be declared abstract.
* abstract class does not mean it should compulsorily have abstract methods but opposite is not true.
* abstract class cannot be used to create objects

**abstract** **class** D {

**void** test1() {

System.***out***.println("test1() method of class D");

}

}

**class** F **extends** D {

**void** test2() {

System.***out***.println("test2() method of class F");

}

}

**class** Run7 {

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

F f1 = **new** F();

f1.test1();

f1.test2();

}

}

o/p



**abstract** **class** W {

**abstract** **void** test1();

**abstract** **void** test2();

**abstract** **void** test3();

}

**abstract** **class** X **extends** W {

**void** test1() {

System.***out***.println("test1() implemented in class X");

}

}

**abstract** **class** Y **extends** X {

**void** test2() {

System.***out***.println("test2() implemented in class Y");

}

}

**class** Z **extends** Y {

**void** test3() {

System.***out***.println("test3() implemented in class Z");

}

}

**class** Run2 {

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

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

Z z1 = **new** Z();

z1.test1();

z1.test2();

z1.test3();

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

}

}



**Abstract Class:**

A method without body is known as abstract method. The abstract method should be declared with the keyword ‘**abstract’**.

If a class contains an abstract method then that class is known as **abstract class**.

**Concrete Class:**

A method which contains both declaration & method body is known as concrete method or complete method. If a class contains **only concrete methods** then such class is known as **concrete class.**

**Notes on Abstract Class:**

* An abstract class can have any number of abstract method as well as concrete methods or only concrete method(s).
* If a class contains atleast 1(one)abstract method then that class should be declared as abstract.
* An abstract class cannot be instantiated.
* An abstract class can be extended to the sub class. Whenever an abstract class is extended to the sub class then the sub class should implement all methods or else the sub class should also be declared as abstract class.
* Whenever we have to use abstract methods the abstract methods should be completed/overridden in the sub class.

**Note:**

When we add abstract method to the super class then all the sub classes are affected.

**Assignment**

Animal class-> Mammal class->Dog class, cat class & tiger class. Create a separate class with main method and create/call objects.

**Interface**

interface Interfacename

{

……………………………….

……………………………… //Interface body

……………………………….

}

Java supports 3 types of definition blocks

1. class Type
2. interface Type
3. enum Type

**Notes on Interface:**

* An interface type starts with keyword interface
* A java file can contain only interface definition block
* After compilation of interface definition block compiler generates .class file for the interface
* Java file can have both class definition block and interface definition block
* Inside Interface definition block concrete methods are not allowed
* The interfaces should contain only abstract methods
* By default the methods in interface are **abstract and public**
* Other than public access specifier we cannot use other access specifier
* In interface we can declare a variable but it should be initialized at the declaration time. All variables in an interface must be public ,static and final i.e. it can declare only constants not instance variables.
* A method **cannot be declared as static** in interface where as a variable can be declared as static (By default variables are public ,static and final)
* A class **cannot extend more than 1 class** (java does not support multiple inheritance) but a **class can implement more than 1 interface**
* We **cannot instantiate an interface**
* An interface can extend one or more other interfaces
* An interface cannot extend anything other than interface
* An interface cannot implement another interface or class

**interface** IDemo1 {

**void** test1();

}

**class** A **implements** IDemo1 {

**public** **void** test1() {

System.***out***.println("test1() implemented in class A");

}

}

**class** Run8 {

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

A a1 = **new** A();

a1.test1();

}

}

o/p



<<interface IDemo1>>

test1();

<<classA>>

test1()

Interface

We cannot instantiate interface.

IDemo1 i=new IDemo1();

**interface** IDemo2 {

**void** test1();

**void** test2();

}

**abstract** **class** B **implements** IDemo2 {

**public** **void** test1() {

System.***out***.println("test1() implemented in class B");

}

}

**class** C **extends** B {

**public** **void** test2() {

System.***out***.println("test2() implemented in class C");

}

}

**class** Run9 {

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

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

C c1 = **new** C();

c1.test1();

c1.test2();

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

}

}

o/p



**interface** IDemo4 {

**void** test1();

}

**interface** IDemo5 {

**void** test2();

}

**class** B **implements** IDemo4 {

**public** **void** test1() {

System.***out***.println("test1() implemented in class B");

}

}

**class** C **extends** B **implements** IDemo5 {

**public** **void** test2() {

System.***out***.println("test2() implemented in class C");

}

}

**class** Run11 {

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

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

C c1 = **new** C();

c1.test1();

c1.test2();

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

}

}



**Note: Extend first then implement**

**interface** IDemo5 {

**void** test1();

}

**interface** IDemo6 {

**void** test2();

}

**class** F **implements** IDemo6, IDemo5 {

**public** **void** test1() {

System.***out***.println("test1() implemented in Class F");

}

**public** **void** test2() {

System.***out***.println("test2() implemented in Class F");

}

}

**class** Run12 {

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

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

F f1 = **new** F();

f1.test1();

f1.test2();

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

}

}



**interface** IDemo5 {

**void** test1();

}

**interface** IDemo6 {

**void** test2();

}

**abstract** **class** F **implements** IDemo5, IDemo6 {

**public** **void** test1() {

System.***out***.println("test1() implemented in class F");

}

}

**class** G **extends** F {

**public** **void** test2() {

System.***out***.println("test2() implemented in class G");

}

}

**class** Run12 {

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

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

G g1 = **new** G();

g1.test1();

g1.test2();

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

}

}



**Difference between abstract Class & Interface**

|  |  |
| --- | --- |
| **Abstract** | **Interface** |
| Should be declared with keyword “abstract” | Should be declared with keyword “interface” |
| Abstract class can contain both concrete and abstract methods | Interface should contain only abstract methods |
| In Abstract class a method should be explicitly declared as abstract. | All methods in interface are abstract by default in interface. |
| We can use all access specifiers | Only public access specifier is allowed |
| It can contain both static and non static methods. | It cannot contain static methods |
| A class should **extend** an abstract class | A class should **implement** an interface |
| A class cannot extend more than one class. Bcz java doesn’t support multiple inheritance | A class can extend more than one interface. |
| In abstract class we can declare variable and initialize later. | But in interface all variables are public,static and final. So we need to initialize while declaration itself. |
| Values of an variable can be modified later. | Values cannot be modified because it is final. |
| Since abstract class support concrete methods, adding them wont affect any classes that extends it. | Since all the methods in interface are abstract by default it affects all the dependent implementation classes. |

* A class implement multiple interfaces and also class can extend another class and implement an interface

Ex:

Motor

getSpeed();

Battery Motor

startByBattery();

Solar motor

startByMotor();

Hybrid

1st way

//Base Interface

**interface** IMotor {

**int** getSpeed();

}

//Sub Interface

**interface** IBatteryMotor **extends** IMotor {

**void** startByBattery();

}

//Sub Interface

**interface** ISolarMotor **extends** IMotor {

**void** startBySolar();

}

**class** BatteryMotor **implements** IBatteryMotor {

**public** **void** startByBattery() {

System.***out***.println("Starting by battery");

}

**public** **int** getSpeed() {

**return** 1400;

}

}

**class** SolarMotor **implements** ISolarMotor {

**public** **void** startBySolar() {

System.***out***.println("Starting by solar");

}

**public** **int** getSpeed() {

**return** 1200;

}

}

//Multiple Inheritance by using interface

**class** HybridMotor **implements** IBatteryMotor, ISolarMotor {

**public** **void** startBySolar() {

System.***out***.println("Starting by solar");

}

**public** **void** startByBattery() {

System.***out***.println("Starting by battery");

}

**public** **int** getSpeed() {

**return** 1500;

}

}

**public** **class** Sample {

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

BatteryMotor b1 = **new** BatteryMotor();

b1.startByBattery();

**int** speed = b1.getSpeed();

System.***out***.println(speed);

SolarMotor b2 = **new** SolarMotor();

b2.startBySolar();

**int** speed1 = b2.getSpeed();

System.***out***.println(speed1);

HybridMotor h1 = **new** HybridMotor();

h1.startByBattery();

h1.startBySolar();

**int** speed2 = h1.getSpeed();

System.***out***.println(speed2);

}

}

**2nd way**

package com.qspiders.pack1;

**interface** iBatteryMotor {

**void** startByBatteryMotor();

}

**interface** iSolarMotor {

**void** startBySolarMotor();

}

/\*

\* Alternate way interface iHBMotor extends iBatteryMotor,iSolarMotor {

\*

\* }

\*

\* class HBMotor implements iHBMotor

\*/

**class** HBMotor **implements** iBatteryMotor, iSolarMotor {

**public** **void** startByBatteryMotor() {

System.***out***.println("starting by batterry");

}

**public** **void** startBySolarMotor() {

System.***out***.println("starting by solar ");

}

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

HBMotor h = **new** HBMotor();

h.startByBatteryMotor();

h.startBySolarMotor();

}

}

**Difference between Method & Constructors**

|  |  |
| --- | --- |
| **Method** | **Constructors** |
| Methods are members of the class which can return value. | Constructors are members of the class used to build the object of the class. |
| Method names are identifiers which describes the behavior of the class. | Constructor name should be same as classname. |
| Methods have return types | Constructors doesn’t have return types |
| In methods return statements are used to return value. | Return statements are not used in constructors. |
| Methods can be declared both as static or nonstatic | Constructors cannot be declared as static or nonstatic |
| Non static Methods are involved in inheritance. | Constructors are not involved in inheritance |
| Methods can be overloaded and overridden | Constructors can be overloaded but cannot be overridden. |
| Methods can be declared as abstract. | Constructors cannot be declared as abstract. |
| Methods can be declared as final. | Constructors cannot be declared as final. |
| Recursive calls to the methods are allowed. | Recursive calls to the constructors are not allowed. |
| Methods are involved in polymorphism. | Constructors are not involved in polymorphism. |

Note: When ever you declare **final variable** it should be initialized in the same line

**class** Run14

{

**final** **static** **int** ***i***=10;

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

{

System.***out***.println("main method starts....");

System.***out***.println("i="+***i***);

Run14.***i***=100; // Cannot reinitialize final variable

System.***out***.println("main method ends...");

}

}



**class** B {

**final** **void** test1() {

System.***out***.println("test1() of class B");

}

}

**class** C **extends** B {

**void** test1() {

System.***out***.println("test1() of class C");

}

}

**class** Run15

{

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

{

C c1=**new** C();

c1.test1();

}

}



**final class -example**

**final** **class** B {

**void** test1() {

System.***out***.println("test1() of class B");

}

}

**class** C **extends** B {

**void** test2() {

System.***out***.println("test1() of class C");

}

}

**class** Run15 {

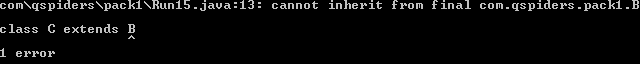
**public** **static** **void** main(String args[]) {

C c1 = **new** C();

c1.test1();

}

}



Note: final class cannot be inherited or extended

**Interview Question:**

**Diff b/w final ,finally & finalize**

**final keyword:**

* If a variable is declared as final such variables cannot be modified.
* A method can be declared as final but cannot be overridden in the subclass
* A class can be declared as final. final class cannot be inherited to the subclass
* Constructors cannot be declared as final
* All the variables inside an interface by default is a final
* The final variable should be initialized at the time of declaration itself

**Wrapper Class**

To convert primitive data type into equivalent object 🡪 Byte, Short, Integer, Long, Float, Double, Character, Boolean

1. Converting primitive data type into an object is called as **auto boxing**. (Compiler does it)
2. Converting an object(which is converted from auto boxing) to primitive data type is called as **un boxing.**(Developer does it)

In both cases we use wrapper class. Every data type respective wrapper class is given by JDK

Every wrapper classes will have overloaded constructor. They do not use default constructor in wrapper class.

**Object class is Super most class of wrapper class.**

Number

Integer

intObj

Java

Integer

In every wrapper class **tostring()** method is overridden in such a way that it prints the value rather than address

Note:

* Generally whenever a reference variable is printed the address of the object will be printed. The address is usually represented with fully qualified class name-fullyqualifiedclassname@hexadecimal address of the object.
* Whenever a reference variable of any wrapper class is used or printed then prints the primitive data of the wrapper class instead of address because in every wrapper class **tostring() method**of the object class has been overridden to display or return the primitive data instead of address.

**public** **class** WrapperDemo1 {

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

System.***out***.println("main method starts");

**int** i = 10;

System.***out***.println("i=" + i);

Integer intObj = **new** Integer(i);

// boxing operation

System.***out***.println(intObj);

System.***out***.println("----------");

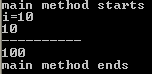
Integer intObj2 = **new** Integer(100);

System.***out***.println(intObj2);

System.***out***.println("main method ends");

}

}



//Integer intObj2=100;√

//Integer intObj=i; √ (Auto Boxing) JDK 1.5 onwards

Object

Number

Short

Integer

Byte

Long

Float

Double

**Notes on Wrapper classes:**

* In java.lang
* No single constructor (only Character), overloaded constructor
* tostring() method is overridden
* equals() method is overridden
* hashCode() method is overridden

Every wrapper class inherit from super class Number(abstract class). It is inherited from Object class as we cannot create instance of Number class as it is abstract.

The Number related wrapper classes are extending super class which is known as **Number**. The Number class methods are overridden in every wrapper class which deals with Number.

**public** **class** WrapperDemo2 {

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

System.***out***.println("main method starts");

**int** i=100;

Integer intObj=**new** Integer(i);// boxing operation

System.***out***.println(intObj);

**int** j=intObj.intValue(); //unboxing operation

System.***out***.println(j);

System.***out***.println("main method ends");

}

}



**public** **class** WrapperDemo3 {

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

System.***out***.println("main method starts");

**int** i=100;

Integer intObj=**new** Integer(i); // boxing operation

System.***out***.println(intObj);

**int** j=intObj.intValue(); //unboxing operation

System.***out***.println(j);

**double** d=intObj.doubleValue();

System.***out***.println(d);

**double** d1=1.2;

Double doubleObj=d1;

**int** k=doubleObj.intValue(); //unboxing & narrowing

System.***out***.println(k);

**int** m=(**int**)intObj.doubleValue();//unboxing &

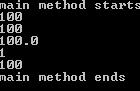
//explicit narrowing

System.***out***.println(m);

System.***out***.println("main method ends");

}

}



\*\*\*\*\*\*\*\*\*\*\*\*\*\*Important-Preference of execution of arguments\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**class** B {

**void** test(**double** d) {

System.***out***.println("double argument method");

}

**void** test(Integer intObj) {

System.***out***.println("Integer argument method");

}

**void** test(Number num) {

System.***out***.println("Number argument method");

}

**void** test(Object obj) {

System.***out***.println("Object argument method");

}

**void** test(**int** d) {

System.***out***.println("integer argument method");

}

}

**public** **class** WrapperDemo4 {

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

System.***out***.println("main method Starts");

B b1 = **new** B();

b1.test(10);

System.***out***.println("main method ends");

}

}



Preference of execution->Same type, Auto widening, Wrapper classes, Number class and Object class

**Interview Question: When does stack unwinding happen?**

While executing any program the methods of the classes will be loaded into the stack for the execution purpose. Stack will be loaded with methods in the order the way it is called. If the last entered method generates any exception then java looks for the handler in the same method. If no handler is found in the method then the exception propagates to the called method. In the called method if the exception is not handled , the exception will propagate to its called method finally the exception reached the java.

It is the responsibility of Java to handle the exceptions. Java does not know to handle the exception then it forcibly removes all the method from the stack & terminates the program execution. This is known as **stack unwinding.**

**Object Class**

All the methods are/can be overridden

* toString()
* equals()
* hashCode() this will be done by default

extends java.Object

public class A {

A()

{

super(); //Constructor will be declared on our behalf

}

**}**

**class** A {

**int** i;

**public** String toString() {

**return** "i=" + i;

}

}

**public** **class** Run1 {

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

A a1 = **new** A();

System.***out***.println(a1);

System.***out***.println("-------");

System.***out***.println(a1.toString());

}

}



**class** B {

**int** i;

**int** j;

**public** String toString() {

**return** "i=" + i + ",j=" + j;

}

}

**public** **class** Run2 {

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

B b1 = **new** B();

System.***out***.println(b1);

System.***out***.println("main method ends");

}

}



**class** C {

}

**public** **class** Run3 {

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

C c1 = **new** C();

C c2 = **new** C();

C c3 = **new** C();

System.***out***.println(c1);

System.***out***.println(c2);

System.***out***.println(c3);

System.***out***.println(c3.equals(c1));

System.***out***.println(c2.equals(c3));

c1 = c3;

System.***out***.println(c1);

System.***out***.println(c2);

System.***out***.println(c3);

// equals method not overridden

// hence will compare address

System.***out***.println(c1.equals(c3));

System.***out***.println(c1 == c3);// comparing address

// For understanding

String s1 = **new** String("Java");

String s2 = **new** String("Java");

System.***out***.println(s1 == s2);

System.***out***.println(s1.equals(s2));

}

}

o/p

com.qspiders.ObjectClassDemo.C@1e63e3d

com.qspiders.ObjectClassDemo.C@1004901

com.qspiders.ObjectClassDemo.C@1b90b39

false

false

com.qspiders.ObjectClassDemo.C@1b90b39

com.qspiders.ObjectClassDemo.C@1004901 same address

com.qspiders.ObjectClassDemo.C@1b90b39

true

true

false

true

**class** D {

**int** i;

D(**int** i) {

**this**.i = i;

}

**public** String toString() {

**return** "i=" + i;

}

**public** **boolean** equals(Object obj) {

**return** **this**.i == ((D) obj).i;

}

}

**public** **class** Run4 {

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

System.***out***.println("main method starts");

D d1 = **new** D(10);

System.***out***.println(d1);// i=10

System.***out***.println("------------");

D d2 = **new** D(20);

System.***out***.println(d2);// i=20

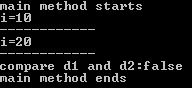
System.***out***.println("------------");

System.***out***.println("compare d1 and d2:" + d1.equals(d2));

System.***out***.println("main method ends");

}

}



**class** E {

**int** i;

E(**int** i) {

**this**.i = i;

}

**public** String toString() {

**return** "done";

}

**public** **int** hashCode() {

**return** i;

}

}

**public** **class** Run5 {

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

System.***out***.println("main method starts");

E e1 = **new** E(15);

System.***out***.println(e1);

System.***out***.println(e1.hashCode());

E e2 = **new** E(20);

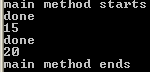
System.***out***.println(e2);

System.***out***.println(e2.hashCode());

System.***out***.println("main method ends");

}

}



**finalize() method**

This is a method provided by Java to run some code just before the object is deleted by the gargabe collector. This is a protected method of the Object class

and needs to extend the the Object class to override or use it.

**public** **class** FinalizeDemo **extends** Object {

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

FinalizeDemo fd = **new** FinalizeDemo();

**try** {

fd.finalize();

} **catch** (Throwable e) {

e.printStackTrace();

}

}

}

**Enumerations**

In simple terms an **enumeration**  is a list of named constants that define a new data type that has a fixed set of values.

An object of an enumeration type can hold only the values that are defined by the list.

**enum** Payment {

***CHEQUE***, ***CREDITCARD***, ***DEBITCARD***

}

**public** **class** Sample {

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

Payment pmt;

pmt = Payment.***CREDITCARD***;

System.***out***.println("value of pmt: " + pmt);

pmt = Payment.***CHEQUE***;

**switch** (pmt) {

**case** ***CHEQUE***:

System.***out***.println("Payment made by :" + pmt);

**break**;

**case** ***CREDITCARD***:

System.***out***.println("Payment made by :" + pmt);

**break**;

**case** ***DEBITCARD***:

System.***out***.println("Payment made by :" + pmt);

**break**;

}

}

}



**Note:**

* Java implements enumerations as class types.
* We don’t instantiate an **enum** using **new** , it act much like a class, i.e. you can give it Constructors,add instance variable and methods and even implement interfaces

**enum** Payment {

***CHEQUE***, ***CREDITCARD***, ***DEBITCARD***

}

**public** **class** EnumDemo2 {

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

Payment pmt;

pmt = Payment.***CREDITCARD***;

System.***out***.println("Printing all payment constants");

Payment pts[] = Payment.*values*();

**for** (Payment p : pts) {

System.***out***.println(p);

}

}

}



**enum** Payment {

***CHEQUE***(1000), ***CREDITCARD***(1500), ***DEBITCARD***(2000);

**private** **int** minPaymt;

Payment(**int** i) {

minPaymt = i;

}

**int** getMinPaymt() {

**return** minPaymt;

}

}

**public** **class** Sample {

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

Payment pmt;

pmt = Payment.***CREDITCARD***;

System.***out***.println("Minimum payment for cheque is " + Payment.***CHEQUE***.getMinPaymt());

System.***out***.println("Minimum payment for differen modes");

Payment pts[] = Payment.*values*();

**for** (Payment p : pts) {

System.***out***.println("For " + p + " is " + p.getMinPaymt());

}

}

}



**Note:**

1.enumeration can’t inherit another class(all enumerations automatically inherit java.lang.Enum class)

2.enumeration cannot be a superclass,i.e. enumeration can’t be extended

**OverLoaded Constructors:**

Similar to method overloading, Constructors can also be overloaded. Here number of parameters, type of parameters should be different.

**Note:**

Comparing to Method overloading, main difference is constructor do not return any value while method can.

**class** ConsEx {

**int** i, j;

ConsEx() {

System.***out***.println("Inside Constructor with no arguments");

i = 0;

}

ConsEx(**int** i) {

System.***out***.println("Inside Constructor with single argument(int) ");

**this**.i = i;

}

ConsEx(**double** d) {

System.***out***.println("Inside Constructor with single argument(double) ");

i = (**int**) d;

}

ConsEx(**int** i, **int** j) {

System.***out***.println("Inside Constructor with two arguments ");

**this**.i = i;

**this**.j = j;

}

}

**class** OverLoadedConsDemo {

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

ConsEx c1 = **new** ConsEx();

ConsEx c2 = **new** ConsEx(55);

ConsEx c3 = **new** ConsEx(32.6);

ConsEx c4 = **new** ConsEx(6, 7);

System.***out***.println("c1.i: " + c1.i);

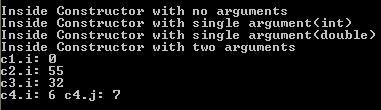
System.***out***.println("c2.i: " + c2.i);

System.***out***.println("c3.i: " + c3.i);

System.***out***.println("c4.i: " + c4.i + " c4.j: " + c4.j);

}

}



**File System in Java**

**public** **class** Sample {

**public** **static** **void** main(String[] args) **throws** IOException {

System.***out***.println("main method starts");

File file = **new** File("d:\\Sample");

**if** (!file.exists()) {

file.mkdir();

}

System.***out***.println(file.getAbsolutePath());

System.***out***.println(file.getName());

System.***out***.println(file.getParent());

System.***out***.println(file.isDirectory());

System.***out***.println(file.length());

File[] fils = file.listFiles();

System.***out***.println(fils.length);// unspecified for a dir

**for** (Object o : fils) {

System.***out***.println(o);

}

File file1 = **new** File("d:\\IEDriverServer\_Win32\_2.32.3.zip");

System.***out***.println(file1.length() / 1024);

// file1.delete();

File fil = **new** File("d:\\");

System.***out***.println(fil.getFreeSpace() / (1024 \* 1024 \* 1024));

System.***out***.println("main method ends");

}

}

**o/p**

main method starts

d:\Sample

Sample

d:\

**true**

0

4

d:\Sample\IEDriverServer\_Win32\_2.32.3.zip

d:\Sample\Log.txt

d:\Sample\LoginInfo.txt

d:\Sample\Sample.txt

790

10

main method ends

**Note:**

Object of type File is used to represent /create both File and Directory.

**File file=new File(“D:\\sample”);**

It always creates a File object, and then does one of the two things

1. If D:\\sample does not exist, **no actual file is created**

2. If D:\\sample does exist, the new File object refers to the existing file/directory

**File can be created in two ways:**

It is done in two ways

1)File file=new File(“D:\\sample.txt”);

file.createNewFile();

2)File file=new File(“D:\\sample.txt”);

PrintWriter pw=new PrintWriter(file);

Reading from the file.

**package** com.qspiders.fileHandling;

**import** java.io.\*;

**public** **class** Demo2 {

**public** **static** **void** main(String[] args) **throws** IOException{

System.*out*.println("main method starts");

File f1=**new** File("D:\\Sample.txt");

FileReader FRead=**new** FileReader(f1);

BufferedReader FBufRead=**new** BufferedReader(FRead);

String s;

**while**((s=FBufRead.readLine())!=**null**)

{

**if**(s.length()>0)// to skip a blank line

System.*out*.println(s);

}

System.*out*.println("main method ends");

}

}

Writing into a file using BufferedWriter.

**package** com.qspiders.fileHandling;

**import** java.io.\*;

**public** **class** Demo3 {

**public** **static** **void** main(String[] args) **throws** IOException{

System.*out*.println("main method starts");

File f1=**new** File("D:\\Sample.txt");

FileWriter FWriter=**new** FileWriter(f1);

BufferedWriter FBufWrite=**new** BufferedWriter(FWriter);

FBufWrite.newLine();

FBufWrite.write("Good Morning All" );

FBufWrite.newLine();

FBufWrite.write("Writing to a file");

FBufWrite.close();

System.*out*.println("main method ends");

}

}

**Note:**

* To write into a new file or to overwrite

FileWriter obj=**new** FileWriter(f1);

* To append at the end of the file

FileWriter obj=**new** FileWriter(f1,**true**);

Writing into a file using PrintWriter.

**package** com.qspiders.fileHandling;

**import** java.io.\*;

**public** **class** Demo4 {

**public** **static** **void** main(String[] args) **throws** IOException{

System.*out*.println("main method starts");

File f1=**new** File("D:\\Sample.txt");

FileWriter FWriter=**new** FileWriter(f1);

PrintWriter PWrite=**new** PrintWriter(FWriter);

PWrite.println("Writing thru printwriter");

PWrite.println("easy way");

PWrite.close();

System.*out*.println("main method ends");

}

}