**Declarations and Access Control - Part 1**

A java Program can contain any no. of classes but at most one class can be declared as public. "If there is a public class the name of the Program and name of the public class must be matched otherwise, we will get compile time error". If there is no public class then any name we gives for java source file.

**Import Statement:**

whenever we are using import statement it is not require using fully qualified names, we can use short names directly. This approach decreases length of the code and improves readability.

**Fully qualified name**: java.util.ArrayList l=new java.util.ArrayList();

**Using import:**

import java.util.ArrayList;

ArrayList l=new ArrayList();

Two types of import statements:

1) Explicit class import : import java.util.ArrayList; àRecommended in realtime projects

2) Implicit class import: import java.util.\*; à Not recommended in realtime projects

Note :

· In any java Program the following 2 packages are not require to import because these are available by default to every java Program.

1. java.lang package

2. default package(current working directory)

· "Import statement is totally compile time concept" if more no of imports are there then

more will be the compile time but there is "no change in execution time".

**Difference between C language #include and java language import.**

->In the case of C language #include all the header files will be loaded at the time of

include statement hence it follows static loading.

-> But in java import statement no “.class” will be loaded at the time of import statements

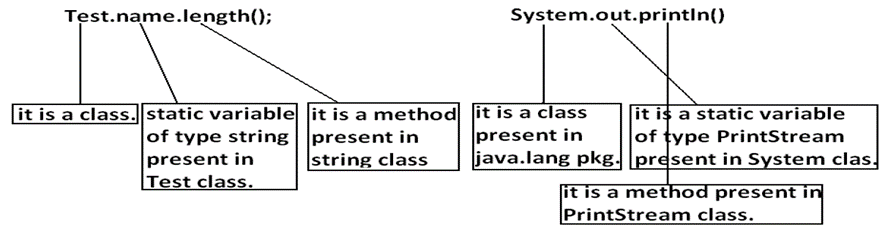
in the next lines of the code whenever we are using a particular class then only

corresponding “.class” file will be loaded. Hence it follows “dynamic loading” or “loadon

–demand” or “load-on-fly”.

**Explain about System.out.println();**

****

****

**Package Statement:**

It is an encapsulation mechanism to group related classes and interfaces into a single

module.

The main objectives of packages are:

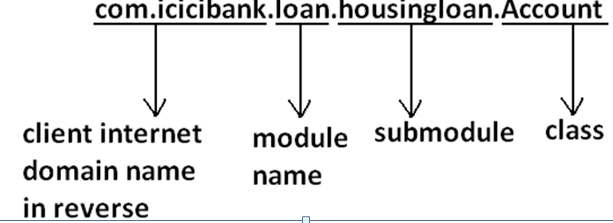
To resolve name conflicts.

To improve modularity of the application.

To provide security.

There is one universally accepted naming conversion for packages that is to use

internet domain name in reverse.



**How to use package statement:**

package com.durgajobs.itjobs;

class HydJobs

{

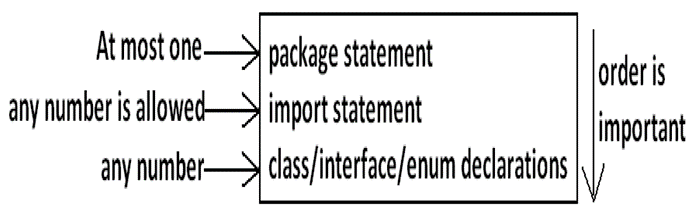
public static void main(String args[]){

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

**}**

**}**

**Order of import, package:**



**Class Modifiers:**

Whenever we are writing our own classes compulsory we have to provide some information about our class to the jvm. Like

1. Whether this class can be accessible from anywhere or not.

2. Whether child class creation is possible or not.

3. Whether object creation is possible or not etc.

We can specify this information by using the corresponding modifiers. The only applicable modifiers for Top Level classes are:

1. Public

2. Default

3. Final

4. Abstract

5. Strictfp

If we are using any other modifier we will get compile time error.

**Default class:** If a class declared as the default then we can access that class only within the current package hence default access is also known as "package level access".

**Note :** We will discuss class modifiers later

**Method Modifiers:**

**Public Members:** If a member declared as the public then we can access that member from anywhere

"but the corresponding class must be visible" hence before checking member visibility we have to check class visibility.

**Default Member** : If a member declared as the default then we can access that member only within the

current package hence default member is also known as package level access.

**Private Members:** If a member declared as the private then we can access that member only with in

the current class. **Private methods are not visible in child classes** where as abstract methods should

be visible in child classes to provide implementation hence private, abstract combination is illegal for methods.

**Note :** We will discuss other method modifiers later.

**OOPS - Part2**

**Is-A Reletionship:**

1. Also known as inheritance.

2. By using "extends" keywords we can implement IS-A relationship.

3. The main advantage of IS-A relationship is reusability.

**Example:**

class Parent {

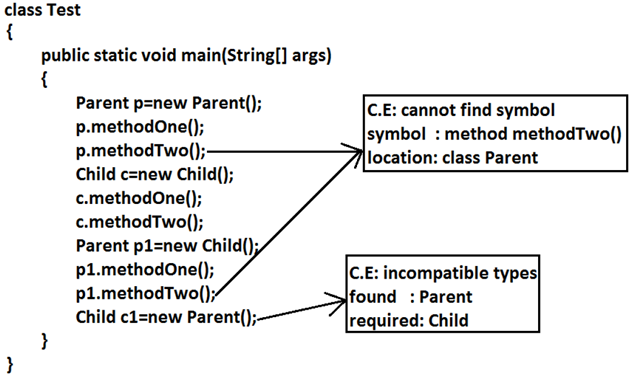
public void methodOne(){ }

}

class Child extends Parent {

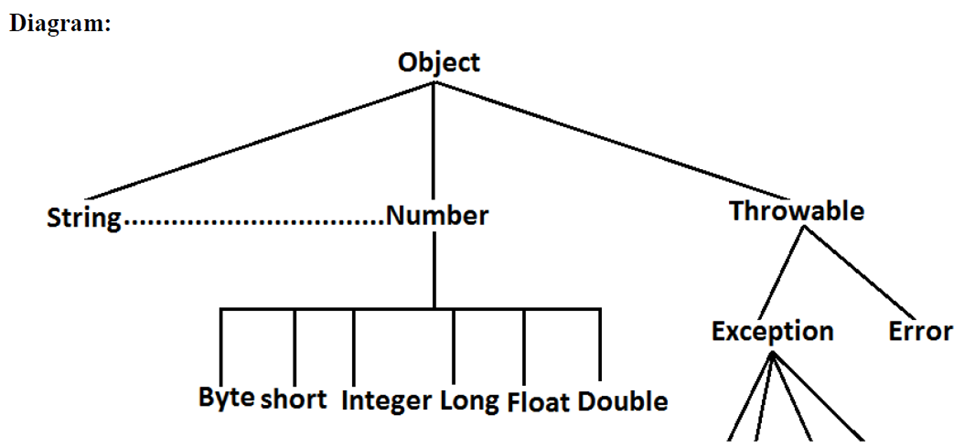
public void methodTwo() { }

}



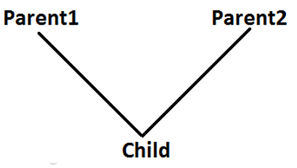
· For all java classes the most commonly required functionality is define inside object class hence object class acts as a root for all java classes.

· For all java exceptions and errors the most common required functionality defines inside Throwable class hence Throwable class acts as a root for exception hierarchy.

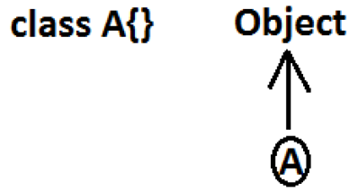


**Note:**

1)Java doesn’t support multiple inheritance for classes.

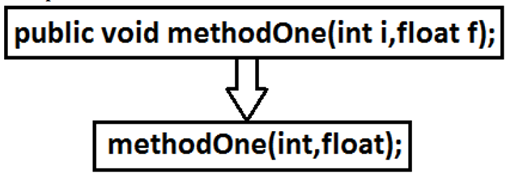


2)If our class doesn't extends any other class then only our class is the direct child class of object.



**Method Signature:**

In java, method signature consists of name of the method followed by argument types.



In java return type is not part of the method signature. Compiler will use method signature while resolving method calls.

**class Test {**

**public void m1(double d) { }**

**public void m2(int i) { }**

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

**Test t=new Test();**

**t.m1(10.5);**

**t.m2(10);**

**t.m3(10.5); //CE**

**}**

**}**

**CE : cannot find symbol**

**symbol : method m3(double)**

**location : class Test**

**Polymorphism:**

Same name with different forms is the concept of polymorphism.

**Example 1:** We can use same abs() method for int type, long type, float type etc.

Example:

1. abs(int)

2. abs(long)

3. abs(float)

**Example 2:**

We can use the parent reference to hold any child objects. We can use the same List reference to hold ArrayList object, LinkedList object, Vector object, or Stack object.

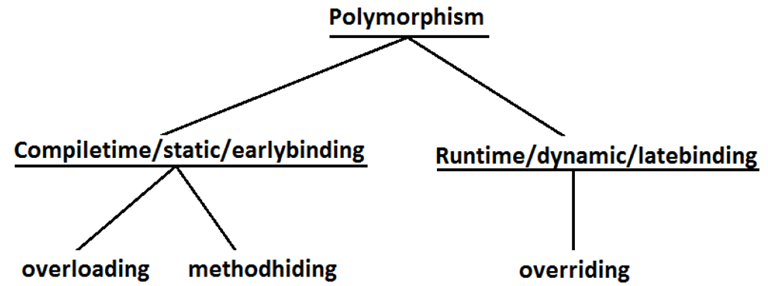
Example:

1. List l=new ArrayList();

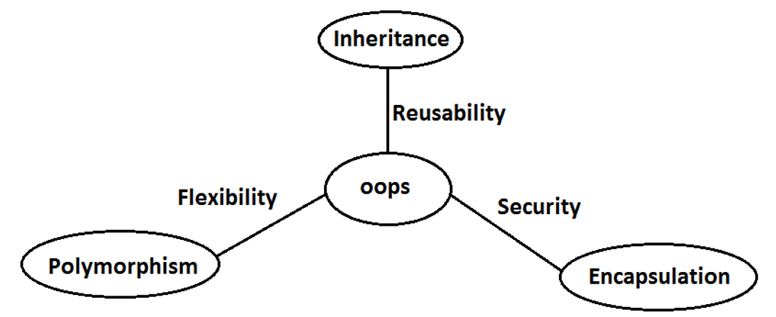
2. List l=new LinkedList();

3. List l=new Vector();

4. List l=new Stack();



3 pillers of OOPs



**Overloading :**

Two methods are said to be overload if and only if both having the same name but different argument types.

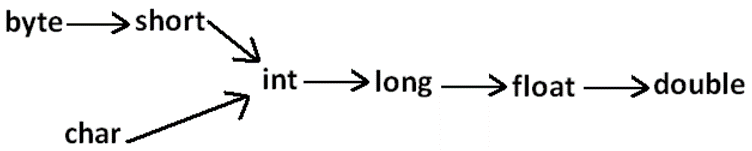
In overloading compiler is responsible to perform method resolution(decision) based on the reference type(but not based on run time object). Hence overloading is also considered as compile time polymorphism(or) static polymorphism (or)early biding.

**Automatic promotion in overloading** :

· In overloading if compiler is unable to find the method with exact match we won't get any compile time error immediately.

· 1st compiler promotes the argument to the next level and checks whether the matched method is available or not if it is available then that method will be considered if it is not available then compiler promotes the argument once again to the next level. This process will be continued until all possible promotions still if the matched method is not available then we will get compile time error. This process is called automatic promotion in overloading.

The following are various possible automatic promotions in overloading.



**Example:**

**class Test**

**{**

**public void methodOne(int i)**

**{**

**System.out.println("int-arg method");**

**}**

**public void methodOne(float f) //overloaded methods**

**{**

**System.out.println("float-arg method");**

**}**

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

**{**

**Test t=new Test();**

**//t.methodOne('a');//int-arg method**

**//t.methodOne(10l);//float-arg method**

**t.methodOne(10.5);//C.E:cannot find symbol**

**}**

**}**

**Overriding :**

1. Whatever the Parent has by default available to the Child through inheritance, if the Child is not satisfied with Parent class method implementation then Child is allow to redefine that Parent class method in Child class in its own way this process is called overriding.

2. The Parent class method which is overridden is called overridden method.

3. The Child class method which is overriding is called overriding method.

Example 1:

**class Parent {**

**public void property(){**

**System.out.println("cash+land+gold");**

**}**

**Public static void marry() {**

**System.out.println("Parent Method"); //overridden**

**method**

**}**

**}**

**class Child extends Parent{ //overriding**

**public static void marry() {**

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

**//overriding method**

**}**

**}**

**class Test {**

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

**Parent p=new Parent();**

**p.marry();//Parent Method**

**Child c=new Child();**

**c.marry();//Child Method**

**Parent p1=new Child();**

**p1.marry();//Child Method**

**}**

**}**

**Note:**

· In overriding method resolution is always takes care by JVM based on runtime object hence overriding is also considered as runtime polymorphism or dynamic polymorphism or late binding.

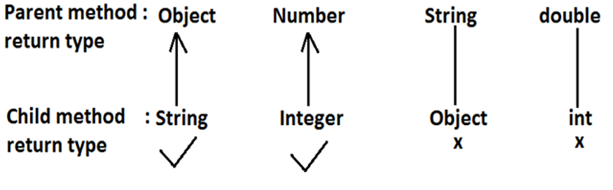
· The process of overriding method resolution is also known as dynamic method dispatch.

**Rules for Overriding:**

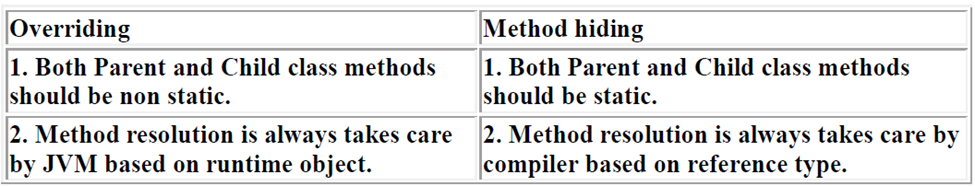
1. In overriding method names and arguments must be same. That is method signature must be same.

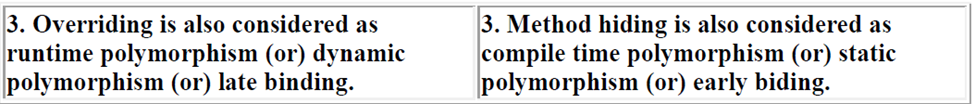
2. Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.

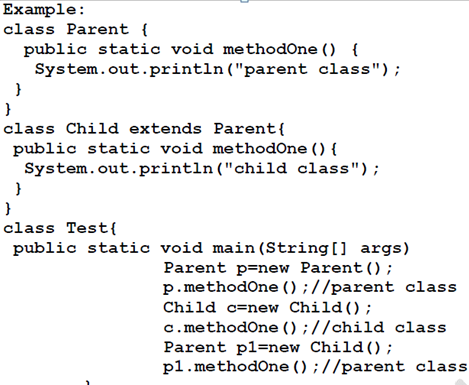
3. According to this Child class method return type need not be same as Parent class method return type its Child types also allowed.



**Method Hiding:**

****

****

****

**Differences between overloading and overriding:**

|  |  |  |
| --- | --- | --- |
| **Property** | **Overloading** | **Overriding** |
| **1) Method names** | Must be same. | Must be same. |
| **2) Argument type** | Must be different (at least order). | Must be same. |
| **3) Method signature** | Must be different. | Cannot be overriding. |
| **4) Return types** | No restrictions. | Must be same until Java 1.5, but from Java 1.5 onwards, co-variant return types are allowed. |
| **5) private, static, final methods** | Can be overloaded. | Cannot be overridden. |
| **6) Access modifiers** | No restrictions. | Weakening/reducing is not allowed. |
| **7) Throws clause** | No restrictions. | If child class method throws any checked exception, the parent class method should throw the same checked exceptions or its super classes, but no restrictions for unchecked exceptions. |