Constants in java

A constant is the identifier which can't be change during execution of the program.

Note : To declare the variable such constants in java use a keyword called final.
final is a keyword which is acting as three places. They are :
1. variable level
2. method level
3. class level

1) At variable level -> If the final variable is initialized further modification are not allowed.

```
eg, final int a=10;
            a=a+10; // not allowed
```

when the final variable is declared immediate assignment is possible but further modification are not possible.

```
e.g, final int a;
            a=10;     //allowed
            a=a+10; // not allowed
```

2) At method level -> once the method is final which is not possible to override, i.e final methods can't be overridden in the sub classes.

```
Syntax : final rtype mname(mparams if any)
            {
            Block of statement;
            }
Eg, final void sum()   // not possible to override
        {
        ............
        }
```

3) At class level -> Once the class is final which can't be inheritable i.e final classes can't be inheritable or reusable.

```
final class a
  {
  ...........
  }
class b extends a        // error
   {
   ............
    }
```

Packages

A package is a collection of different types of class, interfaces and subpackages and in terms in subpackages is divided to classes, interfaces and subpackages and so on.

Advantage

1) To packages we can achieve the slogan of write once and reuse any number of times.

2) Application development time is reduced.
3) Replication of the code is reduced hence it increases the consistency of the application and reduces to storage cost.

There are two types of packages

1) pre – defined package
2) user defined package

1) **pre – defined package :-** A predefined package is one which comes along with the software to make the user applications in an efficient way to develop.

As the part of J2SE we have the following package :

i) **java.lang.* ->** used for specifying language functionality.
This is the default package will be imported our application automatically.
ii) **java.awt.* ->** used for developing GUI – oriented components (Radio Button, check box).
iii) **Java.io.* ->** used for developing string oriented application or file- oriented application.
iv) **Java.awt.event.* ->** used for developing event driven application for GUI component.
v) **Import java.applet.* ->** used for developing browser oriented application.
vi) **Import java.util.* ->** used for developing utility application (for achieving the performance in java/j2ee kind of projects. (collection framework).
vii) **import java.net.* ->** used for developing network related applications.

User – defined packages

A package is said to be user defined package if and only if which is developed by the user to place the list of classes, list of interface and sub-packages to simplicity the user application.

D:\> javac –d . <class_name>.java

creating packages :

To creating our own packages involves the following steps

1) Develop the package at the beginning of a file using the form

package <pkgname>;

2) Define the class that is to be put in the package and declare it public.
3) Create a subdirectory under the directory where the main source files are stored.
4) Store the listing as the classname.java file in the subdirectory created.
5) Compile the file. This creates . class file in the subdirectory.

Note 1 :- java also supports the concept of package hierarchy. This is done by specifying multiple names in a package statement, separated by dots.

package first_package.second_package;

Accessing a package

Java package can be accessed either using a fully qualified class name or using a shortcut approach through the import statement. We use import statement when there are many references to a particular package or the package name is too long and unwidely.

The import statement can be used to search a list of packages for a particular class.

import package1.package2.classname;

We can have any no. of packages in a package hierarchy.

Note :- The statement must end with a semicolon (;) . The import statement should appear before any class definitions in a source file. Multiple import statements are allowed. eg, import firstpackage.secondpackage.myclass;

Note : After defining this statement, all the members of the class myclass can be directly access using the classname or its objects directly without using the package name.

We can also use another approach.

Syntax : import packagename.*;

The star(*) indicates that the compiler should search this entire package hierarchy when it encounters a class name. This implies that we can access all classes contained in the above package directly.

Advantages

The advantage is that we need not have to use long package repeatedly in the program.

Drawback

The major drawback of the shortcut approach is that it is difficult to determine from which package a particular member came.

```
package bank;
public class account
  {
  String name;
  int id,age;
  public void getdata(String na, int ag, int i)
    {
    name=na;
    id = i;
   age = ag;
    }
public void putdata()
  {
```

```
    System.out.println(“ name =” + name);
    System.out.println(“id =” +id);
    System.out.println(“ age=” +age);
   }
  }
  import bank.account;
  class bankdemo
    {
    public static void main(String args[])
      {
      account ac = new account();
      ac.getdata(“Amit kumar”,12,25);
      ac.display();
    }
}
```

1. What is polymorphism? Explain the difference between runtime and compile-time polymorphism?

Ans : polymorphism is a term that describes a situation where one name may refer to different methods. In java there are two types of polymorphism.

a) Compile-time polymorphism
b) Run-time polymorphism

a) Compile-time polymorphism -> It means function call is resolved at compile time. It means which function code will execute in response to a particular function call is known to compiler at compile time.
Example : method overloading.

b) Run-time polymorphism-> It means function call is resolved at run-time. It means which function code will execute in response to a particular function call is not known till execution of program.

Example : Dynamic method dispatch(DMD)

```
class A
 {
 void callme()
   {
   System.out.println("Inside A's callme method");
   }
 }
class B extends A
 {
 void callme()
   {
   System.out.println("Inside B's callme method");
   }
 }
class C extends A
```

```
 {
 void callme()
   {
   System.out.println("Inside C's callme method");
   }
 }
class dmd
  {
  public static void main(String args[])
    {
    A a=new A();
    B b=new B();
    C c=new C();
    A r;
    r=a;
    r.callme();
    r=b;
    r.callme();
    r=c;
    r.callme();
    }
  }
```

Access specifiers in Java

Java allows you to control access to classes, methods and fields so, called access specifiers. Java offers four access specifier.

i) public -> public classes, methods, and fields can be accessed from everywhere.

```
public class square
   {
   public int x,y,size;
   }
```

ii) protected -> protected methods and fields can only be accessed within the same class to which the methods and fields belong, within its subclasses, and within classes of the same package, but not from anywhere else.

```
public class abcd
  {
  protected int x,y;
  }
```

iii) default(no specifier) -> If we don't set access to specific level, then such a class, method, or field will be accessible from inside the same package to which the class, method, or field belongs, but not from outside this package.

iv) private -> private methods and fields can only be accessed within the same class to which the methods and fields belong. Private methods and fields are not visible within subclasses and are not inherited by subclasses.

```
public class square
  {
   private double x,y;
  }
```

Finalizer methods -> java supports a concept called finalization, which is just opposite to initialization. It automatically frees up the memory resources used by the objects. But objects may hold other non-object resources such as file descriptors or window System fonts. The garbage collector can't free these resources. In order to free these resources we must use a finalizer methods.

The finalizer methods is simply finalize() and can be added to any class. Java calls that method whenever it is about to reclaim the space for that object. The finalize method should explicitly define the tasks to be performed.

```
public void finalize()
  {
  ............
  }
```

this -> this is an implicit object by the JVM which is used for two purposes :

1) To distinguish between the formal parameters and class data members therefore the class data members must be preceded by implicit object called this.

2) "this" is an implicit object which is used to refer the current class object.

```
class rectangle
  {
  int length,breadth;
  void show(int length, int breadth)
    {
    this.length=length;
    this.breadth=breadth;
    }
 int calculate()
  {
 return(length*breadth);
   }
 }
public class example
 {
 public static void main(String args[])
  {
  rectangle r=new rectangle();
  r.show(5,6);
  int area=r.calculate();
  System.out.println("the area of a rectangle is" + area);
  }
```

```
 }
```

super -> super is used for pointing the super class instance.

```
class a
  {
  int k=10;
  }
class test extends a
  {
public void m()
   {
System.out.println(super.k);
  }
class demo
   {
  public static void main(String args[])
    {
    Test t=new Test();
    t.m();
   }
}
```

super() -> It is used for calling super class default constructor. To call the default constructor of super classes explicitly no need to specify super.

Accessing superclass members

If our method overrides one of its superclass's methods, we can invoke the overridden method through the use of the keyword super. We can also use super to refer to a hidden field.

```
public class superclass
   {
   public void m1()
     {
     System.out.println("Super class");
     }
  }
public class subclass extends superclass
  {
public void m1()
  {
  super.m1();
  System.out.println("sub class");
  }
}
class pdemo
  {
public static void main(String args[])
  {
```