**3.5 Define a package**

**Package**: A package represents a directory that contains related group of classes and interfaces.

Package is a collection of classes, interfaces and subpackages.

A subpackage internally contains collection of classes, interfaces and also subpackages.

**Benefits of packages:**

1.Packages are useful to arrange related classes and interfaces into a group. For example , in java, all the classes and interfaces which performs input and output are stored in java.io package.

2.The class contained in the packages of other program can be easily reused.

3. The classes and interfaces of a packages are isolated from the classes and interfaces of another package i.e., two classes in two different packages can have the same name.

4.Packages provide a way to hide classes and interfaces in a separate subdirectory, so that accidental deletion of classes and interfaces will not take place.

5.Packages also provide a way for separating design from coding.

**Types of packages:**Java supports two types of packages

* 1. Java API packages
  2. User defined packages

**1. Java API packages (or) Java system packages:** Java API [Application Programming Interface] provides a large no. of classes grouped into different packages according to functionality.

Frequently used java API packages are

|  |  |
| --- | --- |
| **Package name** | **Contents** |
| java.lang | Language support classes. These are classes that java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, strings, math functions, threads and exceptions. |
| java.util | Language utility classes such as vectors, Hash tables, random numbers , date, etc. |
| java.io | Input / output support classes. They provide facilities for the input and output of data. |
| java.awt | Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on. |
| java.net | Classes for networking. They include classes for communicating with local computers as well as with internet servers. |
| java.applet | Classes for creating and implementing applets. |

**Using java API packages [using system packages]:**

**2.User defined packages:** we can define our own packages. These user defined packages are created as follows:

We must first declare the name of the package using the ‘package’ keyword followed by a package name then we define a class as public, just as we normally define a class.

packagepackagename; // package declaration

public class classname // class definition

{

----------------

----------------

}

* Here the package name is name of the package.
* The class name is the name of the class.

**CREATING PACKAGES:**Creating our own package involves the following steps:

1. Declare the package at the beginning of file using the form

**packagepackagename;**

1. Define the class that is to be put in the package and declare it **public.**
2. Create a subdirectory under the directory where the main source files are stored.
3. Store the **.java** file in the subdirectory created.
4. Compile the .java file and this creates **.class** file in the subdirectory.

**3.6 Describe the concept of classpath**

**Classpath:**Packages are mirrored by directories. Even though packages solve many problems from access control and name space collisions, they cause serious problems some times when we compile and run the program. We consider the route of any package hierarchy by using class path.

Java run time system look for packages by three ways:

1. **By default the run-time system uses the current working directory as its starting point.**
2. **Using directory path by setting the CLASSPATH environmental variable.**
3. **Using –classpath option with java and javac to specify the path to our classes.**

|  |
| --- |
| **There are two ways to load the class files temporary and permanent.** |

* Temporary
  + By setting the classpath in the command prompt
  + By -classpath
* Permanent
  + By setting the classpath in the environment variables

In order to create**CLASSPATH** environmental variable, we follow the steps given below.

1. On the desktop, select ‘my computer’ and double click on it.
2. Select C Drive -> Program Files->java->jdk 1.7.0->bin and double click on it.
3. In the address bar select the address, copy it and then close the window.
4. Right click on ‘my computer’ select the ‘properties’ option
5. The properties dialog box will appear. In this select advanced tab button.
6. Select environmental variables double click on it.
7. Select ‘new’ button, then a window will appear

**Variable name**

**Variable value**

1. In this window, type variable name as ‘path=’
2. And for variable value paste the address we have copied earlier and put ‘semicolon’ at the end of this address.
3. Click on ok button.

**Command prompt:**

**set path=C:\Program Files\Java\jdk1.7.0\_07\bin;;**

**3.7Describe the concept of Access protection**

**Access Protection:** An Access Specifier is a keyword that is used to specify how to access members of a class or the class itself. It may be necessary in some situations to restrict the access to certain variables and methods from outside the class to achieve this in java, we apply visibility modifiers [access specifiers] to the instance variables and methods.**Java provides four types of access modifiers:**

* 1. Public
  2. Private
  3. Protected
  4. default

**Public :’**public’ members of a class are accessible every where outside the class. So any other program can read them and use them.

**Private :**Private members of a class are not accessible any where outside the class. They are accessible only within the class by the methods of that class.

**Protected:** A variable or method which is declared as protected can be accessible to all classes and subclasses in the same package and it is also accessible to sub class in other packages.

**Default:**If no access specifier is written by the programmer , then the java compiler uses a ‘default’ access specifier 'default’ members are accessible outside the class, but with in the same directory .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access modifier**  **Access location** | **Public** | **Protected** | **Default**  **(No access specifier)** | **Private** |
| **Same class** | yes | yes | yes | yes |
| **Subclass in same package** | yes | yes | yes | No |
| **Other classes in same package** | yes | yes | yes | No |
| **Subclass in other packages** | yes | yes | No | No |
| **Non-Subclasses in other packages** | yes | No | No | No |

**3.8 Use of classes from another package**

exampleprogram for packages

|  |  |
| --- | --- |
| package p1;  public class A  {  public void display()  {  System.out.println("i am user defined packages");  }  } | B.java  import p1.A;  class B  {  public static void main(String args[])  {  A obj=new A();  obj.display();  }  } |

Step1 : Take a note pad

Step2: enter the program

step 3 : To create a user defined package "package packagename"

step4 : save the program as "class name.java(ex:A.java)"

same process for other notepad also

step1 : save the program as "class name.java(ex:B.java)"

step2 : Here we import the package(for ex : pack.A)

Compilation :javac -d . classname.java(it creates subdirectory pack and keeps that class in that directory)(Only for user defined package)

javac classname.java (it compiles main function here we need not to create any sub directory)

**3.8.2 Usage of class from a class**

|  |  |
| --- | --- |
| M.java  package p2;  public class M  {  public static int p=9;  public void display()  {  System.out.println(“class M”);  System.out.println(“p=”+p);  }  } | packagetest1.java  import p1.A;  import p2.M;  class packagetest1 extends M  {  public static void main(String args[])  {A ob1=new A();  M ob2=new M();  ob1.display();  ob2.display();  p=10;  ob2.display();  }  } |

**3.9 Appreciate the concept of importing packages**

Importing packages (Accessing packages):

Rules for importing packages:

1. The import statement must end with a semicolon.
2. The import statement should appear before any class definition in a sourced file.
3. Multiple import statements are allowed.

There are three ways to access the package from outside the package.

1. import package.\*;
2. import package.classname;
3. fully qualified name.

**1) Using packagename.\***

* If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages.
* The import keyword is used to make the classes and interface of another package accessible to the current package.

|  |  |
| --- | --- |
| Save by A.java  package pack;  public class A  {  public void msg()  {  int a=20,b=4;  System.out.println("thevalue is"+(a\*b));  }  } | save by B.java  import pack.\*;  class B  {     public static void main(String args[])  {      A obj = new A();  exp obj1=new exp();  obj.msg();     obj1.msg();    }  } |
| Save by exp.java  package pack;  public class exp  {  public void msg()  {  for(int i=0;i<5;i++)  {  System.out.println(i);  }  }  } | C:/>javac -d . A.java  javac -d . exp.java  javac -d . B.java  java B  output :  the value is 80  0  1  2  3  4 |

#### 2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

|  |  |
| --- | --- |
| package p1;  public class A  {  public void display()  {  System.out.println("i am user defined packages");  }  } | B.java  import p1.A;  class B  {  public static void main(String args[])  {  A obj=new A();  obj.display();  } } |

#### 3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name

|  |  |
| --- | --- |
| //save by A.java  package pack;  public class A  {   public void msg()  {  System.out.println("Hello");  }   } | //save by B.java  class B  {    public static void main(String args[])  {      pack.A obj = new pack.A();      obj.msg();    }  } |

**3.10 Know the concept of interfaces**

We kn**ow that in multiple inheritance**, subclasses are derived from multiple super classes. If two super classes have same names for their **members (**variables and methods) then which member is inherited into subclass is the main confusion in the multiple inheritance. This is the reason, java does not support the concept of multiple inheritance. This confusion is reduced by using multiple interfaces to achieve multiple **inheritance.**

**Interfaces**:

* An interface is a specification of method prototypes. This means,only method names are written in the interface without method bodies.
* An interface will have 0 or more abstract methods which are all public and abstract by default.
* An interface can have variables which are public static and final by default. This means all the variables of the interfaces are constants.
* We cannot create an object to an interface, but we can create a reference of interface type. Interface reference can refer to the objects of its implementation classes.
* None of the methods in an interface can be private, protected or static.
* All the methods of an interface should be implemented in its implementation classes.
* An interface can extend another interface.
* An interface cannot implement another interface.
* A class can implement multiple interfaces.

Syntax: class classname implements interface

Example: class Sample implements interfac1,interface2…

**3.11 Define an interface**

**Defining an interface:**

* An interface is basically a kind of class. Like classes, interfaces contain methods and variables but with a major difference.
* An interface is a specification of method prototypes. This means, only method names are written in the interface without method bodies.
* An interface will have 0 or more abstract methods which are all public and abstract by default.
* An interface can have variables which are public static and final by default. This means all the variables of the interfaces are constants.
* The syntax for defining an interface is very similar to that for defining a class. The general form of an interface definition is:

**Syntax:** interface interfacename

{

Variables declaration;

Methods declaration;

}

Here, ‘interface’ is the keyword and interface name is any valid java variable.

* Variables are declared as follows:

Syntax:static final type variable=value;

Example: static final int a=10;

* Methods declaration will contain only a list of methods without any body statements.

Syntax: return-type method-name (parameter list);

Example for interface definition:

interface item

{

static final int n=10;

static final String name=”hai”;

void display();

}

**3.12 Write the difference between class and interface**

|  |  |
| --- | --- |
| Class | Interface |
| 1)The members of a class can be constants or variables | 1)the members of an interface are always declared as constants i.e their values are final |
| 2)The class definition can contain code for each of its methods. | 2)methods in an interface are abstract in nature i.e there is no code associated with them,Later defined by class that implements interface |
| 3.It can be instantiated by declaring objects | 3.It cannot be used to declare objects.It can be only be inheriteted by a class |
| 4.can use access access specifiers like public,private or protected | 4.can only use public access specifier. |

**3.13 Implement interfaces**

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the implements statement in a class definition. This is done as follows:

classclassname implements interfacename

{

body of classname

}

Here the class classname “implements” the interface interfacename.

* A more general form of implementation may look like this:

**classclassname**extends superclass implements interface1,interface2,……………interfacen

{

Body of classname

}

The above syntax shows that a class extends another class while implementing interfaces.

// program on implementing interfaces

interface Area

{

static final float pi=3.14f;

float compute(float x,float y);

}

class Rectangle implements Area

{

public float compute(float x,float y)

{

return(x\*y);

}

}

class Circle implements Area

{

public float compute(float x, float y)

{

return(pi\*x\*x);

}

}

class interfaceTest

{

public static void main(String args[])

{

Rectangle rect=new Rectangle();

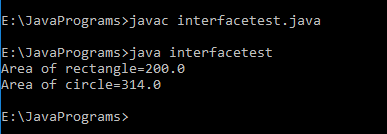
System.out.println("Area of rectangle="+rect.compute(10,20));

Circle cir=new Circle();

System.out.println("Area of circle=" +cir.compute(10,0));

}

}Output:



**Extending interfaces:** One interface can inherit another by use of the keyword extends. The syntax is the same as for inheriting classes.

When a class implements an interface that inherits another interface, it must provide implementation of all methods defined within the interface inheritance.

Note: Any class that implements an interface must implement all methods defined by that interface, including any that inherited from other interfaces.

interface f1

{

void disp1();

}

interface f2 extends f1

{

void disp2();

}

class A implements f2

{

public void disp1()

{

System.out.println("This is display of i1");

}

public void disp2()

{

System.out.println("This is display of i2");

}

}

public class Ext\_iface

{

public static void main(String args[])

{

A obj = new A();

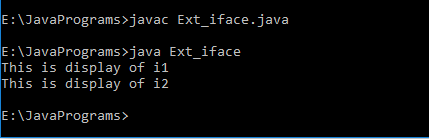
obj.disp1();

obj.disp2();

}

}

Output



**Multiple inheritance using interface:**

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the implements clause in a class definition, and then create the methods declared by the interface.

The general form of a class that includes theimplements clause looks like this:

Access-specifier class classname [extendssuperclass] [implements interface, [, interface..]]

{

// class body

}

If a class implements from more than one interface, names are separated by comma.

class student

{

int rollno;

voidgetrollno(int r)

{

rollno=r;

System.out.println("roll no is" +rollno);

}

}

class marks extends student

{

int m1,m2;

voidgetmarks(int x,int y)

{

m1=x;

m2=y;

System.out.println("subject1 marks is"+m1);

System.out.println("subject2 marks is" +m2);

}

}

interface sports

{

float score=10.0f;

voidputwt();

}

class multi extends marks implements sports

{

public void putwt()

{

System.out.println("Displaying student details .. ");

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

}

void display()

{

float total=m1+m2+score;

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

}

}

class Multi1

{

public static void main(String args[])

{

multiobj = new multi();

obj.getrollno(10);

obj.getmarks(90,90);

obj. putwt();

obj.display();

}

} **3.14 Know the scope of variables in interfaces**

Interface can be used to declare a set of constants that can be used in different classes. Interfaces do not contain method definitions, there is no need to worry about implementing any methods. The constant values will be available to any class that implements the interface. The values can be used in any method, as part of any variable declaration, or anywhere where we can use a final value.

Example:

interface A

{

int a=50;

float b=30.8**f**;

}

class B implements A

{

intlen=a;

public float calculate()

{

return(len\*b);

}

}

**c**lass C

{

public static void main(String args[])

{

B b=new B();

System.out.println(“Result=” +(b.calculate()));

}

}

**Output:**

