# Packages,final keyword,about object class and its methods and object cloning with examples

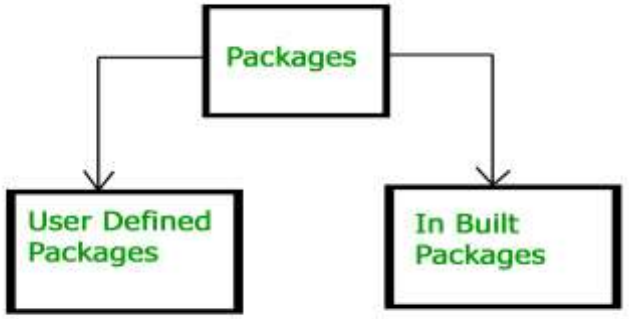
-K.L.Madhavi

# Java Package

**Package** in java is a mechanism to encapsulate a group of classes, sub packages and interfaces.Packages are used for:

- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier.
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- Packages can be considered as data encapsulation (or data-hiding).

# Types of packages




**Built-in Packages:**These packages consist of a large number of classes which are a part of Java **API**.Some of the commonly used built-in packages are:
**1.java.lang:** Contains language support classes(e.g classes which defines primitive data types, math operations). This package is automatically imported.
**2. java.io:** Contains classes for supporting input / output operations.
**3. java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
**4. java.applet:** Contains classes for creating Applets.
**5. java.awt:** Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc). 6)
**6. java.net:** Contain classes for supporting networking

**User-defined packages:** These are the packages that are defined by the user.The “package” keyword is used to create user defined package.

**Syntax:** package <package_name>

- Whatever the classes we want to create in the package ,It should not contain main class.
- Multiple programs should be written for placing multiple classes in same package name.

## Benefits of User-Defined Packages

- **Code Organization:**User-defined packages allow developers to group related classes and resources together, making it easier to navigate and understand the codebase.
- **Code Reusability:** By creating user-defined packages, you can encapsulate commonly used classes, utilities, or modules, making them easily reusable across different projects.

```
// Name of the package must be same as the directory
// under which this file is saved
package myPackage;

public class MyClass
{
        public void show()
        {
            System.out.println("Myclass class");
         }
}
```

Save→Myclass.java
Compile→javac –d.A.java

Now we can use the **MyClass** class in our program.

```
/* import 'MyClass' class from myPackage */
import myPackage.MyClass;

public class MainDemo
{
       public static void main(String args[])
      {

          // Creating an instance of class MyClass in the package.
          MyClass obj = new MyClass();

          obj.show();
}
}
```

Save→MainDemo.java
Compile→javac MainDemo.java
Execution→java MainDemo

# Final Keyword

- The final keyword in Java is used as a non-access modifier applicable only to a variable, a method, or a class. It is used to restrict a user in Java.

| Final Variables | Final Methods | Final Classes |
|---|---|---|
| Once assigned, a final variable's value cannot change. It effectively creates a constant. | Final methods cannot be overridden by subclasses, preserving their behavior. | Final classes cannot be extended, preventing inheritance from them. This ensures class integrity. |

# Example Programs

## Final Variable

```
class Bike
{
    final int speedlimit=90 //final var
    void run()
    {
      speedlimit=120;
    }
    public static void main(String args[])
    {
        Bike obj=new Bike();
        obj.run();
    }
}
```

## Final Method

```
class Bike
{
    final void run()
    {
        System.out.println("running bike");
    }
}
class Honda extends Bike
{
     void run()
     {
       System.out.println("running honda bike");
     }
     public static void main(String args[])
     {
         Honda  obj=new Honda();
         obj.run();
     }
}
```

## Final class

```
final class Bike{}
class Honda1 extends Bike{
{
    void run()
    {
        System.out.println("running");
    }
    public static void main(String args[])
    {
        Honda1 obj=new Honda1();
        obj.run();
    }
}
```

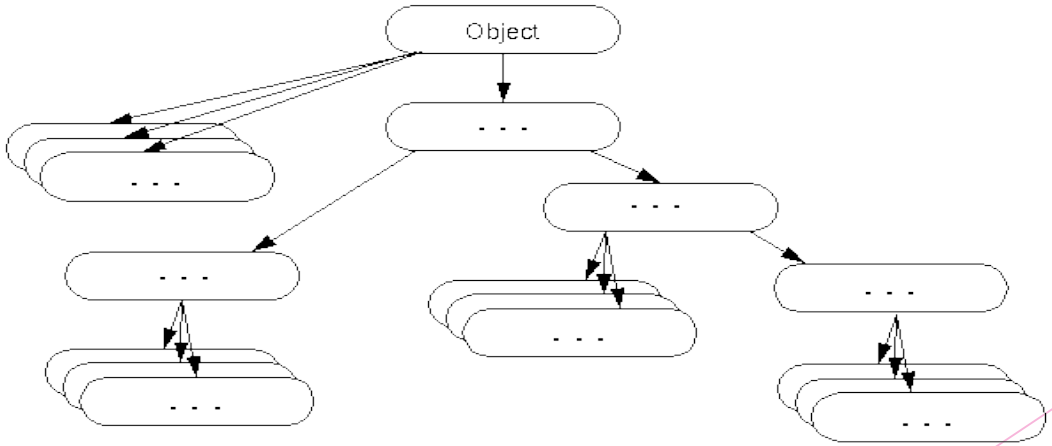
# Object class in Java

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

- Package:java.lang
- Every class in java Directly / Indirectly derived from Object Class.
- It is the Parent class of all classes in java.
- Uses:

1.to define common behaviour of Objects like cloning,comparing etc.

2.to refer an object whose Type is not known.



# Example Program

```
public class ObjectClassExamplel{
        public static void main(String args[])
        {
              checkObjectType(1);
              checkObjectType(2L);
              checkObjectType(1.5f);
              checkObjectType("string type");
              checkObjectType(6.2d);
        }
        public static void checkObjectType(object input){
              if(input instanceof Integer){
                    System.out.println(input+" is of Integer type.");
              }
              else if(input instanceof Float){
                    System.out.println(input+" is of Float type.");
              }
              else if(input instanceof Long){
                    System.out.println(input+" is of Long type.");
              }
              else if(input instanceof String){
                    System.out.println(input+" is of String type.");
              }
              else
              {
                    System.out.println(input+" is of "+input.getClass().getTypeName()+" type.");
              }
}
```

# Object Class Methods

- Class & String Methods(2)
- Hashing Method(1)
- Object Comparison & Cloning Method(2)
- GC Finalization Method(1)
- Object Notifying Methods(2)
- Thread Waiting Methods(1)

# Object Class Methods

| Method | Description |
|---|---|
| public final Class getClass() | returns the Class object of this object. Class object can be used to get the metadata of defined class. |
| public String toString() | Returns the string representation of this object. |
| public int hashCode() | Returns the hashcode number for this object. |
| public Boolean equals(Object obj) | Compares the given object to this object. |
| protected Object clone() throws CloneNotSupportedException | Creates and returns the exact copy of this object. |
| protected void finalize() throws Throwable | Is invoked by GC before destroying an object |
| public final void notify() | Wakes up single thread which is waiting to acquire objects monitor |
| public final void notifyAll() | Wakes up all the threads which are waiting to acquire objects monitor. |
| public final void wait()throws InterruptedException | Causes the current thread to weight until another thread notifies |

# Object Cloning

- The object cloning is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.
- The **clone() method** is defined in the Object class.

  **Syntax** of the clone() method is as follows:

  **protected Object clone() throws CloneNotSupportedException**
- The clone() method saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time so to decrease it we use object cloning.

# Example Program

```
class Student implements Cloneable{
int rollno;
String name;
Student(int rollno,String name){
this.rollno=rollno;
this.name=name;
}
Public Object clone()throws CloneNotSupportedException{
return super.clone();
}
Public static void main(String args[]){
try{
Student s1=new Student(2,”Arya”);
Student s2=new (Student)s1.clone();
System.out.println(s1.rollno+””+s1.rollno”);
System.out.println(s2.rollno+””+s2.name”);
}catch(CloneNotSupportedException c){}
}}
```