Unit 3

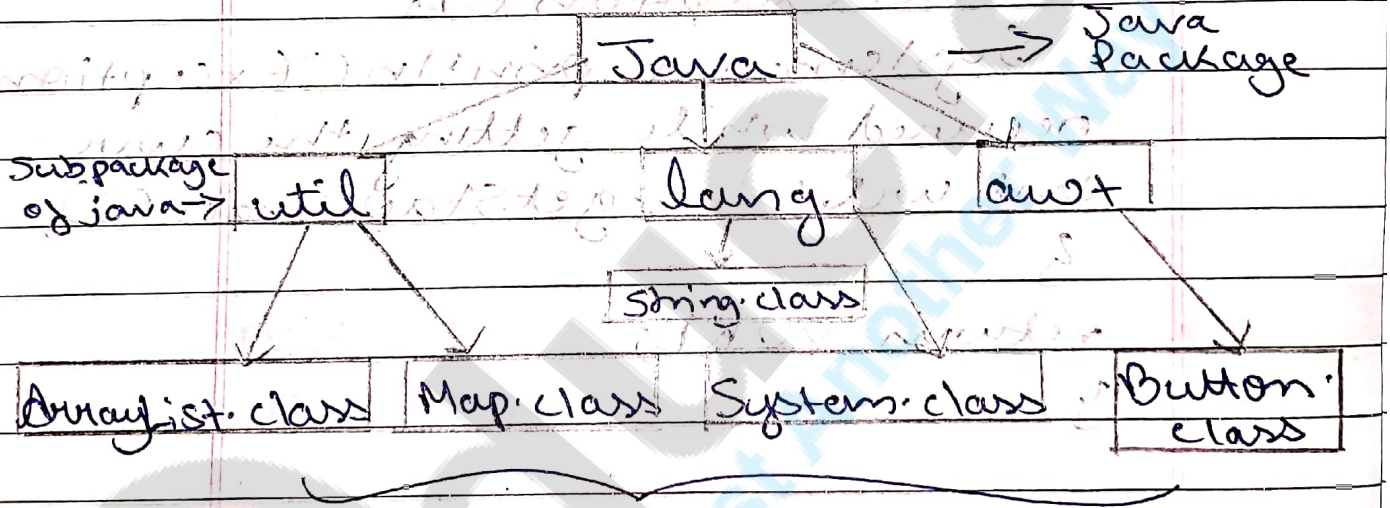
Q] Package concept / Creating user defined package

A package is a collection of similar types of classes, interfaces and sub-packages.

Purpose of package

The purpose of package concept

is to provide common classes and interfaces for any problem separately. In other words if we want to develop any class or interface which is common for most of the java programs then such common classes and interfaces must be place in a package.



Packages in Java are the way to organize files when a project has many modules. Same like we organize our files in computer. For example we store all movies in one folder and songs in other folder, here also we store same type of files in a particular package for example in awt

PAGE NO.....
DATE. ...../...../.....

package we have all the classes and interfaces for design GUI components.

Advantages of package

- Package is used to categorize the classes and interfaces so that they can be easily maintained.
- Application development time is less, because reuse of the code.
- Application memory space is less (main memory)
- Application execution time is less
- Application performance is enhanced (improved)
- Redundancy (repetition) of code is minimized
- Package provides access protection.
- Package removes naming collision.

Types of package

Package are classified into two type which are given below:

1) Predefined or built-in package
2) User defined package

Predefined or built-in package
These are the package which are already designed by the Sun Microsystem and supply as a part of java API, every predefined package is collection of predefined classes, interfaces and sub-package.

User defined package
If any package is designed by the user is known as user defined package. User defined package are those which are developed by java programmer and supply as a part of their project to deal with common requirement.

Rules to create user defined package
- package statement should be the first statement of any package program.
- Choose an appropriate class

PAGE N: .....
DATE: ...../...../.....

name or interface name and whose modifier must be public.
- Any package program can contain only one public class or only one public interface but it can contain any number of normal classes.
- Package program should not contain any main class (that means it should not contain any main()).
- modifier of constructor of the class which is present in the package must be public. (This is not applicable in case of interface because interface have no constructor).
- The modifier of method of class or interface which is present in the package must be public (This rule is optional in case of interface because interface methods by dfault public)
- Every package program should be saved either with public class name or public Interface name

//Sum.java → Save package programs with 'public' class name

```
package MyPackage → first statement of java program is package
public class Sum → class modifier must be 'public'
{
public Sum() → constructor modifier must be 'public'
{
System.out.println("Sum class constructor");
}
public void show() → method modifier must be 'public'
{
System.out.println("Sum class constructor");
}
public void show() → method modifier must be 'public'
{
System.out.println("Sum class method");
}
}
```

Compile package programs

For compilation of package program first we save program

with public className.java and it is compiled using below syntax:

Syntax:

javac -d . className.java

javac -d path className.java

In above syntax "-d" is a specific tool which tells to java compiler create a separate folder for the given package in given path. when we give specific path then it creates a new folder at that location and when we use . (dot) then it create a folder at current working directory.

Note: Any package program can be compiled but cannot be executed or run. These program can be executed through user defined program which are importing package program.

Example of package program

Package program which is saved with A.java and compiled by javac -d . A.java

Example

```
package mypack;
public class A
{
  public void show()
{
System.out.println("Sum method");
}
}
```

Import above class in below program using import packageName.className

Example

```
import mypack.A;
public class Hello
{
public static void main(String
arg[])
A a=new A();
 a.show();
System.out.println("Show() class
  A");
}
}
```

In the above program first we create package program

which is saved with A.java and compiled by "javac -d . A.java". Again we import class "A" in class Hello using "import mypack.A;" statement...

3) Access control protection :-
Packages add another dimension to the access control. Java provides many levels of protection to allow fine-grained control over visibility of the variables and methods within classes, subclasses, and packages.
Classes and packages are means of encapsulating and containing the name space and scope of the variables and methods.
Packages behaves as containers for classes and other subordinate packages.
Classes act as containers for data and code.

Class Members Visibility

The Java's smallest unit of abstraction is class. Because of the interplay between the classes and packages, Java addresses the following four categories of visibility for class members:

- Subclasses in same package
- Non-subclasses in same package
- Subclasses in different packages
- Classes that are neither in same package nor in subclasses

The three access modifiers are:

- public
- private
- protected

Anything declared as public can be accessed from anywhere.

Anything declared as private can't be seen outside of its class.

PAGE NO
DATE / /

Class Member Access

When a member doesn't have an explicit access specification, then it is visible to the subclasses as well as to the other classes in the same package. This is the default access. And if you want to allow an element to be seen outside your current package, but only to the classes that subclass your class directly, then declare that element protected.

	Private	Protected	Public	No modifier
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	Yes	Yes	No

Different package non-subclass	No	No	Yes	No

This table applies only to the members of classes. A non-nested class has only two possible access levels i.e. default and public. When a class is declared as public, then it is accessible by any other code. If a class has default access, then it can only be accessed by the other code within its same package. When a class is public, it must be only the public class declared in the file that must have the same name as the class.

Examples:-

1) In this example, we will create two packages and the classes in the packages will be having the default access modifiers and we will try

PAGE NO.....
DATE...../...../.....

// to access a class from one package from a class of second package.

```
package p1;
class Geek {
    void display() {
        System.out.println("Hello
            World!");
    }
}

package p2;
import p1.*;
class GeekNew
{
    public static void main(String
    args[]) {
        Geeks obj = new Geek();
        obj.display();
    }
}
```

Output:
Compile time error

2) In this example, we will create two classes A and B within same package p1. We will declare a method in class A as private and try to access this method from class B and see the result.

```
package P1;
class A
{
private void display()
{
System.out.println("GreeksforGreeks");
}
}
class B
{
public static void main(String args[])
{
A obj = new A();
obj.display();
}
}
```

Output

error: display() has private access
in A obj.display();

3) In this example, we will create two packages p1 and p2. Class A in p1 is made public, to access it in p2. The method display in class A is protected and class B is inherited from class A and this protected method is then accessed by creating an object of class B.

```
package p1;
public class A
{
protected void display()
{
System.out.println("Geeks for
Geeks");
}
}

package p2;
import p1.*;
class B extends A
{
public static void main(String
```

```
args[]){
{
B obj = new B();
Obj.display();
}
}
```

Output

GeeksforGeeks

4)
```
package p1;
public class A
{
public void display()
{
System.out.println("GeeksforGeeks");
}
}

package p2;
import p1.*;
class B
{
public static void main(
String args[]){
{
A obj = new A();
```

obj.display();
}
}

Output
Cracks for Cracks

Q) Java interface:
Another way to achieve abstraction in Java, is with interfaces.
An interface is a completely "abstract class" that is used to group related methods with empty bodies:

Example
interface Animal {
public void animalSound(); // interface method (does not have body)
public void run();
}

To access the interface methods, the interface must be "imple-

mented" by another class with the implements keyword. The body of the interface method is provided by the "implement" class.

Example

```
interface Animal {
  public void animalSound();
  public void sleep();
}

class Pig implements Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    System.out.println("Zzz");
  }
}

class MyMainClass {
  public static void main(
    String args[])
  {
    Pig myPig = new Pig();
    myPig.animalSound();
```

myPig.sleep();
}
}

-like abstract classes, interfaces cannot be used to create objects.
-Interface methods do not have a body - the body is provided by the "implement" class.
- On implementation of an interface, you must override all of its methods
- Interface methods are by default public, static and final
- An interface cannot contain a constructor.

why and when to interfaces?
1) To achieve security - hide certain details and only show the important details of an object (interface).
2) Java does not support "multiple inheritance". However, it can be achieved with interfaces, because the class can implement multiple

Interfaces

Multiple Interfaces

```
interface FirstInterface {
    public void myMethod();
}
interface SecondInterface {
    public void myOtherMethod();
}
class DemoClass implements
FirstInterface, SecondInterface {
    public void myMethod() {
    System.out.println("Some text
    ...");
    }
    public void myOtherMethod() {
    System.out.println("Some
    other text...");
    }
}

class MyMainClass {
    public static void main(
    String[] args) {
    DemoClass myObj = new
    DemoClass();
```

```
        myObj.myMethod();
        myObj.myOtherMethod();
    }
}
```