Unit - III Package and Interface

(#) Defining package

- Packages in Java are a mechanism that encapsulate a group of classes, sub packages, and interface.

- Packages are used for :-

↳ Prevent naming conflicts by allowing classes with same name.
↳ They make it easier to organize, locate and use classes
↳ Also provide controlled access for protected member.

- The package keyword is used to create a package in Java.

Ex:-
```
Package my pack ;
    public class simple {
        public static void main (String [] args) {
            System.out.println ("welcome to Package");
        }
    }
```

Types of packages in Java

- Built - in Packages
- User - defined Packages

① Built - in Packages

- These packages consist of a large number of classes which are a part of Java API.
- Some of the commonly built - in packages are :-

↳ java.lang
Contain language support classes. This packages are automatically imported.

↳ java.io
Contain classes for supporting input/output operations.

↳ java.util
Contains utility classes which implement data structure like Linked List, Dictionary.

↳ java.applets
Contains classes for creating applets.

↳ java.net
Contains classes for supporting networking operation.

② User - defined Packages

These are the packages defined by user.

. Create the package

first we create a directory myPackage then create the Myclass inside the directory with first statement being package name.

```
package myPackage ;

Public class Myclass
 {
    Public static void main (String [] args) ;
     {
        System.out.println (" Packages") ;
     }
 }
```

Importing Packages

- Java provides 'import' Keyword to import classes of another package inside your class. Once a class is imported, you can use the imported class anywhere in your class.
- Importing packages allows us to import not only class but also interface.
- Package import is a feature in java which allows us to reuse the classes available in a package.

Syntax of importing a package

```
// To import all classes of a package
import packagename.* ;

// To import specific class of a package
import packagename.classname ;

// To import all classes of a sub package
import packagename.subpackagename.* ;

// To import specific class of a sub package
import packagename.subpackagename.classname ;
```

Ex:-

```
// Importing all class from a package
import java.util.* ;

// Importing specific class from a package
import java.util.ArrayList ;
```

Java.lang

- In Java, java.lang is one of the core packages that are automatically imported into every Java program.
- It contain classes that are fundamental to Java programming language.
- The most important classes are object which is the root of the class hierarchy, instance.
- following are the important classes in java.lang packages :-

↳ Boolean
↳ Byte
↳ Character
↳ Compiler
↳ Double
↳ float
↳ Integer
↳ Long
↳ Number
↳ Object
↳ Package
↳ Runtime
↳ String
↳ Void

java.util

In Java, java.util is a package that provide a collection of utility classes and interface to perform various common tasks.

- Classes found in java.util are :-

↳ List - Array list, Linked list
↳ Set - Hash Set, Tree Set
↳ Map - Hash Map, Tree Map
↳ Queue - Priority Queue

↳ Date and time - Classes that work with date and time.

↳ Utilities -
- Random
- Scanner
- formatter

Difference between java.lang and java.util

java.lang

- The core package of java language that provide fundamental classes and feature.
- Provides fundamental classes such as string, integer and object.
- The classes are automatically imported in every java program.
- Used for basic programming tasks such as string manipulation.

java.util

- Utility package that provide useful classes and feature for more advanced programming task.
- Provide utility classes such as Array list, Hash Map and Scanner.
- The classes need to be imported in Java program.
- Used for advanced programming task such as date/time manipulation.

java.io

The java.io package in Java provide a set of input and output (I/O) classes that enables you to read from and write to various data source such as files, sockets and stream.

- Classes within java.io packages are :-

↳ file - Represent a file or directory path
↳ file InputStream - Reads a data from a file as byte.
↳ file OutputStream - Writes data to a file as byte.
↳ InputStream Reader - Reads bytes and decode them into character.
↳ OutputStream Writer - Encodes character into bytes using charset

Defining and Implementing Interface

- An interface in Java programming language is defined as an abstract type used to specify the behaviour of a class. An interface in Java is a blueprint of a behaviour. A Java interface contain static constant and abstract methods.

- The interface in Java is a mechanism to achieve abstraction.

- By default, variable in an interface are public, static and final.

- It is used to achieve abstraction and multiple inheritance.

Syntax

```
interface {
    // declare constant field
    // declare method
    // By default
}
```

- To declare an interface, use the interface keyword.

• Ex :-

```
// File : Animal.java
public interface Animal {
        void make sound ( ) ;
        void eat ( ) ;
    }
```

Implement Interface

• The implements keyword is used to implement an interface.

• To access the interface method, the interface must be "implemented" by another class with implement keyword.

Ex :-

```
// File : Dog.java
public class Dog implements Animal {

    // Implementing the make Sound method
    public void make Sound ( ) {
        System.out.println ("woof") ;
    }

    // Implementing the eat method
    public void eat ( ) {
        System.out.println ("Dog is eating") ;
    }
    // Another methods specific to Dog class added here
}
```

Variable in Interface

. In Java, an interface variable is public, static and final. this means that variable's value cannot be changed once it is assigned.

. Interface variable should always be declared and assigned a value. They cannot be left uninitialized.

. It is recommended to use uppercase letter and underscore to name interface variable.

. Interface variable are constant and cannot be modified after initialization.

Ex :-

```
public interface MyInterface {
        // This is constant
        int MY_CONSTANT = 10 ;
    }
```

Extending Interface

. Extending an interface in Java can be done to create a new interface that inherit the method of existing interface.

. This allows you to build on existing functionality without altering the original interface.

Ex:-

```
// Define Base interface
interface Animal {
    void eat();
    void sleep();
}

// Extend the base interface
interface Mammal extend Animal {
    void giveBirth();
}

// Implement Extend interface
class Human implement Mammal {
    public void eat() {
        System.out.println("Eating food");
    }

    public void sleep() {
        System.out.println("sleeping");
    }
    public void giveBirth() {
        System.out.println("Giving birth");
    }

}

public class Main {
    public static void main(String[] args) {
        Human human = new Human();
        human.eat();
        human.sleep();
        human.give birth();
    }
```

In this example;

- Animal is a base interface with eat() and sleep() method.
- Mammals extend Animal and add a new method giveBirth().
- Human is a class that implement Mammal interface.

Nested Interface

- We can declare interface as member of a class or another interface. Such an interface is called a member interface or nested interface.
- In Java, nested interface can be declared with the public, protected, default or private access specifier.
- Uses

↳ To group related interface together
↳ To create more secure code
↳ To implement multiple inheritance
↳ To define contract between classes

- A nested interface is an interface that is declared within another interface or class.

Ex :-

```
// Outer class
Class Outer Class {

    // Nested Interface
    public interface Nested Interface {
        void nested Method ( ) ;
    }

// Implementing nested interface in different class
Class Implementing class implement Outer Class . Nested Interface {

    public void nested Method ( ) {
        System . out . print ln ("Nested method implementation
    }
}

Public class Main {
    public static void main (String [] args ) {
        Outer class . Nested Interface obj = new Implementing class();
        obj. nested method ( ) ;
    }

}
```

- Outer class is a class that contain nested interface.