UNIT 4 PACKAGES AND INTERFACES

Structure

4.0 INTRODUCTION

Java programming language provides a powerful feature called packages which are groups of related classes and interfaces together in a single unit. Packages are used to manage large groups of classes and interfaces to avoid naming conflicts. The Java API itself is implemented as a group of packages. Java supports two types of packages, i.e. built-in packages and user-defined packages. In this unit, you will learn more about the user-defined packages as well as interfaces.

Similar to the Class, Interface is also one of the important features of java. In this unit, you will learn how to define methods in an interface and how to use those methods in a class. Each Interface is compiled into a separate bytecode file just like a regular class, but you cannot create an instance of the interface. This unit provides detailed information about the interfaces from declaration to its implementation. Using this unit, you can learn how to use static as well as default methods in the interfaces. Apart from this information, you will know about the feature of the interface through which you can use multiple inheritances in a java program. In this unit, examples are given at each step which will help you to understand the concepts and write java program.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- explain the need and the purpose of packages,
- create own package and import packages in program,
- explain the concept of interfaces,
- implement interfaces, and

- work with default and static methods in an interface.

4.2 PACKAGE

In java, a package is a collection of similar types of classes, interfaces and sub-packages. It is just like a folder in a file directory. A package statement identifies the directory path for a given class. For example, when you write an application program in java programming language, there may be many individual classes. You can place related classes into a folder and name this folder for organizing these classes; now, it is called a package. In other words, you can say that the package statement defines a namespace in which classes are stored. It is a good practice to group the related classes in a package. Packages are used to avoid name conflicts and to write better maintainable code.

There are two types of packages in java, i.e. built-in packages and user-defined packages.

Built-in packages

As you are already aware, the Java API is a library of prewritten classes to help in programming. Built-in packages are those packages that Java API provides to simplify the task of java programmer. Some java API is discussed in detail in Block 3 Unit 4 of this course.

Java provides several built-in packages or predefined packages. The complete list of these built-in packages can be found at the link:
https://docs.oracle.com/javase/8/docs/api/.

Here are some of the commonly used built-in packages:

- java.lang – it contains fundamental classes to the design of the Java programming language.
- java.io − classes for input and output functions are grouped in this package
- java.util – it contains utility classes

User-Defined Packages

A package that the user creates is known as user-defined package. Besides using built-in packages, whenever the user (programmer) wants to categorize the classes and interfaces and avoid naming conflicts, the user creates a user-defined package for better java programming.

You can create your own packages . For example, the following :

```
└── javaclasses
    └── IGpack
        └── socis.java
```

You need to use the package keyword to create a User-defined package. The subsequent section of this unit gives more about the creation of own package and how to import built-in packages and user-defined packages in a java program.

4.3 ACCESS RULES IN PACKAGES

Before you get into the depth of the topic Packages in Java, you may need to know about accessing rules in packages. You have already come across java access control mechanism and their access modifier keywords such as public, private and protected while writing your java program in previous units of this course. As yet, you have known that package is a collection of related classes. It can also have interfaces and sub-packages. When you put your classes in a package, what will be the accessibility or scope of class within the package? The discussion on this point is given in this section.

Packages in Java add another dimension to access control. Java provides many security levels that offer the visibility of members within the classes, subclasses, and packages. Classes and packages both define the feature of data encapsulation. Class acts as containers for data and methods, whereas Packages works as containers for classes and other sub packages. The package is naming as well as a visibility control mechanism. Java supports four categories regarding the visibility of the class members between classes and packages are concerned:

- Sub classes in the same package
- Sub classes in different packages
- Non-subclasses in the same package
- Classes that are neither in the same package nor subclasses

You can limit the access of classes, methods and variables throughout packages by using access specifiers. The three main access modifiers private, public, and protected provide a range of access required by these categories. A class can have only two access modifier, one is default and another is public. If the class has default access then it can only be accessed within the same package. When a class has public access, it can be accessed from anywhere by any other code outside of the package in which that class is. The following table applies only to members of classes. In the table, 'Yes' means that the data is accessible and 'No' means data cannot be accessible.

Table 1: Java access control mechanism for class member

	Public	**Private**	**Protected**	**No Modifier**
Inside the Same class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	Yes	No	Yes	No
Different package non-subclass	Yes	No	No	No

You can understand the java access control mechanism in a very simple manner. You can declare anything as **public then** it can be accessed from anywhere. Anything

declared as **private** can be seen only within that class. When you do not mention any access modifier, it is called **default** access modifier, and it is visible to sub-classes as well as to other classes in the same package. It means that the scope of this modifier is limited to the package only. If you declare anything as **protected** then code is accessible in the same package and its subclasses present in any other package. This way, java packages provide access control to the classes.

4.4 FINDING PACKAGES AND CLASSPATH

When you develop an application, there may be a number of classes, and for organizing them, you need to create a package. While packages solve many problems related to access control and namespace conflicts perspective. Also, it is important to note that, they become origin of some difficulties when you compile and run java program. This is because the specific location that the java compiler will consider as the root of any package hierarchy is controlled by CLASSPATH. The CLASSPATH is an environment variable that Application ClassLoader uses to locate and load the .class files. The classpath is the path that the Java runtime environment searches for classes and other resource files.

For example, let us say if you create a class socis under a package named IGpack. The directory name must match your package name exactly. So, you create a directory similar to package name as IGpack and put your java file (for example socis.java) inside that directory. Now, your current working directory is IGpack and compiles your socis.java file. After compiling the socis.java file, the class file is stored in IGpack directory as shown in the figure 1.

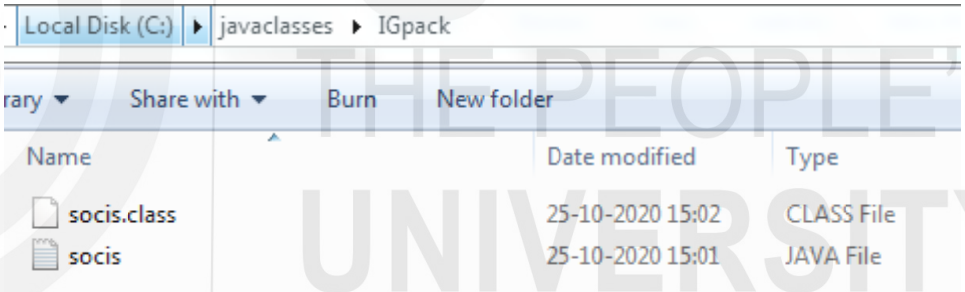


Figure 1: Directory structure for socis.java file

When you execute socis.class file through the java interpreter, it generates an error message: “Could not find or load main class socis” ,as shown in the figure 2.

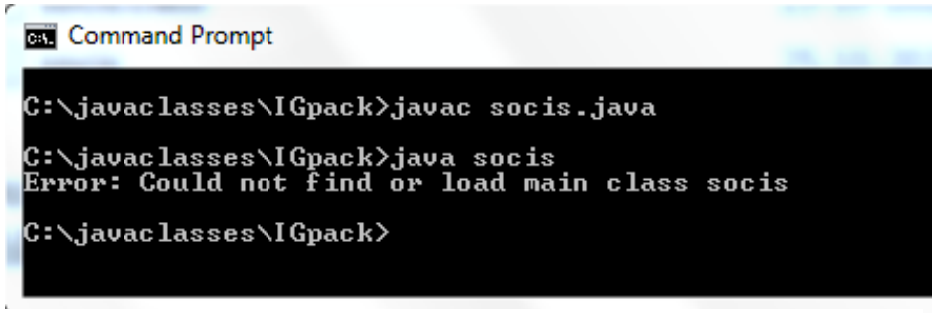


Figure 2: Command Prompt for execution of socis file

This is because the class is now stored in a package called IGpack. This class is not a simple class; it is created under the package. You can refer this class with package name like IGpack.socis. If you run with this reference, you will get an error message again. The reason behind these error messages is hidden in your CLASSPATH variable. In this example, IGpack is the name of a directory which is created by you. Actually, there does not exist any IGpack directory in the current working directory. For this reason, you need to set your development class hierarchy to classpath environment variable. CLASSPATH sets the top of the class hierarchy. Then you will be able to run your java file from any directory using the **java IGpack.socis** command. For example, if you are working on your source code in a **c:/javaclasses** directory then set your CLASSPATH to c:\javaclasses;

For classpath variable setting, you can refer environment setup for java development in Unit1 Block 1 of this course. Now, you can run the above example using **java IGpack.socis** command, program will run successfully and show the result as given in figure3.

Figure 3: Command Prompt screen for successful running of socis file

4.5 CREATING OWN PACKAGES

In the previous section, you have learned about the packages which comprise the built-in packages and user-defined packages. Built-in packages are part of the java development kit, and users develop User-defined packages to group the related classes, interfaces, and sub-packages.

In this section, you will learn how to create your own package and sub package within the existing packages, along with examples. Java provides package commands to you for creating your own package. It is written as the first statement in a java source file. There can be only one package statement in each source file. You can define more than one file in the same package statement. The general form of the package statement is given below:

```
package pkg_name;
```

Here, package is java keyword, and pkg_name is the name of the package. For example, the following statement creating a package called MyFirstPackage.

```
package MyFirstPackage;
```

You must remember that each package in Java has its unique name. When you define the package name, it must match with the directory name exactly. *You cannot rename a package without renaming the directory in which the classes are stored.*

You can also create a hierarchy of packages. For this, you can simply separate each package name from the other package name by using a period. The general form of multi-levelled package statement is shown below:

```
package pkg_name1[.pkg_name2[.pkg_name3]];
```

The following examples explain how to create your own packages:

Creating a package in java is very easy; simply include a package command followed by the name of the package as the first statement in java source file like the following:

```
package mypack;
public class student
{
   String studentId;
   String studentName;
}
```

Creation, compilation and execution of Java Package

In the next example, you will learn about **creation, compilation and execution of java package.**

Example-2: Create a class named 'socis' under the package 'IGpack1'. You can follow some simple steps for creation, compilation and execution of java package.

```
package IGpack1;  // creation of package
class socis
{
    public void method1()
    {
        System.out.println("method1 of socis");
    }
    public static void main(String args[])
    {
        socis obj = new socis();
        obj.method1();
    }
}
```

- For testing purpose, you can create a folder 'javaclasses' in C drive. You can write the above source code in a notepad and save as 'testpack.java' in "javaclasses" folder.
- If you are not using any IDE, you need to follow the syntax given below and compile 'testpack.java' file using the javac command:

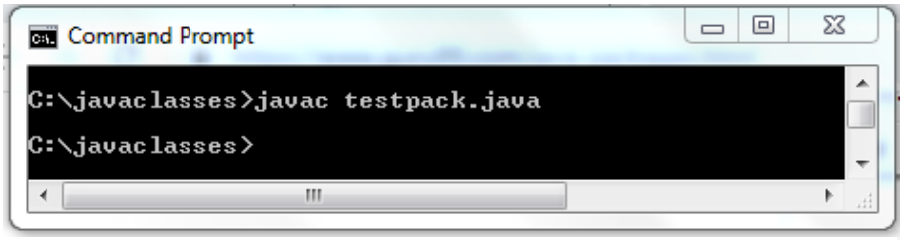


Figure 4: Command Prompt for compiling testpack Java program

- After successfully compilation of ‘testpack.java’, a class file ‘socis’ is created under your “javaclasses” folder.

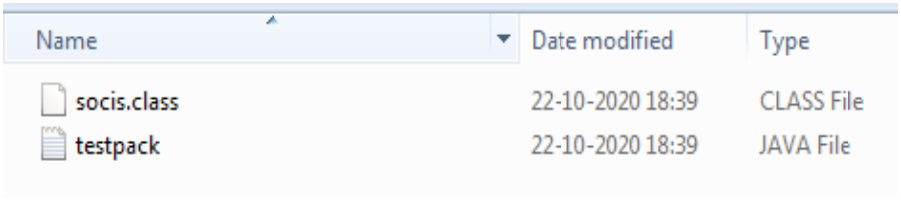


Figure 5: File Structure for testpack JAVA file

Compilation of Java Package program

- Now you can compile java program with package statement using the following command at the Command Prompt. This command dictate the compiler to create a package. The -d switch in the command specifies the destination where to put the generated class file. If you want to keep the package within the same directory, you can use “.” (dot). The "." operator represents the current working directory.

```
javac –d . testpack.java
```

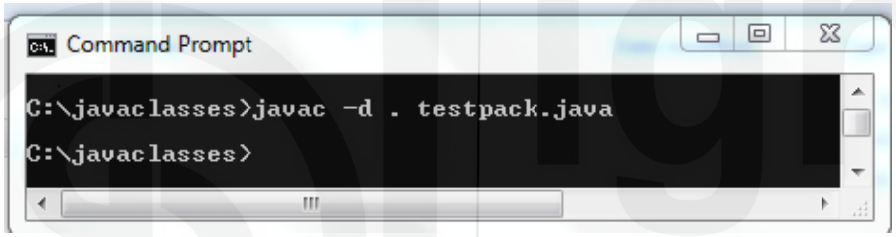


Figure 6: Command Prompt for compiling Java program to create package in current working directory

- After successfully compilation, a folder with the given package name is created in the specified destination and the compiled class file will be placed in that folder. You can see in following figure 7 “IGpack1” named package has been created.

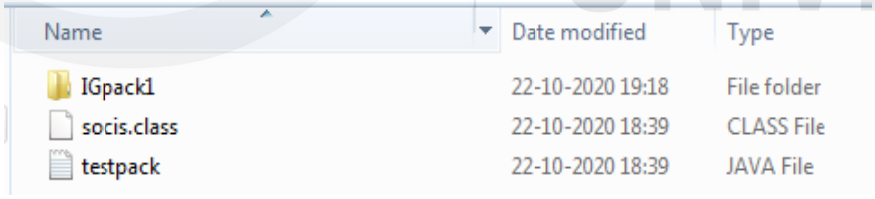


Figure 7: File Structure after package creation

- When you open the Java Package ‘IGpack1’ inside you will find “socis.class” file as shown in figure 8.

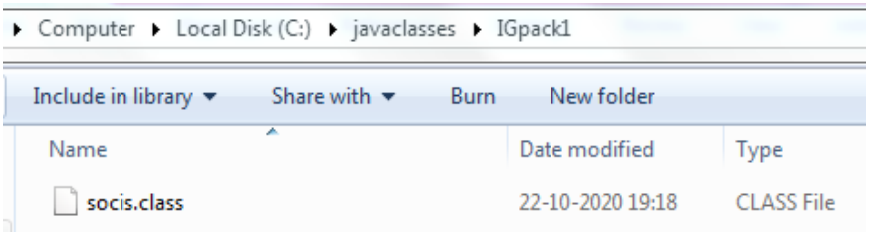


Figure 8: File Structure for socis.class file under IGpack1 package

Executing Java Package program

... When you execute socis.class file using the following command, it will display result like in figure 9.

```
java IGpack1.socis
```

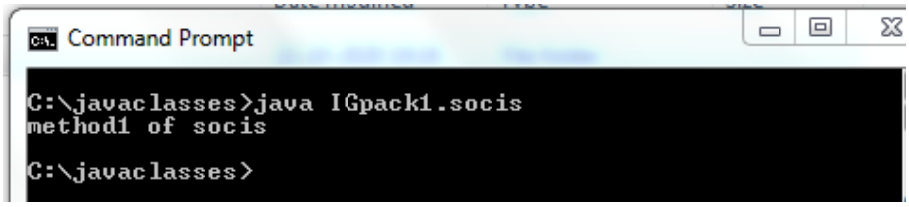


Figure 9: Output Screen for Executing Package in Java

In the above example, the current directory is "javaclasses". If you want to create a package in the parent directory, you can take your same file (i.e. testpack.java) and compile using the following command. See the figure 10 for details. In this case, the file will be saved in C Drive. The double dot(..) represents the parent directory.

```
javac –d .. testpack.java
```

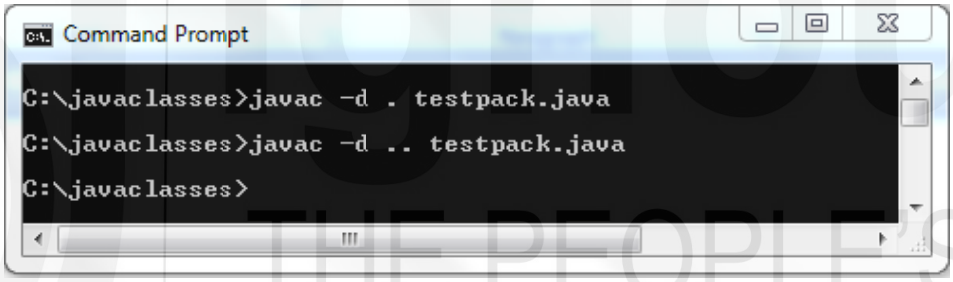


Figure 10: Command Prompt for compiling Java program to create package in parent directory

... You can see the following figure-11, package IGpack1 is created in C drive. When you click on this folder, it will show "socis.class" file.

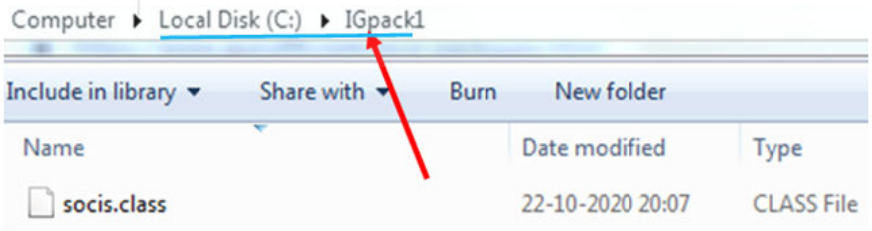


Figure 11: File Structure of creating socis.class file in parent directory

Creation of a sub package within your existing java package

You have seen the creation of own packages, compilation, and execution of Java Package program through the above example. If you want to create a sub package within your existing java package, you can take the above source code (i.e. testpack.java) with a slight modification in the package statement only as like the following:

```
package IGpack1.IGpack2;
```

Here, IGpack1 is parent package and IGpack2 is sub-package. Now compile the modified testpack.java file using the following command:

```
javac –d . testpack.java
```

After successfully compilation the file, sub package IGpack2 is created (shown in figure-12) under the existing package 'IGpack1' and it contains socis.class file as shown in figure-13.

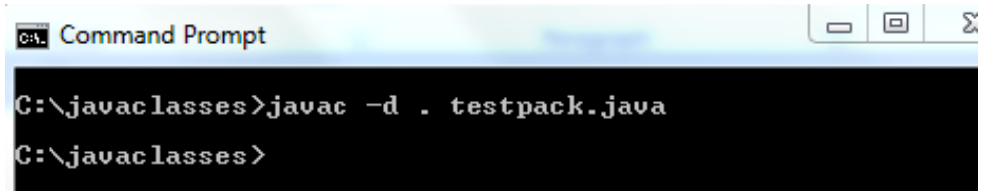


Figure 13: Compiling testpack.java file for creation of sub-package

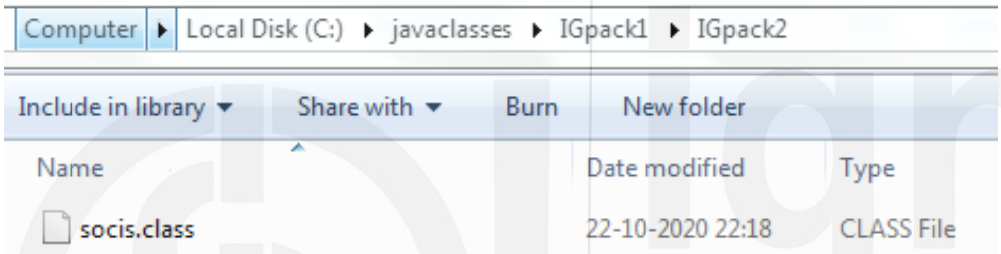


Figure 12: File structure for sub-package IGpack2

Execution of sub package

For executing the source code of sub-package, you can mention the fully qualified name of the class i.e. main package name followed by the sub-package name followed by the class name as the following:

```
java IGpack1.IGpack2.socis
```

This is how the package is executed and gives the output as "method1 of socis" from the code file as shown in figure-14

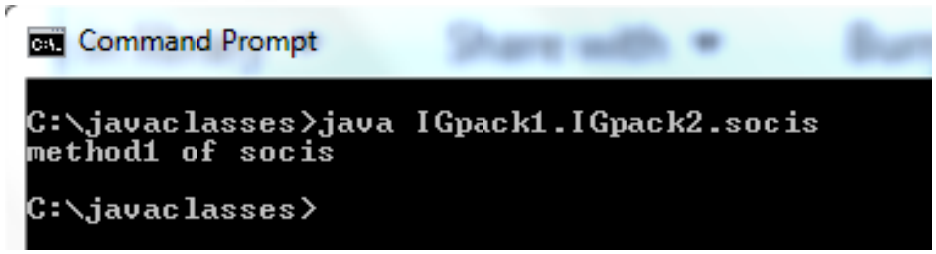


Figure 14: Command Prompt for executing of sub-package program

The same idea as discussed is used for creating packages using some IDE. To create your own Java package using NetBeans IDE, right click on the source packages and select New -> Java Package, as shown in figure 15.

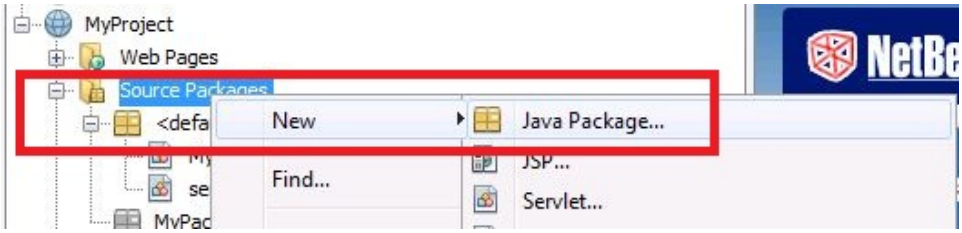


Figure 15: Creation of Package in NetBeans

After selection of the above procedure, you can enter name and location of package. you can see the figure 16.

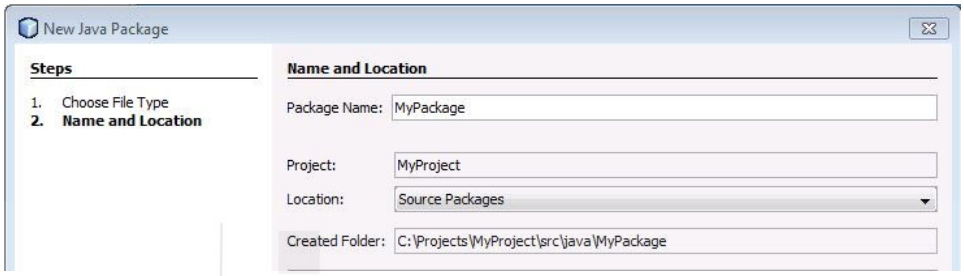


Figure 16: Name and Location of Package

Now, you can select the Finish button from the Button Panel.

Figure 17: Button Panel

This way you can create your own package. For creating java class under the package, right-click on the package, select New->Java Class. A package declaration is automatically inserted into each new source file it creates.

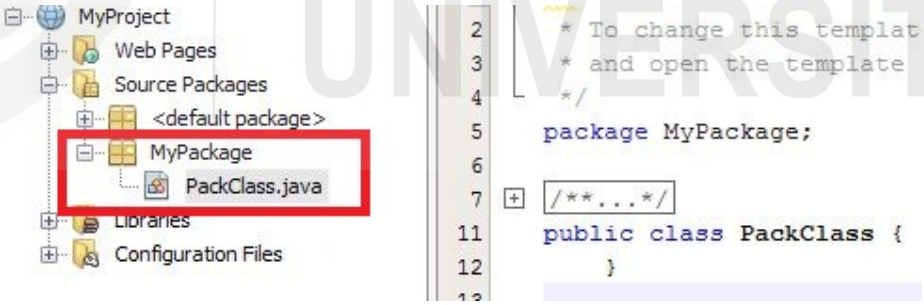


Figure 18:Creation of Java class under Package

To create a sub package in NetBeans, right-click on the parent package, select New-> Java Package. Also give a name to new sub package, you can see example figure-19:

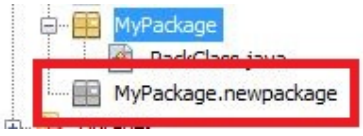


Figure 19: Creation of Sub-package in NetBeans

To create a Java sub class under the sub-package, right click on sub-package and select New-> Java Class and then give a name to new sub class. you can see in figure-20 for creation of 'subclass.java' file under the 'newpackage'.

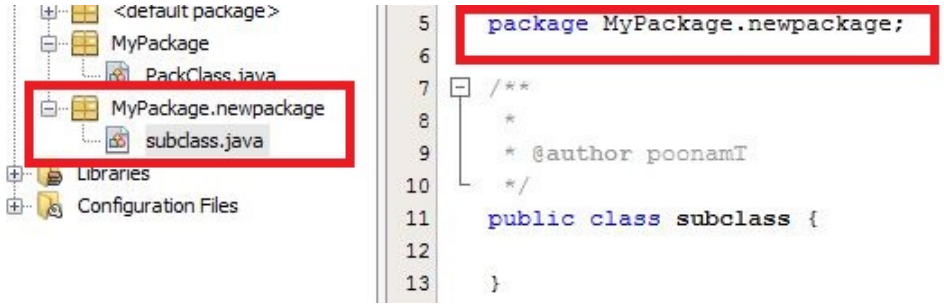


Figure 20: Creation of sub-class under the sub-pakage

This section clarified you the creation of your own package and sub package in java. You have also learnt the compilation as well as execution of package and sub-package program.

4.6 IMPORTING PACKAGES

This section describes how to import built-in packages as well as user-defined packages in your java source file.

Java provides import statement for including packages in java source file. Remember that in java programming, the import statement is written directly after the package statement (if it exists) and before the class definition. You can write more than one import statements. The general form of the import statement is as follows:

```
import package1[.package2].(classname|*);
```

You can import any specific package member or import all the types contained in a particular package.

For example,

```
import java.util.*;   // Import the whole package
import java.util.Calendar;  // Import a single class;
```

In the above example, java.util is a standard java package while Calendar is an abstract class of the java.util package. It provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc.

The above example shows you how to import built-in packages in java program. In a similar manner, you can use import statement to include your created packages in java source code. Let's discuss user-defined packages with the help of example.

Example: Create a class called Addition inside a package named MyPackage and save it as "Addition.java" in "javaclasses" folder which is already created in the previous section.

```
package MyPackage;
public class Addition
{
```

```
    public int add(int a, int b)
    {
        return a+b;
    }
}
```

Now let us see how to use this package in another program. Write the following code and save it as TestPackage.java

```
import MyPackage.Addition;
public class TestPackage
{
    public static void main(String args[])
    {
        Addition obj = new Addition();
        System.out.println(obj.add(100, 200));
    }
}
```

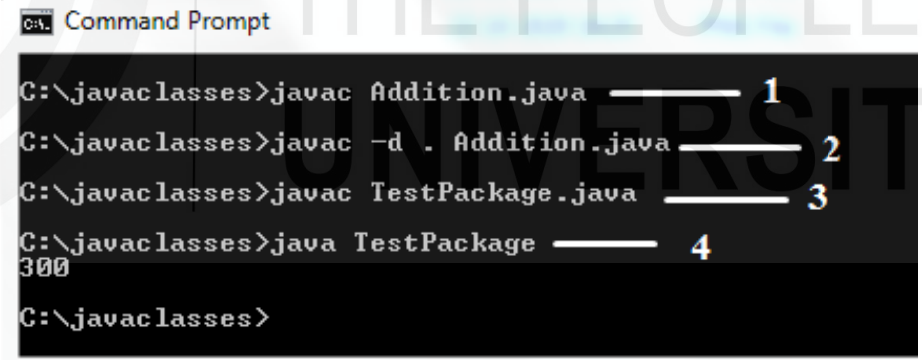
Now, compile both the files and run TestPackage file, using the following steps.Also the same process are shown in figure 15.

Step 1: Compile Addition.java file using **javac Addition.java** command at the command prompt and it creates Addition.class

Step 2: Now, you can use javac –d . Addition.java and it creates Addition.class file inside MyPackage

Step 3: Compile TestPackage.java file and it creates TestPackage.class

Step 4: Run TestPackage with java interpreter and it shows the addition of two numbers



The file structure of the above example is shown in following figure22:

Figure 21: Command Prompt for compiling and executing Java Program

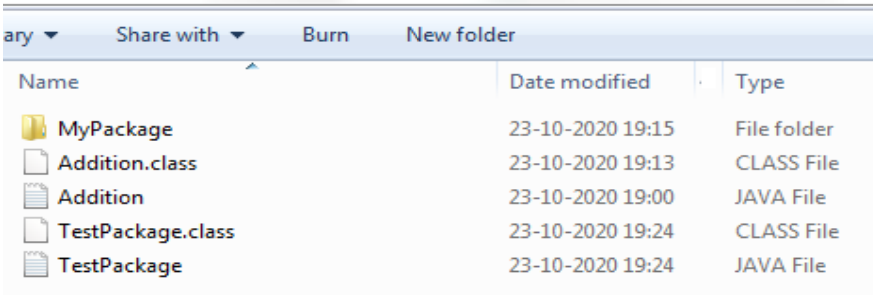


Figure 22: File structure of the above importing example

To use the class Addition, you have imported the package `MyPackage`. In the above program you have imported the package as `MyPackage.Addition in` TestPackage.java; this only imports the Addition class. However, if you have several classes inside package `MyPackage` , you can import the package like shown below.

```
import MyPackage.*;
```

☞ Check Your Progress-1

1. What are Packages? What are the advantages of packages? Write the name of the default package in JDK.

   _____
   _____
   _____
   _____

2. What are different types of packages in Java?

   _____
   _____
   _____
   _____

3. How to compile a source code of Java that is created as a package? Also, write command for executing Java Package program.

   _____
   _____
   _____
   _____

4. How packages are imported in java program?

   _____
   _____
   _____
   _____

4.7 BASICS OF INTERFACE IN JAVA

You have gone through the concept of classes in Unit 4 Block 1 of this course. In this section, you will learn about the basic features of interfaces in java programming languages.

Both Class and Interface are important concepts that build the foundation stone for java programming. In java, an interface is syntactically similar to classes, but they lack instance variables, and their methods are just declared without having any body.

It means that the Interface can contain only constants declaration, method declaration, default methods, static methods.

The following table shows some differences between classes and Interface:

	Class	**Interface**
Method	Abstract and concrete methods are defined.	Only abstract methods are defined
Member specifier	Members of a class can be defined as public, private, protected or default.	By default, all the members are public
attributes and behaviours	A class describes the attributes and behaviours of an object.	An interface describes behaviours that a class implements.

The interface contains one or more method declarations without their implementations. All the methods of an interface are abstract methods. The variables in an interface are public, static and final. Once it is defined, any number of classes can implement an interface. The Interface keyword is used to create an interface. An interface is a powerful mechanism for defining behaviour among unrelated classes.

As you already know, a class with the abstract keyword in its declaration is called abstract class. An abstract class can have both abstract methods, i.e. methods without a body, and regular methods with implementations. You have already studied the concept of abstract classes in Unit 2 Block 2 of this course. You can lay down some comparison between an interface and abstract class as under:

- Interface can be implemented using 'implements' keyword, whereas an abstract class can be inherited using the 'extends' keyword.
- Interface can only have public members, while abstract class can have any type of members like public, private etc.
- Variables declared in a Java interface is by default final, whereas an abstract class may contain non-final variables.
- Interface is completely abstract and cannot be instantiated. A Java abstract class also cannot be instantiated but can be invoked if a main() exists.
- An interface can extend another Java interface only, while an abstract class can extend another Java class and implement multiple Java interfaces.

The next sections give you more details on defining, implementing and applying interface. You will also know about Default Interface Methods, Issues of Multiple Interfaces, and the use of Static Methods in an Interface. In addition to this, you can extends an interface using *extends clause*. So, you can go thoroughly and run your program correctly.

4.8 DEFINING, IMPLEMENTING AND APPLYING INTERFACE

In the previous section, you have studied the basics of Interfaces. Now, this section will present you with how to define constants and methods in an interface as well as

how a class can implement this interface. The syntax of creating an interface is very close to that for creating a class with few exceptions. The most significant difference is that none of the methods in interface may have a body. The interface can be seen as a prototype for a class.

4.8.1 Defining an Interface

Interface keyword is used to declare an interface. The interface has the following general syntax:

<accessibility modifier> **interface** <interface-name> <**extends** interface-clause>

```
{
  // interface declaration
   <constant declarations>
  <method prototype declarations>
  <nested interface declarations>
}
```

Here, access modifier is either **public** or not used. When you do not mention any access modifier then it treats as default access modifier, and it is visible to other members of the package in which it is declared. When it is declared as public, the interface can be used by anyone. The interface-name is the name of interface, and it can be a valid identifier. The interface declaration part can contain member declarations which cover constant declarations, method prototype, nested interface declarations.

Here is a simple example of declaring an interface:

```
public interface SimpleInterface
{
   public String xyz = "IGNOU";
   public void method1();
}
```

As you can see, an interface is declared by using the Java **interface** keyword. The above interface example contains one variable and one method. The variable can be accessed directly from the interface, like this:

```
System.out.println(SimpleInterface.xyz);
```

Constants in Interfaces

You can define constant in an interface. The constants declaration is considered to be public, static and final. An interface constant can be accessed by any user/client (i.e. interface or a class) using its fully qualified name irrespective of whether the user/client implements or extends its interface. However, if a client is an interface that extends this interface or a class that implements this interface then the client can access constants directly without using the fully qualified name.

Interface Methods

In java an interface contains one or more method declarations. All methods in an interface are public, even if you leave out the public keyword in the method declaration. As mentioned earlier, the interface cannot specify any implementation for

these methods. It is the job of the classes implementing the interface to specify an implementation for the methods of the interface.

For defining Interface in Java using Netbeans IDE, start NetBeans software and create your project as JavaApplication and enter name of the project and select Finish button from the button panel. Now, expand Source Packages and right-click on the your own defined Package and select New->Java Interface.

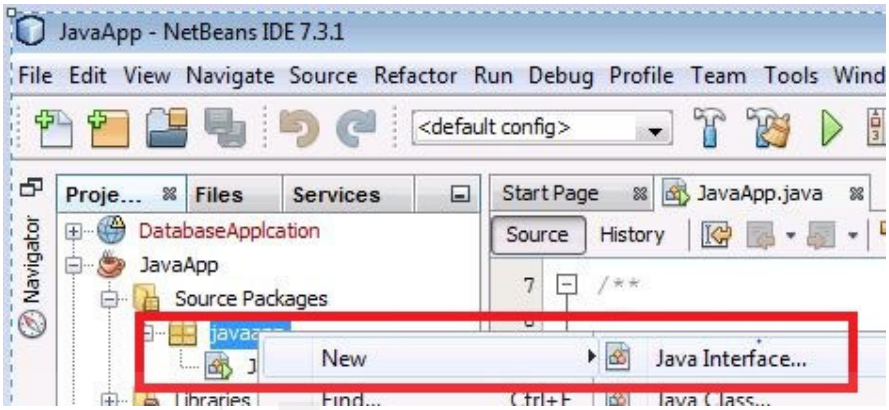


Figure 23:Creation of Interface in NetBeans

Now, Enter name and location of the interface.

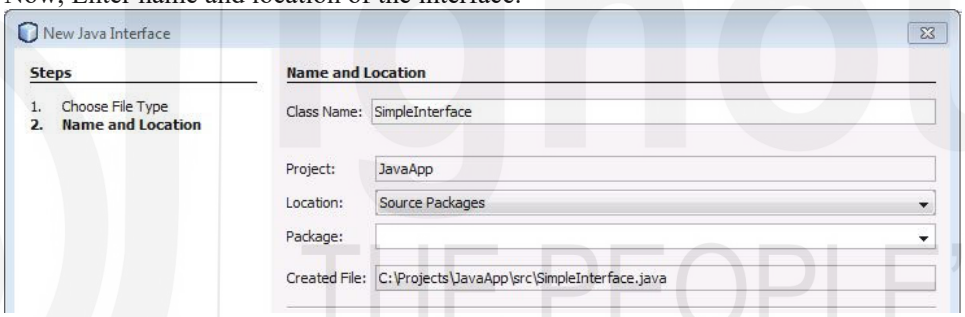


Figure 24:Name and Location of Interface

This way you can create an Interface in Java using NetBeans.

4.8.2 Implementing Interfaces

As yet you know that the methods in the interface are declared with an empty body. Once it is defined, a class can implement an interface for accessing these methods.It is compulsory for a class that implements an interface to all the methods declared in the interface. Any number of interfaces can be implemented by a class using the **implements** keyword. The syntax for implementing interfaces is given below.

<*access specifier*> class [extends *superclass-name*]
[implements *interface-name*[,*interface-name*…]]
{
// class body
}

For example the following program shows you how a class implements an interface. The two methods are defined in interface - ImpleInterface, and the class has to implement all the methods declared in the ImpleInterface .

interface ImpleInterface

```
{
  public void method1();
  public void method2();
}
class Impleclass implements ImpleInterface
{
    public void method1()
    {
        System.out.println("implementation of method1");
    }
    public void method2()
    {
        System.out.println("implementation of method2");
    }
    public static void main(String arg[])
    {
        ImpleInterface imp = new Impleclass();
        imp.method1();
        imp.method2();
    }
}
```

Now, you can compile and run the program. After successfully running, it will give the output as like the following figure 25:

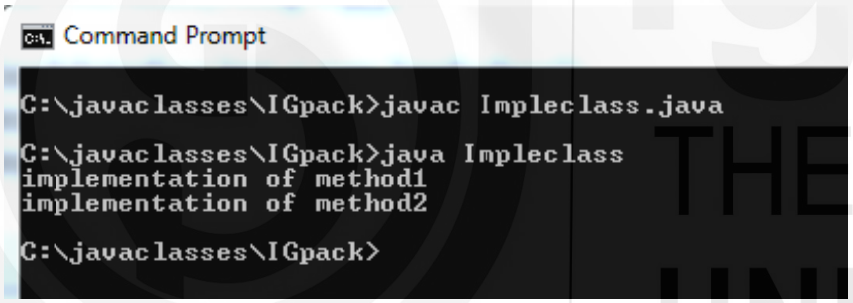


Figure 25: Command Prompt for compiling and executing of Implementing Interface Example

The file structure of the above program is shown in figure 26:

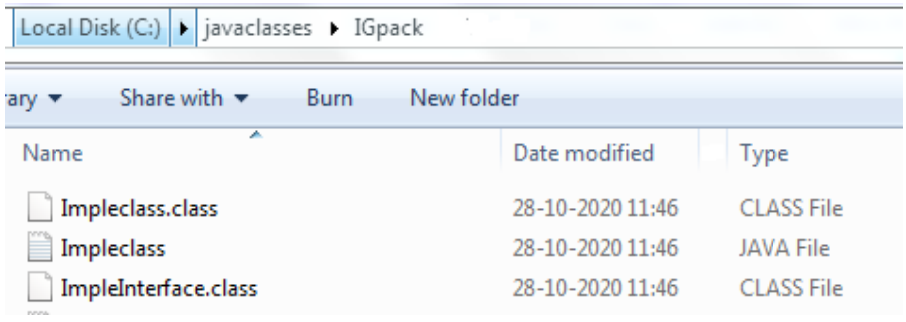


Figure 26: File structure for Implementing Interface Example

Now you can implement interface using the above example in NetBeans. For this you can write the code in the created

Interface such as 'SimpleInterface.java' like the following Figure-27:

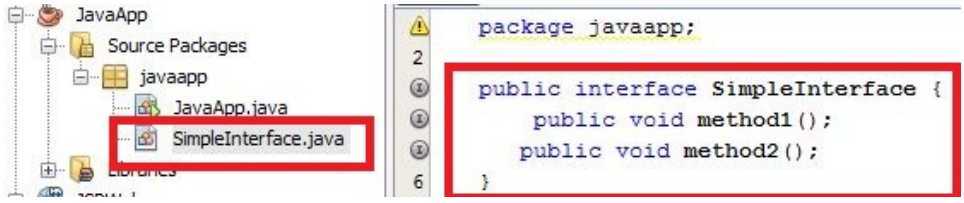


Figure 27: Example of Interface in NetBeans

Now implement the Interface named 'SimpleInterface' in Java Class like the following Figure-28:

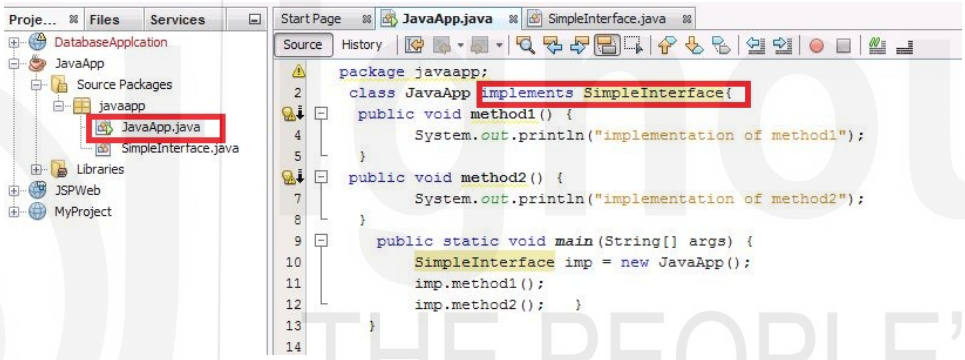


Figure 28: Implementaion of Interface using NetBeans

Now, you can compile the java class as well as Interface using F9 and right-click on the Java Class program and select 'Run File' or Shift+F6 to run the program and after successfully running, it will give the output as like the following figure-29

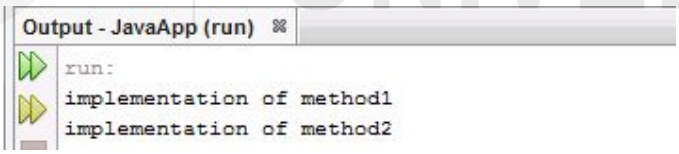


Figure 29: Output of the above Java Program

Accessing Implementations through Interface References

You have seen in the above example, there are two methods in the interface which are implemented by the class. In the Java program, you would have noticed that variable **imp** is declared to be of the interface type **ImpleInterface**, it is assigned an instance of **Impleclass**. The variable imp can be used to access method1() and method2(). You can access the methods declared by its interface declaration using interface reference variable. You cannot use interface reference variable to access non-interface method.

4.8.3 Applying Interfaces

In the preceding sections, you have been explained about the declaration and implementation of interface. This section will explain you on how to use both of them with a real example.

Following example will help you in understanding the concept of declaring and implementing interface by the class. In this example, an interface named Area is defined which contains constant and method which implemented by two classes such as Rectangle & Circle and they will compute area as per their shape.

// **Example for Implementing Interfaces**
//**InterfaceTest.java**

```
interface Area                              //interface defined
{
        final static float pi=3.14f;        // constant declare
        float compute(float x, float y);    // method declaration
}

class Rectangle implements Area          //interface Area implemented
{   public float compute(float x, float y)
        { return (x * y); }
}
 class Circle implements Area               // interface Area implementation
{ public float compute(float x, float y)
        { return (pi * x * y); }
}

class InterfaceTest
{
     public static void main(String args[])
     {
        Rectangle rect = new Rectangle();
        Circle cir  = new Circle();
        Area area;                                  // interface object
        area=rect;
        System.out.println("Area of Rectangle =" + area.compute(10,20));
        area=cir;
        System.out.println("Area of Circle =" + area.compute(10,20));
     }
}
```

After successfully compilation, it will show output as shown in following figure-30:

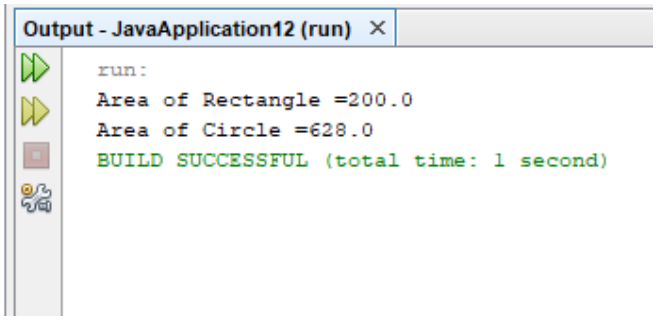


Figure 30: Command Prompt showing output of above Example

The file structure of the above example is shown in following figure 31:.

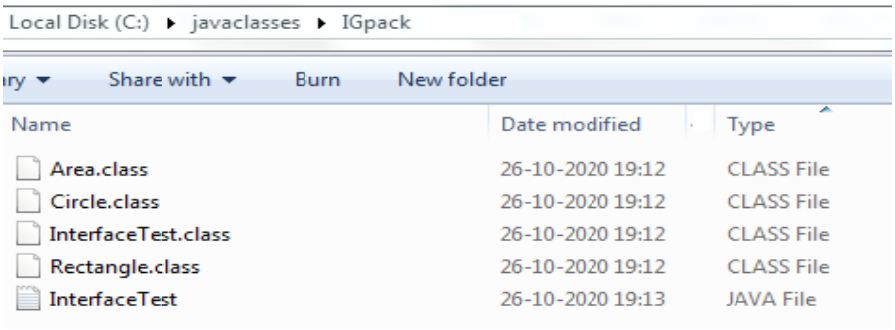


Figure 31: File structure of the above example

4.9 EXTENDING INTERFACE

An interface can extend other interface by using the keyword **extends**. The syntax is similar to inheriting classes. When a class implements an interface that inherits other interface then the class must implement all methods defined within the interface inheritance chain.

For Example: The following program demonstrates the use of extending interface:

```
interface X1 {
                    void method1();
                    void method2();
                  }
// interface X2 includes method1() and method2() of interface X1

interface X2 extends X1
{ void method3();  }   // adds method3()

// Class must implement all the methods of X1 and X2
class TestClass implements X2
{
    public void method1()
                    { System.out.println("method 1"); }
    public void method2()
                    { System.out.println("method 2"); }
    public void method3()
                    { System.out.println("method 3"); }

}
class InterfaceExtend
{
    public static void main(String arg[])
    {
        TestClass obj = new TestClass();
        obj. method1();
```

```
        obj. method2();
        obj. method3();
    }
}
```

After compiling and running the above program, it will shows output as shown in following figure-32:

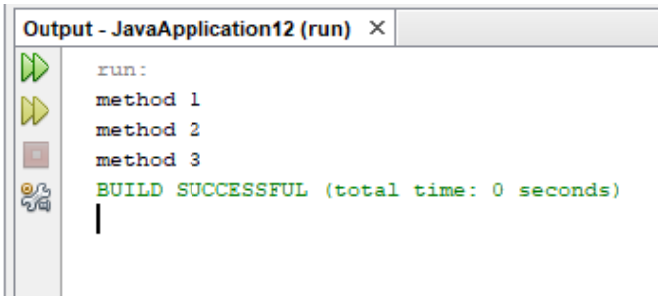


Figure 32: Output screen for Extending Interface Example

The file structure of the above Extending Interface Example is shown in figure-33.

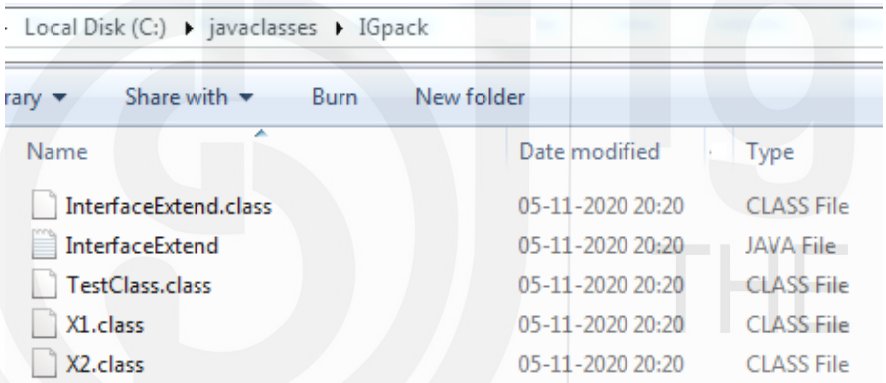


Figure 33: File Structure for Extending Interface Example

☞ Check Your Progress-2

1. Explain the concept of an interface.

   _____
   _____
   _____
   _____

2. Can you extend interfaces in Java? Explain with example.

   _____
   _____
   _____
   _____

3. Which of these field declarations are valid for an interface?

Select the two correct answers.

a) public static int area = 40;

b) private final static int area = 40;

c) final static int area = 40;

d) int area;

4. What is the difference between interface and abstract class ?

5. Write a java program to find an area of rectangle using Interface. For this program, create a class Rectangle that implements an interface 'FindArea' which contains an abstract method 'Area'. You can specify the length and width values in a program.

4.10 DEFAULT INTERFACE METHODS

In the preceding sections, you have learned that all the methods of interfaces are public and abstract by default. In this section, you will learn how to declare and use default methods. You can also define default as well as static methods in the interface. In the section 4.12, you will find about the use of static methods in the interface.

If you want to add new methods in the interface, it will require change in all the implementing classes. The solution of this situation is default method. The default method allows the java developers to add new methods in the existing interfaces without affecting the classes that implement these interfaces. It is not mandatory to provide implementation for default methods of interface by the implementing class. You can override these methods in the classes that implement these interfaces.

Default methods are those methods that are defined in the interface with the keyword default. There is no need to specify the public modifier in default methods, it is implicitly defined public. For example:

```
interface defaultInterface
{
    default void method1()
    {
        System.out.println("This is default method");
    }
}
```

Example: The following program demonstrates the use of default method in Interface.

In this program, default method named method2() and another regular method named method1() are defined in an interface which are being called in the main() method of the Implementation Class 'example'.

```
interface defaultInterface
{
    public void method1();
    default void method2()      // declaration of default method
    {
        System.out.println("This is default method");
    }
}

// Implementation Class
class example implements defaultInterface
{
    public void method1()
    {
        System.out.println("This is regular method");
    }

     public static void main(String arg[])
    {
        example obj = new example();
        obj.method1();
        obj.method2();  //calling the default method of interface

    }
}
```

When you compile and run the above java program, it will show output as the following figure 34.

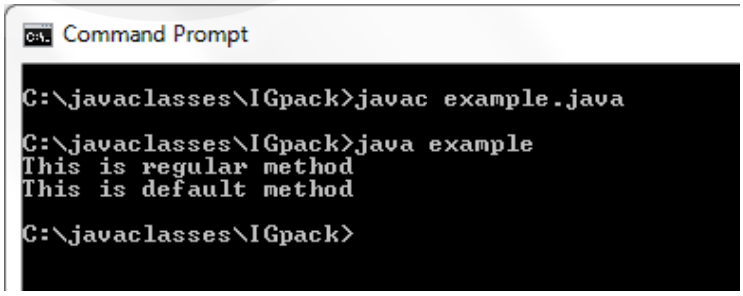


Figure 34: Command Prompt Screen for example of default method in Interface.

The file structure of the above example program is as follows:

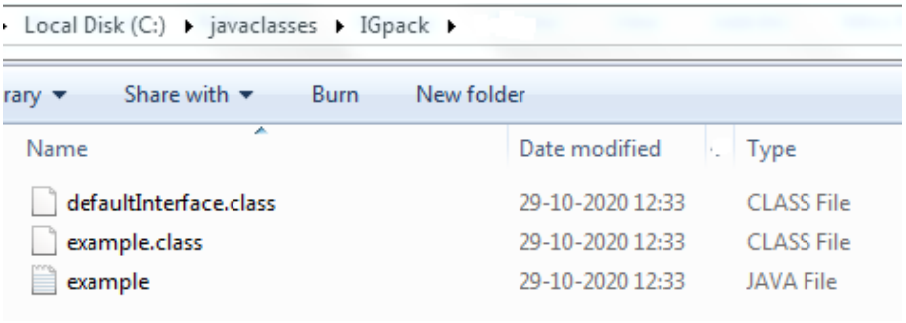


Figure 35: File structure of the above example

4.11 ISSUES OF MULTIPLE INTERFACES

You have been gone through about the inheritance in Unit 1 Block 2 of this course. You already know that Multiple Inheritance is a feature of object oriented programming where a class can inherit properties of more than one parent class. *Java is object oriented language but does not support multiple inheritances* in *the case of class.* On the other hand, there is another mechanism for achieving multiple inheritances through the Interfaces because a class can implement multiple interfaces in java.

Consider the following example through which you can understand the issues of multiple inheritances.

```
// First Parent class
class P1
{
   void method1()
   {
      System.out.println("method of Class P1");
   }
}

// Second Parent Class
class P2
{
   void method1()
   {
      System.out.println("method of Class P2");
   }
}

// Error : Demo class is inheriting from multiple classes
class Demo extends P1, P2
{
  public static void main(String args[])
  {
     Demo d = new Demo();
     d.method1();
  }
}
```

Output of the above program is compilation error as shown in figure-36:

Figure 36: Output of the above program

You have seen in the above program, both the parent class P1 and P2 have methods with the same signature. Upon calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

Java provides solution for the above problem in the form of multiple interfaces and class can implement two or more interfaces.

For example, if both the implemented interfaces contain default methods with the same method signature then implementing class should explicitly specify which default method is to be used or it should override the default method.

```
interface P1
{
     default void display()
     {
      System.out.println("Default method: Interface P1");
     }
}

interface P2
{
    default void display()
    {
     System.out.println("Default method: Interface P2");
    }
}

// Implementation class
class TestMulti implements P1, P2
{
   // Overriding default display() method
   public void display()
    {
      /* use super keyword to call the display method of P1 interface */
      P1.super.display();

      /* use super keyword to call the display method of P2 interface */
      P2.super.display();
    }

   public static void main(String args[])
   {
```

```
        TestMulti t = new TestMulti();
        t.display();
    }
}
```

When you compile the above Java program, it will show output looks like the following figure 37

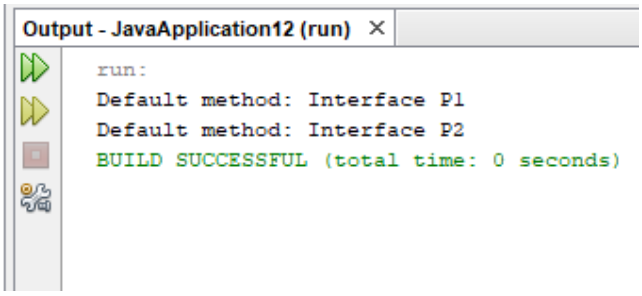


Figure 37: Command Prompt Screen for test multiple inheritance through interfaces program example

4.12 USE OF STATIC METHODS IN AN INTERFACE

In the preceding section, you have gone through the default method in the interface. This section will give you a demonstration of the use of static methods in an interface.

As the name static indicates that once it is defined, you can use it but cannot change it. When you use static method in an interface then calling class cannot change in static method. You can declare it with the static keyword at the beginning of the method signature and they provide an implementation.

The interface static method is similar to the default method of interface but it cannot be overridden in implementation Classes. The static method is dissimilar to other regular methods which are defined in the Interface; the static method contains the complete definition of the function and since the definition is completed and the said method becomes static. Therefore, these methods cannot be overridden in the implementation class. Interface name should be instantiated with it(static method) to use a static method as it is a part of the Interface only.

Example: The following Java program demonstrates the use of the static method in Interface.

```
interface staticInterface
{
    //regular method
     public void method1();

    // default method
    default void method2()
    { System.out.println("The default method is define in interface"); }
```

```
    // static method
    static void method3()
    { System.out.println("This is Static Method");   }
}
// Implementation Class
class StaticExample implements staticInterface
{
    public void method1()
    {  System.out.println("This is regular method");  }

    public static void main(String arg[])
    {
        staticexample obj = new staticexample();

        // Calling the abstract method of interface
        obj.method1();

        // Calling the default method of interface
        obj.method2();

        // Calling the static method of interface
        staticInterface.method3();
    }
}
```

When you compile and run the above Java Program, it will show output as like the following figure 38:

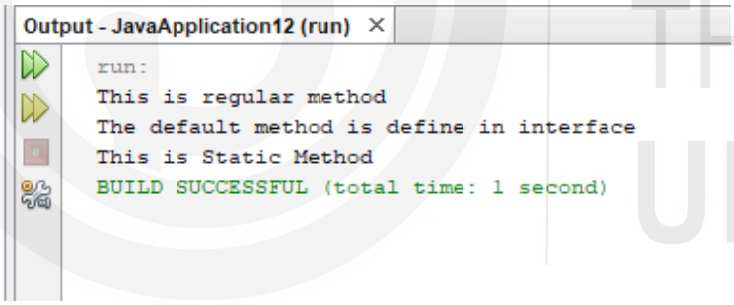


Figure 38: Output Screen for use of Static method in Interface

The file structure of the above java program is looks like the following figure-39:

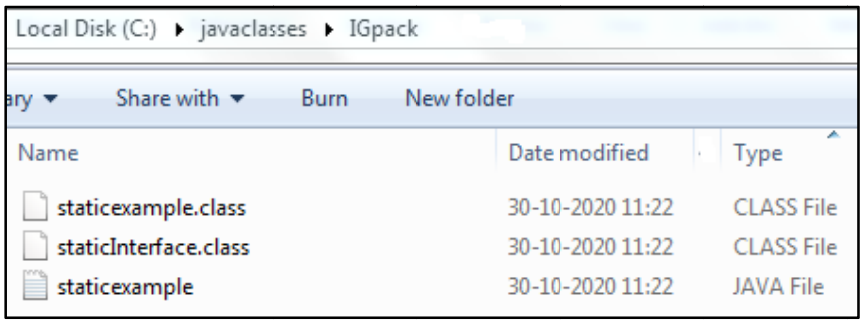


Figure 39: File structure of the above java program

☞ Check Your Progress-3

1. What are the default methods in interface?

2. What is the use of static methods in interfaces?

3. What is the difference between static and default methods in interfaces?

4. What is incorrect with the following interface?

```
public interface A
{
  void method1(int avalue)
   {
     System.out.println("Hello! Student");
   }
}
```

5. Write a Java Program to add two numbers using static method in interface. You can specify input values in a program.

4.13 SUMMARY

In this unit you have learned mainly two important features namely packages and interfaces. The role of packages is to make groups of related classes and interfaces. So, a package is collection of classes, interfaces and sub packages. The package is a grouping mechanism in which related class files are grouped and made available to other applications and other parts of the same application. It provides a way to organize related classes and interfaces to avoid naming conflicts. You can put classes that you developed in packages and distribute the package to others. Java language itself comes with a rich set of packages which are standard packages. This unit make yo able to differentiate between built-in packages and user-defined package. You are also able to create your own package and importing packages in Java program. This unit is also explained to you the interfaces. Now you are able to create interface and use them in a class. With interfaces, you can obtain the effect of multiple inheritances. With the introduction of default methods, you can add additional features to the interfaces without affecting the end-user classes. You have also learned about the use of static methods in interface. Now you are fully able to use packages and interfaces in Java program.

4.14 SOLUTIONS/ANSWER TO CHECK YOUR PROGRESS

☞ Check your Progress-1

1) The package is a grouping mechanism in which related class files are grouped and made available to other applications and other parts of the same application. So, the package is a collection of related classes and interfaces. The package declaration should be the first statement in a Java class.

   Packages offer several advantages like the following:

   - You can define their own packages to bundle a group of classes and interfaces, etc.
   - Using Package, you can avoid naming conflicts.
   - It is a good practice to group related classes implemented by you so that a programmer can easily determine the related classes, interfaces, enumerations, and annotations.
   - Using packages, it is easier to provide access control and locate the related classes.

     The JDK includes the default package name as java.lang package. This package provides the fundamental classes that are necessary to design a basic Java program. The important classes are Object, which is the root of the class hierarchy and Class, instances of which represent classes at run time.

2) Two types of packages in Java are Standard or Built-in packages and User-defined packages. Built-in packages are those packages which Java API provides to simplify the task of Java programmer. The packages which the users create are called as User-defined package.

3) The Java command for compiling Java source code using the following syntax:

javac –d . filename.java

For example: javac –d . student.java

For executing package programs, the fully qualified name of a class is <package>.<classname>. For example: java javapackagename.student

4) Java has an import keyword, which is used to access java package and its classes into the java program. You can import any specific package member or import all the types contained in a particular package. The import statement is written directly after the package statement (if it exists). You can write more than one import statements.

☞ Check your Progress-2

1) Interface can contain only constants declaration, method declaration, default methods, static methods and nested types. The interface contains one or more method declarations without their implementations. Once it is defined, any numbers of classes can implement an interface. The Interface keyword is used to create an interface. An interface is a powerful mechanism for defining behaviour among unrelated classes.

2) An interface can extend other interface using the keyword extends. The syntax is similar to inheriting classes. When a class implements an interface that inherits other interface, then the class must implement all methods defined within the interface inheritance chain. For example, please refer to section 4.9 of this unit.

3) The correct answers are (a) and (c). Fields in interfaces declare named constants and are always public, static and final. None of these modifiers are mandatory in a constant declaration. All named constants must be explicitly initialized in the declaration.

4) The difference between interfaces and abstract classes are as under:

- Interface can be implemented using ‘implements’ keyword whereas abstract class can be inherited using ‘extends’ keyword.
- Interface can only have public members, while abstract class can have any type of members like public, private etc.
- Variables declared in a Java interface is by default final, whereas an abstract class may contain non-final variables.
- Interface is completely abstract and cannot be instantiated; A Java abstract class also cannot be instantiated but can be invoked if a main() exists.
- An interface can extend another Java interface only, while an abstract class can extend another Java class and implement multiple Java interfaces.

5)

```
// create an interface
interface FindInterface
{
        void Area(int length, int width);
}

// implement the FindInterface interface
```

```
class Rectangle implements FindInterface
{
        // implementation of abstract method of interface
          public void Area(int length, int width)
        {
                System.out.println("Area of the rectangle = " + (length * width));
        }
}
class Test
{
       public static void main(String[] args)
       {
            // create an object
              Rectangle r = new Rectangle();
              r.Area(8, 9);
        }
}
```

☞ Check your Progress-3

1) When you want to add new functionality to an existing interface then you can use default methods. For creating default methods, use the default keyword and add the definition for the method. Default methods can be provided to an interface without affecting implementing classes as it includes an implementation. If each added method in an interface is defined with implementation, then no implementing class is affected. An implementing class can override the default implementation provided by the interface.
2) You can have static methods in an interface (with body), and it cannot be override in implementation Classes. If your interface has a static method you need to call it using the name of the interface, just like static methods of a class.
3) When you want to add new functionality to an existing interface without breaking the rules then you can use static methods. Static methods in an interface are similar to default methods, but they cannot be overridden in implementation Classes. The static methods are declared by using the static keyword. You can access static methods by using the interface name.
4) It has a method implementation in it. Only default and static methods have implementations.

5)

```
interface staticInterface
{
     // static method
         static void add()
         {
              int x, y, z;
              x=2;
              y=3;
              z=x+y;
              System.out.println("Static Method in an Interface");
              System.out.println("X + Y =" + z);
         }
 }
// Implementation Class
class staticexample implements staticInterface
{
```

```
    public static void main(String arg[])
    {
        // Calling the static method of interface
        staticInterface.add();

    }
}
```

4.15 REFERENCES/FURTHER READING

- Herbert Schildt "Java The Complete Reference", McGraw-Hill,2017.
- Horstmann, Cay S., and Gary Cornell, " *Core Java: Advanced Features*" Vol. 2. Pearson Education, 2013.
- Prasanalakshmi, "*Advanced Java Programming*", CBS Publishers 2015
- Sagayaraj, Denis, Karthik and Gajalakshmi ,"*Java Programming – for Core and Advanced Users*", Universities Press 2018 .
- Khalid A. Mughal and Rolf W. Rasmussen, " A programer's Guide to Java Certification", Addison-Wesley Professional, 2003.
- https://www.w3schools.com/java/
- https://docs.oracle.com/javase/tutorial/java/package/packages.html
- The Complete Reference Java – Patrick Naughton and Herbert Schildt