GCA S4 14

Programming in Java

SEMESTER - IV

BACHELOR OF COMPUTER APPLICATION

BLOCK - 2

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakati Mahanta, Professor, Gauhati University
Prof. (Retd.) Pranhari Talukdar, Professor, Gauhati University
Dr. Jyotiprokash Goswami, Associate Professor, Assam Engineering College

Course Co-ordinator (s) : Dr. Tapashi Kashyap Das, Assistant Professor, KKHSOU
Ms. Sruti Sruba Bharali, Assistant Professor, KHSOU

SLM Preparation Team

UNITS	CONTRIBUTORS
7, 8, 9, 10, 11, 12	**Mr. Pranjit Baruah**, KVK, Barpeta

Editorial Team

Content : **Dr. Minakshi Gogoi,** GIMT
Language : **Prof. Robin Goswami,** Retd. Professor, Cotton College
Structure, Format & Graphics : **Ms. Sruti Sruba Bharali,** KKHSOU

January, 2019

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-Non Commercial-Share Alike 4.0 License (international): http://creativecommons.org/licenses/by-nc-sa/4.0/

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

Head Office: Patgaon, Rani Gate, Guwahati - 781017; Web: www.kkhsou.in/web_new
City Ofice: Housefed Complex, Dispur, Guwahati - 781006

The University acknowledges with thanks the financial sipport provided by the **Distance Education Bureau, UGC, New Delhi**, for the preparation of this study material.

CONTENTS

		Pages
UNIT 7 :	**Packages and Interfaces**	**129-145**
	Packages: Defining a package, Understanding class path, Importing Packages, Interfaces: Defining an Interface, Implementing interfaces, Applying Interfaces, Variable in interfaces	
UNIT 8 :	**Exception Handling**	**146-159**
	Exception handling fundamentals, Exception types, Uncaught exceptions. Using try and catch JAVA's build-in exceptions, User defined exception subclasses	
UNIT 9 :	**I/O in Java**	**160-178**
	I/O Basics: Streams, The stream classes, predefined streams, Reading console input, Writing console output, Reading and writing files	
UNIT 10:	**Introduction to Applets**	**179-200**
	Applets and the World Wide Web, The Applet Class, Applets and HTML , The Life Cycle of an Applet, Responding to Events, Using Window Components, Adding Audio and Animation	
UNIT 11 :	**AWT**	**201-214**
	AWT Basics, AWT Components, Containers, Event Handling, Application and Menus	
UNIT 12:	**Swings**	**215-228**
	Swings: Introduction, Swing Components, Event Handling, Display text and image in a window, Layout manager.	

BLOCK INTRODUCTION

This is the second block of the course '***Programming in Java***'. After completing this block, learners will be able to write Java programs using different packages and interfaces. Learners will also be able to write Java programs using Applets, AWT and Swings.

This block comprises the following six units :

Unit - 7 introduces us to packages and interfaces. Concepts like defining and importing packages are covered in this unit. The unit also gives an introduction to defining interfaces and how to apply interfaces.

Unit - 8 introduces us to exception handling. Topics like build in exception and user defined exception classes are discussed in this unit.

Unit - 9 describes different I/O streams in Java. Reading console input, writing console output along with reading and writing files are covered in this unit.

Unit - 10 introduces us to Java applets. Topics like applet class, life cycle of an applet, adding audio and animation and responding to events are described in this unit.

Unit - 11 deals with AWT in Java. AWT basic and components along with event handling are discussed in this unit. Application and menus are also covered in this unit.

Unit - 12 gives us an introduction to swings. Concepts related to swing components and displaying text and images in window along with layout manager are described in this unit.

UNIT 7: PACKAGES AND INTERFACES

UNIT STRUCTURE

7.1 Learning Objectives

7.2 Introduction

7.3 Packages

- 7.3.1 Built-in Packages
- 7.3.2 User Defined Packages
- 7.3.3 Creating Your Own Packages

7.4 Understanding Classpath

- 7.4.1 Setting Java Classpath is windows

7.5 Importing Packages

7.6 Interfaces

7.7 Implementing Interfaces

7.8 Variable in Interface

7.9 Let Us Sum Up

7.10 Further Readings

7.11 Answers to Check Your Progress

7.12 Model Questions

7.1 LEARNING OBJECTIVES

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

- describe packages and interfaces in Java
- describe different types of packages,
- create your own package,
- use package and classpath
- define and implement interface

7.2 INTRODUCTION

This is the first unit of the second block. In the previous block, we have learnt about defining a class in Java. Defining arrays, String and Vectors in Java has also been covered in the first block. We have also learnt about inheritance in Java. Packages are group of related classes and interfaces together into a single unit. That means, packages act as “containers” for

classes. The benefits of organising classes into packages are reusability. Packages also provide a way for separating “design” from coding. Like a header file in C language packages are also required to import at the beginning of a Java program.

Using interface you can achieve the concept of multiple inheritance in java as Java does not support multiple inheritance. In the next unit we will learn about exception handling.

7.3 PACKAGES

A package is a collection of classes, interfaces and sub-packages. That means, packages act as “containers” for classes. There are two types of packages in java, they are predefined or built-in or core packages and user or custom defined packages.

Let us see some of the features of packages :

- Reusability: Reusability of code is one of the most important requirements in the software industry. Reusability saves time, effort and also ensures consistency. A class once developed can be reused in any number of programs.
- Easy to locate files.
- In real life situation there may arise scenarios where we need to define files of the same name. This may lead to “name-space collisions”. Packages are a way of avoiding “name-space collisions”.

7.3.1 Built-in Packages

Built-in packages are those which are supplied as a part of JDK (Java Development Kit) to simplify the task of java programming. Core packages of java starts with java. For example: java.lang.*. We have nine predefined packages which are described in table 7.1.

Table 7.1: Some Built-in Packages

Package Name	Package description
java.lang.*	This is the package which is by default imported for each and every java program. This package is used for convertion of data from string to other data, displaying the result on to the console etc.
java.io.*	This package is used for file handling applications, such as, opening the file in read or write mode, reading or writing the data, etc.
java.awt.*	This package is used for developing GUI (Graphic Unit Interface) such as buttons, check boxes, scroll boxes, etc.
java.awt.event.*	This is the sub package of awt package. It is used for providing the functionality to GUI components. For example when a button is clicked or when check box is checked.
java.applet.*	This package is used for developing browser oriented applications. An applet is a java program which runs in the browser.
java.net.*	This package is used for developing client server applications.
java.util.*	This package contains various classes and interfaces which improves the performance of different applications.
java.text.*	This package is used for formatting date and time.
java.lang.reflect.*	It is sub package of lang package. It is generally used to study runtime information about the class or interface.
java.sql.*	This package is used for executing various operations on data base.

7.3.2 User Defined Packages

A user defined package is developed by programmers to simplify the task of the java programming to keep set of classes, interfaces and sub packages which are commonly used. If any class or interface is commonly used by many java programmer that class or interface must be placed in packages.

Creating package is very easy, just include a package command as the first statement in a Java source file. The general form of the package statement is:

package <package name>;

The following statement creates a package called MyFirstPackage
package MyFirstPackage;

To store packages Java uses file system directories. For example, the .class files of any classes declare as a part of MyPackage must be stored in a directory called MyPackage and directory name must match the package name exactly.

We can create a hierarchy of packages. The general form of a multileveled package statement is :

package pack1[.pack2[.pack3.....packn].....]];

7.3.3 Creating Your Own Packages

To create a package of classes, you have to follow three basic steps. They are explained below :

a) Select a Package Name

The first step is to give a name of the package. The name you choose for your package depends on how you are going to use those classes. Package name should be select in such a way that uniquely identifies your work.

One convention for naming packages that has been recommended by Sun is to use your domain name with the elements reversed. So, for example, packages of sun would be referred to using the name com.sun.java rather than just java. By convention, package names tend to begin with a lowercase letter to distinguish them

from class names. Thus, for example, in the full name of the built-in String class, java.lang.String, it's easier to separate the package name from the class name visually. This convention helps reduce name conflicts.

b) Create the Directory Structure

Step two in creating packages is to create a directory structure on your disk that matches the package name. If your package has just one name (mypackage), you'll only have to create a directory for that one name. If the package name has several parts, you have to create directories within directories. For the package name kkh.edu.csc.eng.assam, you'll need to create an kkh directory and then create a edu directory inside kkh, an csc directory inside edu, and an assam directory inside eng. Your classes and source files can then go inside the assam directory.

c) Use package to Add Your Class to a Package

The final step is putting your class inside packages is to add the package command to your source files. The package command says "this class goes inside this package," and is used like this:

```
package myclasses;
package kkh.edu.csc.eng.assam;
package java.awt;
```

The single package command, if any, must be the first line of code in your source file, after any comments or blank lines and before any import commands.

If your class does not have a package command in it, that class is contained in the default package and can be used by any other classes. But once you start using packages, you should make sure all your classes belong to some package to reduce the chance of confusion about where your classes belong.

7.4 UNDERSTANDING CLASSPATH

Once you installed Java on your machine it is required to set the PATH

environment variable to run the executables (java.exe, javac.exe and so on) from any location without having to type the full path of the command. For example :

C:\javac TestClass.java

Otherwise you need to specify the full path every time you run it, as

C:\Java\jdk 1.7.0\bin\javac TestClass.java

How does Java run-time system know where to look for packages that we created? By default, the Java run-time system uses the current working directory as its starting point. Thus, if your package is in a subdirectory of the current directory, it will be found. Or you have to specify a directory path or paths by setting the CLASSPATH. CLASSPATH is an environment variable which is used by Java virtual machine to locate user defined classes. Java compiler and JVM is used Classpath to determine the location of required class file. Another option is there to use the -classpath option with java and javac to specify the path to your classes.

For example, consider the following package

Package MyPackage

For a program to find MyPackage, one of the above three methods is true. The program can be executed from a directory immediately above MyPackage, or the CLASSPATH must be set to include the path of MyPackage, or the -classpath option must specify the path to MyPackage during the running of programme using java.

MyPackage must not include in the class path. It must simply specify the path to MyPackage. For example, if the path to MyPackage is

C:\Programme Files\Java\MyPackage

Then the class path to MyPackage is

C:\ Programme Files\Java

7.4.1 Setting Java Classpath in Windows

When we right click on My Computer and than choose Properties, advanced system settings and then Advanced and then Environment variable, this will open Environment variable window in win-

dows. Now specify your environment variable CLASSPATH and put the value of your JAVA_HOME\lib and also include current directory by including dot sign.

Now we can check the value of Java classpath in windows by type "echo %CLASSPATH" in DOS command prompt and it will show the value of directory which is included in CLASSPATH.

7.5 IMPORTING PACKAGES

Classes within packages must be fully qualified with their package name or names, it could become tedious to type in the long dot-separated package path name for every class you want to use. For this reason, Java includes the **import** statement to bring certain classes, or entire packages, into visibility. Once imported, a class can be referred to directly, using only its name.

The import keyword is used to import built-in and user-defined packages into java source file. This will allow your class refer to a class that is in another package by using directly its name.

Syntax : import pack1 [.pack2 [.....[.packn]]].*;

For example:

1. import p1.*;
2. import p1.p2.*;
3. import p1.p2.p3.*;

When statement 1 is executing we can import or access all the classes and interfaces of package p1 only but not its sub packages p2 and p3 classes and interfaces.

When statement 2 is executing we can import all the classes and interfaces of package p2 but not p1 and p3 classes and interfaces.

When statement 3 is executing we can import all the classes and interfaces of package p3 but not p1 and p2 classes and interfaces.



CHECK YOUR PROGRESS

Q1. What is the difference between Path and Classpath

Q2. What is package?

Q3. Which of the following is incorrect statement about packages?

a) Interfaces specifies what class must do but not how it does.

b) Interfaces are specified public if they are to be accessed by any code in the program.

c) All variables in interface are implicitly final and static.

d) All variables are static and methods are public if interface is defined pubic.

7.6 INTERFACES

Java does not support multiple inheritance, i.e. java classes cannot have more than one super class. Many applications need the concept of multiple inheritance. Java provides an alternative method of using multiple inheritance known as interfaces to support this concept. Although Java class cannot be a sub class of more than one super class, it can implement more than one interface.

An interface is just as java class, but it only has static constants and abstract method. The interface keyword is used to declare an interface. Here is a simple example to declare an interface.

Program 7.1 : Program to show use of import statements

```
Import  java.lang.*;
//Any number of import statements
Public interface NameOfInterface{
      //Any number of final, static fields
      //Any number of abstract method declarations\
}
Example:
Interface Animal{
      Public void eat ();
      Public void travel ();
}
```

Difference between Class and Interface:

An interface is different from a class in several ways, including:

- An interface does not have any constructors.
- In an interface all the methods are abstract.
- The interface cannot contain concrete (with implementation) methods.
- The only fields that can have in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Understanding relationship between classes and interfaces:

In figure 7.1 we can see that a class can extends another class, an interface extends another interface but a class implements an interface.

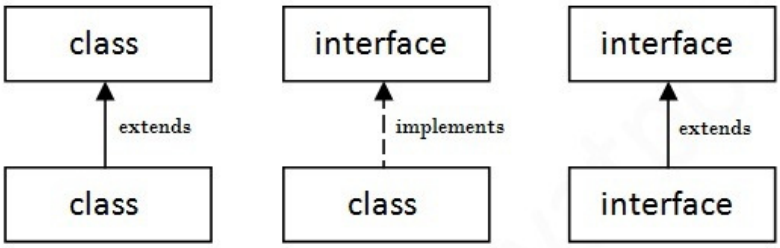


Figure.7.1: Relation between Class and Interface

Note: Initially we can create an interface as we would create a new class, but the name of the file ends in .java and has the same name as the interface itself (including the same capitalization).

7.7 IMPLEMENTING INTERFACES

Once an interface has been defined, one or more classes can implement that interface.

If a class implements more than one interface, the interfaces are separated with a comma. In case, a class is used to implement two interfaces that declare the same method, then the same method will be used by clients of either interface.

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Program 7.2 : Program using implements keyword.

```
public class MammalInt implements Animal{
      public void eat(){
            System.out.println("Mammal eats");
      }
public void travel(){
            System.out.println("Mammal travels");
      }
      Public int noOfLegs(){
```

```
            Return 0;
      }
      public static void main(String args[]){
            MammalInt m =new MammalInt ();
            m.eat();
            m.travel();
      }
}
This would produce the following result:
ammal eats
ammal travels
```

Example2:

```
interface MyInterface{
   public void method1();
   public void method2();
}
class XYZ implements MyInterface{
  public void method1()  {
       System.out.println("implementation of
method1");
  }
  public void method2()  {
       System.out.println("implementation of
method2");
  }
  public static void main(String arg[]){
      MyInterface obj = new XYZ();
      obj. method1();
  }
}
```

Output:

implementation of method1

A set of rules need to be followed during implementation of interfaces: