ଓଡ଼ିଶା ରାଜ୍ୟ ମୁକ୍ତ ବିଶ୍ୱବିଦ୍ୟାଳୟ, ସମ୍ବଲପୁର, ଓଡ଼ିଶା
Odisha State Open University, Sambalpur, Odisha
Established by an Act of Government of Odisha.

PG DIPLOMA IN COMPUTER APPLICATIONS

CSP-45

Programming in JAVA

Block 3

Advanced Concept in Java

EXPERT COMMITTEE

Dr. Manas Ranjan Senapati **(Chairman)**
Associate Prof., Dept. of IT
VSSUT, Burla

Dr. Santosh Kumar Majhi **(Member)**
Assistant Prof., Dept. of CSE
VSSUT, Burla

Mr. Atul Vikash Lakra **(Member)**
Assistant Prof., Dept. of CSE
VSSUT, Burla

Sri Aseem Kumar Patel **(Convener)**
Consultant (Academic)
School of Computer and Information Science
Odisha State Open University
Sambalpur, Odisha

PG DIPLOMA IN COMPUTER APPLICATIONS

Course Writers

Mr. Aseem Kumar Patel
Academic Consultant
Odisha State Open University, Odisha

Editor

Mr. Satya Sobhan Panigrahi
Subash Institute of Technology, Bhubaneswar

Material Production

Dr.Jayanta Kar Sharma
Registrar, Odisha State Open University, Sambalpur

 Printersby : Sri Mandir Publication, Sahid Nagar,Bhubaneswar

UNIT-1 Interface & Packages

Learning Objective

After learning this unit you should be able to

- Know about the concepts of an Interface, packages and its use.
- Learn how to create and initialize an Interface and package.
- Know the reason why java does not support Multiple Inheritance
- Know the build-in and user defined package in Java and its importance.
- Using other people's packages in your own classes
- Designing and working with interfaces
- What packages are and why they are useful for class design

Unit Structure

1.1 Introduction

Packages and interfaces are two capabilities that allow us greater control and flexibility in designing sets of interrelated classes. Packages allow you to combine groups of classes and control which of those classes are available to the outside world; interfaces provide a way of grouping abstract method definitions and sharing them among classes that may not necessarily acquire those methods through inheritance.

Today we'll learn how to design with, use, and create our own packages and interfaces.

1.2 Interfaces

The Java programming language does not permit multiple inheritance, but interfaces provide an alternative. An interface defines a protocol of communication between two objects.

Interfaces, like the abstract classes and methods, provide templates of behavior that other classes are expected to implement. However Interfaces, provide far more functionality to Java and to class and object design than do simple abstract classes and methods.

In the Java programming language, an interface is a reference type, similar to a class that can contain only: static constants and abstract method. Java uses Interface to implement multiple inheritance. A Java class can implement any number of Java Interfaces. All methods in an interface are implicitly public and abstract.

There are no method bodies. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces.

An interface declaration contains signatures, but no implementations, for a set of methods, and might also contain constant definitions.

A class that implements an interface must implement all the methods declared in the interface.

1.3 Defining Interfaces

In Java, the **interface** keyword is how you create an interface type.

An interface definition has two components: the interface declaration and the interface body. The interface declaration declares various attributes about the interface, such as its name and whether it extends other interfaces. The interface body contains the constant and the method declarations for that interface.

Syntax for interface declaration:

```
interface  Interface_Name
{
        Variable  declarations;
        Methods/functions  declarations;
}
```

The java compiler automatically appends public and abstract keywords before the interface method/ functions and public, static and final keywords before interface variables.

Example

```
interface  Osou
{
        int  p=10;
        void display();
}
```

Here the keyword interface tells that Osou is an interface containing one public, static and final variable p and one public and abstract method display() and shows like below implicitly

```
interface  Osou
{
        public  static  finalint  p=10;
        public  abstract void display();
}
```

1.4 Implementing interface

One of the rule of interface is that, it force another class to implement the methods specified in it, as long as that class declares an implements reference towards that interface.

To declare a class that implements an interface, you include an implements clause in the class declaration because interfaces provide nothing but abstract method definitions, you then have to implement those methods in your own classes, using the same method signatures from the interface. Once we include an interface, we have to implement all the methods in that interface.
After the class implements an interface, subclasses of your class will inherit those new methods just as a superclass had actually defined them. If your class inherits from a superclass that implements a given interface, you don't have to include the implements keyword in your own class definition.

Syntax:

class ClassName **implements** interfaceName

Example:

```
interface Animal{
 public void check();
}
class Tiger implements Animal{
  public void check(){
    System.out.println("class Tiger successfully Implemented Interface Method");
 }
  public static void main(String args[]){
    Tiger a = new Tiger();
    a.check();
 }
}
```

OUTPUT:

class Tiger successfully Implemented Interface Method
In the above program the check() method of the interface Animal has be defined in Tiger class.

Implementing interface by more than one class

Any no. of class can implements an interface

Example:

```
interface BankInfo{
        float interestRate();
        }
class HDFC implements BankInfo{
        public float interestRate(){return 7.15f;}
        }
class AXIS implements BankInfo{
        public float interestRate(){return 6.9f;}
        }
class Demo
{       public static void main(String[] args){
        BankInfo b = new HDFC();
        System.out.println("ROI for HDFC: "+b. interestRate());

        BankInfo b1 = new AXIS();
        System.out.println("ROI for AXIS: "+b1. interestRate());
} }
```

1.5 Accessing interface method and variable

The variables of an interface are "final" by default. Final variables are those variables, whose values cannot be changed. The methods of an interface are abstract means it is the responsibility of the class that has implemented to define.

Example: (try to change the value of interface inside class)

```
interface BankInfo{
        float h_int= 7.15f;
        float a_int=6.9f;
        void interestRate();
}

class HDFC implements BankInfo{
public void interestRate()
{
        H_int=6.5f;             // try to change the value from 7.15f to 6.5f
        System.out.println("ROI for HDFC: "+h_int);
        }
}

class AXIS implements BankInfo{
public void interestRate()
{
        System.out.println("ROI for AXIS: "+a_int);
        }
}

class Demo
{
public static void main(String[] args){
HDFC b = new HDFC();
b.interestRate();

AXIS b1 = new AXIS();
b1.interestRate();
} }
```

OUTPUT:

```
Demo.java:10: error: cannot assign a value to final variable h_int
        h_int=6.5f;
        ^
1 error
Tool completed with exit code 1
```

As the h_int is final by default we can't change the value of it.

1.6 Extending interfaces

Just like a class can extend another class to inherit all its super class feature, one interface can inherits from another interface by using the keyword extends. An interface can extend multiple interfaces.

When one interface inherits from another interface, the sub-interface acquires all the method definitions and constants of super interface. To extend an interface, we can use the extends keyword just as we do in a class definition:

Syntax:

```
public interface Interface1 extends Interface2 {
...
}
```

Example:

```
interface Interface1
{
        // body of Interface1
}
interface Interface2 extends Interface1
{
   // Body of Interface2
}
```

Here Interface1 is the super interface and Interface2 is the sub-interface.

Example:

```
interface Abc{
int x=10;
}

interface Xyz extends Abc
{       void show();
}

class Pqr implements Xyz
{
public void show()
{
System.out.println("Extended interface value is.."+x);
}
}
```

```
public class Demo{
public static void main(String args[])
{
        Xyz v2; //Reference variable of Xyz
v2 = new Pqr(); //assign object of class Pqr
        v2.show();
}
}
```

1.7 Packages

A package is a folder or directory that contains variety of classes and interfaces. If our programs are small and use a limited number of classes, we may not want to explore packages at all. But the more Java programming us do, the more classes we'll find we have. And although those classes may be individually well designed, reusable, encapsulated, and with specific interfaces to other classes, we may find the need for a bigger organizational entity that allows us to group our packages.

Packages are useful for several broad reasons:

- It allow us to organize our classes into units. Just as we have folders or directories on our hard disk to organize our files and applications, packages allow us to organize our classes into groups so that we only use what we need for each program.
- It reduce problems with conflicts in names. As the number of Java classes grows, so does the likelihood that you'll use the same class name as someone else, opening up the possibility of naming clashes and errors if you try to integrate groups of classes into a single program. Packages allow you to "hide" classes so that conflicts can be avoided.
- They allow us to protect classes, variables, and methods in larger ways than on a class-by-class basis.
- They can be used to identify our classes. For example, if we implemented a set of classes to perform some purpose, we could name a package of those classes with a unique identifier that identifies us.

1.8 Types of Packages

There are two types of packages in java

1. Build-in packages (API packages)
2. User-defined packages

1.8.1 Build-in packages / System packages

In-Build packages are those packages that are already in java API designed by the developers. These package contains classes, interfaces and methods to perform specific task in a program. This package is also known as java API package.

Here I have mentions some in built packages of java as follows:

java.lang: Contains language support classes (e.g classed which defines primitive data types, math operations). This package is automatically imported.

2) **java.io:** Contains only classes, interface and methods related to input / output operations .

3) **java.util:** util stand for utility package. Contains classes, interface and methods that implement data structures like Linked List, Dictionary and support; for Date / Time operations.

4) **java.applet:** Contains classes for creating and implementing Applets.

5) **java.awt:** awt stands for abstract window toolkit Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).

6) **java.net:** net stands for network, Contain classes for supporting networking operations.

7) **java. lang**: Lang stands for language. This package contains classes and interface related to primitives, string, mathematical functions, threads and exceptions.

8) **java.text:** This package is used for formatting date and time on day to day operation.

9) java.sql: used for database operation

To access the in-built package in user program we writes

import java.package_Name.*; Or

import java.package_Name.class_Name.*;

Example:

import java.lang.*

import java.lang.Math;

Here:

→ java is the base package.

→ lang is a sub package.

→ Math is a class which is present in the sub package lang.

Write a program in java using building package.

```
import java.util.Scanner;
OR
import java.util.*;

class Osou{
        public static void main(String args[]){
         Scanner sc=new Scanner(System.in);

          System.out.println("Enter your Salary");
          int sal=sc.nextInt();
```

```
        System.out.println("salary:"+sal);
        sc.close();
      }
}
```

In the above program, when we write **import java.util.Scanner** means it tells to compiler to add all the methods present in Scanner class in local program so that user can access it. Now we can use all the methods present in Scanner class in our program.

1.8.2 User defined packages

When user creates its own packages called user defined packages.

A user defined package is one which is developed by user or programmer to make java programming easy. Any class or interface is commonly used by many java programmers that class or interface must be placed in packages.

Syntax:
package package1;
Here, package is a keyword which is used for creating user defined packages, package1 represents the package name.

1.9 Adding and accessing class to a User defined packages

Creating your own packages is a difficult, complex process, involving many lines of code, long hours late at night with lots of coffee, and the ritual sacrifice of many goats. Just kidding. To create a package of classes, you have two basic steps to follow.

1. Pick a package name and create the directory structure
2. Use package to add your class to a package

Step-1

First create a directory i.e. **FirstPack** which should be same as the package name. Then ctreate a class i.e. **FirstClass** in side the FirstPack directory where the first statement must be package packageName i.e. FirstPack.

/* File name: FirstClass.java */

```
package FirstPack;

public class FirstClass{

    public int addTwoNum(int a, int b)

    { return (a+b);

    }

    }
```

Syntax for compiling the package

- **javac –d . FileName.java**

Here we write

- javac –d **.** FirstClass.java

Here, -d is an option that gives an instruction to JVM that go to FirstClass.java program take the package name (**FirstPack**) and that package name is created as directory automatically provides no errors are present in FirstClass.java. When FirstClass.java is not containing any errors we get FirstClass.class file and it will be copied automatically into current directory which is created recently i.e., FirstPack (package name). The above program cannot be executed since it doesn't contain any main method.

Step-2

To use classes and interfaces of the package in other Java program we have two approaches, they are **using import statement** and **using fully qualified name approach**.

Using import statement

import is a keyword used to import class or interface in a program

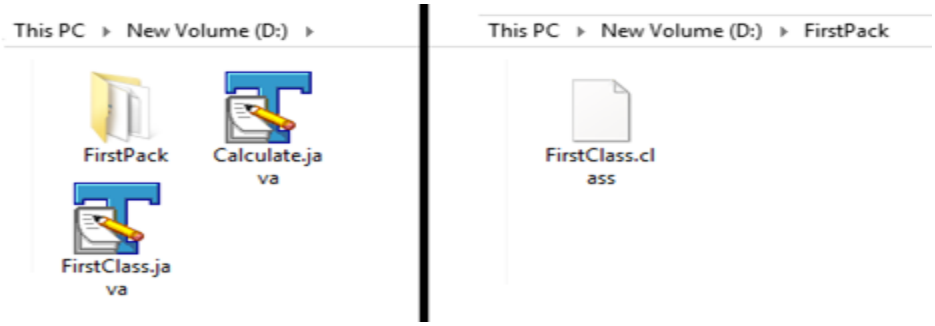
Syntax:
import packageName.*;

Now we use this class i.e. FirstClass in our program

/* File name: Calculate.java */

```
import FirstPack.FirstClass;

public class Calculate{
public static void main(String args[])
{ int p=34; int q=78;
        FirstClass obj = new FirstClass();
        int result=Obj. addTwoNum(p,q);
        System.out.println("Addition of 34 and 78 is…"+result);
        }

}
```

When we compile the above program we get the following error "package FirstPack does not
exist". To avoid the above error we must set the classpath as., SET CLASSPATH = **%CLASSPATH%;.;**



Using fully qualified name approach

Here instead of writing the import statement we specify the full path of the class or interface as follows

/* File name: Calculate.java */

```
import FirstPack.FirstClass;

public class Calculate{
public static void main(String args[])
{ int p=34; int q=78;
        FirstPack.FirstClass obj = new FirstPack.FirstClass();
        int result=Obj. addTwoNum(p,q);
        System.out.println("Addition of 34 and 78 is…"+result);
        } }
```

Finally when we compile and run above program we get

- **javac** Calculate.java // for compilation
- **java** Calculate // for execution

OUTPUT: Addition of 34 and 78 is…112

1.10 Let Us Sum Up

In this unit we have discussed about the interface and packages concept in java. We have seen how to create user defined packages and its implementation and details about build in packages. This unit provides a brief description on interface its use. We have also seen the scope of the variables and method present in interface.

We have also discussed the proper use of implements and extends keyword in interface along with the real time implementation of user defined packages and its use in project work.

1.11 Self-Assessment Questions

Q.1. What is interface?

A. ……...

……

……

……

…….

……..

Q.2. What if we define a method in an interface?

A. ……...

……

……

……

…….

……..

Q.3. When we need to use extends and implements?

A ……...

……

……

Q.4. What will happen if we are not implementing all the methods of an interface in class which implements an interface?
A.……...

……

……

……

……

……

……

Q.5. What is user defined package. How to access an user-defined package from a class explain with an example.

A. ……...

……

……

……

……

……..

1.12 Model Questions

Q.1. Why interface in java?

Q.2. Difference between interface and abstract class

Q.3. WAP in java for addition, subtraction, multiplication and division using package.

Q.4. What is user-defined package? Explain.

Q.5. How interface is the replacement of multiple inheritance in java? Explain.

1.12 References

https://gerardnico.com/lang/java/interface

http://www.dmc.fmph.uniba.sk/public_html/doc/Java/ch16.htm

https://www.geeksforgeeks.org

UNIT-2 Exception Handling

Learning Objective

By the end of this chapter we will learn about:

- How to use exception handling for robustness programing.
- How JVM handles error situations.
- Use of the try-catch statement.
- Learn about programmer Exception message.
- Difference between checked and unchecked exceptions.

Unit Structure

2.1 Introduction

An error is common part of a program. Most of the time we find error in a program when we compile the program or execute the program. An error is an unusual condition or problem arises during programming execution. The reason may be the mistake of programmer, hardware problems, file missing, resources exhaustion etc.

An exception is a runtime error event that occurs in the normal process flow of a program.

Errors are of two types. They are compile time errors and run time errors.

Compile time errors/ Syntax error

These error occurs when compiler unable to compile the piece of code or program. When we compile a source code, first compile checks whether a proper environment in available for the JVM or not to execute the byte code which it has generated for the source code. Thus syntax error results in compilation error.

Basically it occurs due to poor understanding of programming language.

Run time errors

Run time errors are those which are occurring in a program when the user inputs invalid data. Whenever a runtime error occurs, program is terminated and controls comes out of the program unconditionally.

A program need to handle error that occur at runtime. Runtime error is of two type i.e. programming error and runtime environment error.

Trying to dividing by zero, calling a method with invalid parameter, invalid index to access an array element are the example of Programming error. Whereas opening an invalid file, network connection going down, read write error in file are environment error.

2.2 Exception handling

Basically, an exception is a runtime error. But compile time does not comes under this category. All the exception are occurs at runtime but some are detected at compile time whereas some are at runtime.

The exception that are checked at compile time by compiler are called **checked Exception** and those are checked at runtime by JVM are called **Un-checked Exception.**

Let's try to understand about exception handling with an example

Example:

```
class Osou
{
        Public static void main(String args[]){
                Int a=100;
                Int b=0;
                Int c=a/b;
                System.out.println("Ans is…"+c);
}
}
```

OUTPUT:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Osou.main(Osou.java:7)
Press any key to continue . . .
```

Explanation:

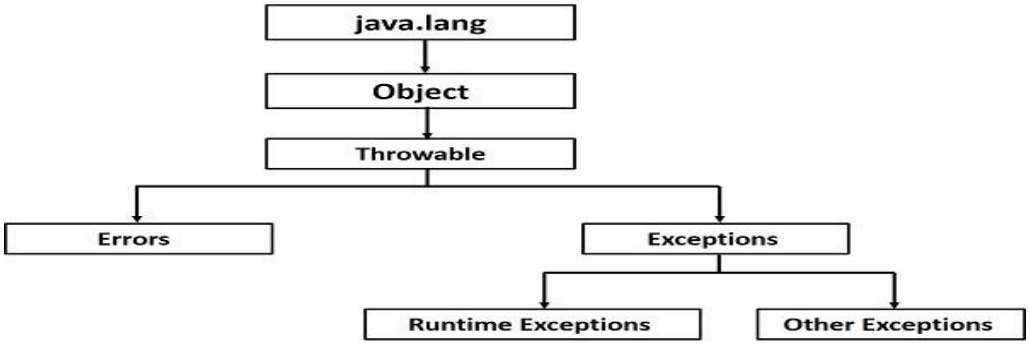
The above program compiled successfully but when executing it, value of c becomes infinity which has no existence in programming language. Thus it terminates the program. This is a logical error. For each and every logical error that is a corresponding exception class present in java.lang package. When logical error occurs JVM creates an object of corresponding exception class of java.lang package as these object is not handled explicitly it return back to JVM. JVM accept this any terminate the program.

If we are able to handle those exception objects then we can prevent the program from termination.

Thus the mechanism of handling these exception class object and avoding them from reaching back to JVM is called **Exception Handling.**

2.3 Exception class Hierarchy

The class hierarchy of error and exception classes in java is as follows



Object class is the superclass of all java classes and Throwable class is the super class of all exception and error class.

2.4 Using Try catch, finally, throws and throw

Java handles exception using the following keywords

- ✓ try
- ✓ catch
- ✓ throw
- ✓ throws
- ✓ finally

2.4.1 Using try and catch block

In try block we keep the block of statements which are to be monitored by JVM at run time i.e., try block must contain those statements which causes problems at run time. If any exception occurs the control will be skipped automatically to appropriate catch block. The try block recognize the exception class objct created by JVM and transfer them to the catch block.

Catch block is used to provide user friendly message by catching system generated message. In catch block it assign the object to a particular reference so that program will not terminate unexpectedly.

Syntax:

```
try{
}
catch(Exception e)
{
}
```

Example:

```
class Osou
{
        public static void main(String args[]){
                int a=100;
                int b=0;
        try{
                int c=a/b;
                System.out.println("Ans is…"+c);
                }
        catch(Exception e){
System.out.println("value of b canot be zero :"+e);
}
                }
}
```

Explanation:

In the above program when an exception occurs in line no 7 (int c=a/b) JVM create an object of corresponding exception class and send it to catch block where the reference of Exception class (Exception e) catch it and show a user friendly message instead of an system generated message.

If any exception is occurs in try block, execution will be stopped and the rest of the statements in try block will not be executed at all and the control will go to catch block. For every try block there must be at least one catch block. In this way we can handle any kind of exception using try and catch block.

2.4.2 Using Multiple catch block

A try block may have any number of catch blocks. If an exception occurs in the try block, the control passed to the first catch block in the list. If the exception type of exception, matches with the first catch block it gets caught, if not the exception is passed down to the next catch block. This continue until the exception is caught or falls through all catches.

Example:

```
class Mexception
{
 public static void main(String[] args)
 {
  try
  {
   int a []={10,20,30,40};
   int c=50/0;
   a[4]=50;
  }
  catch(ArithmeticException e)
  {
   System.out.println("divide by zero Exception Occurs");
  }
  catch(ArrayIndexOutOfBoundsException e)
  {
   System.out.println("Array index out of bound exception occurs");
  }
 }
}
```

OUTPUT: divide by zero Exception Occurs

Explanation: Though in the above program both ArithmeticException and ArrayIndexOutOfBoundsException occurs in try block but first, program encounter with ArithmeticException in catch block and it will be handled there.

If we remove the line no. 8 (int c=50/0 ;) then the output will be **Array index out of bound exception occurs.**
The only disadvantage of this is that, it is not prior known to programmer about the type of exception so it will be very difficult to mention the exception type.

2.4.3 Using throw Keyword

throw keyword is used to throw an exception explicitly. Only object of Throwable class or its sub classes can be thrown. Program execution stops on encountering throw statement, and the closest catch statement is checked for matching type of exception.

There are two purpose to use throw keyword.

1. Define our own set of conditions and throw an exception explicitly
2. To create user defined exception class

Syntax:

throw new exception_class("message");

Points to remember:

- We are not allowed to write any statement after the throw statement otherwise it will show compile time error
- We can use throw keyword only for throwable type otherwise we will get compile time error "incompatible types".

Define our own set of conditions and throw an exception explicitly

Normally, when ever runtime error occurs java create an object of corresponding Exception class and through it to JVM and terminate program like when we divide a number by zero java create an object of ArithmeticException class and terminate the program. We can throw ArithmeticException when we divide number by 5, or any other numbers, yes, this is possible. What we need to do is just set the condition and throw any exception using throw keyword.

Example:

```
public class Osou{
  void eligibilityCheck(int age){
        int r;
        if((age>=60))
          throw new ArithmeticException("its Your retirement age!");
        else
          r = 60-age;
```

```
            System.out.println("you have "+r+ " years left to retire");
  }
  public static void main(String args[]){
        Osou obj = new Osou ();
        obj. eligibilityCheck(65);
        System.out.println("End Of Program");
  }
}
```

OUTPUT:

```
Exception in thread "main" java.lang.ArithmeticException: it?s Your retirement a
ge!
        at Osou.eligibilityCheck(Osou.java:5)
        at Osou.main(Osou.java:12)
Press any key to continue . . .
```

Explanation: In the above example we have added our own condition for ArithmeticException i.e.(if age>=60) java through an exception and terminate the program.
Here age is 65, So it meet the condition and create an object of ArithmeticException class and through it with the message "it's your retirement age!" to JVM and terminate the program.
As we have not written any code to catch the thrown object it will show the error message.
But if we want to handle this kind of exception with our own message then we need to introduce try and catch block in the above program.

Here is the example

```
public class Osou{
  void eligibilityCheck(int age){
        int r;

        if((age>=60))
          throw new ArithmeticException("its Your retirement age!");
        else
          r = 60-age;
          System.out.println("you have "+r+ " years left to retire");
  }
  public static void main(String args[]){
        Osou obj = new Osou ();
        try{
        obj. eligibilityCheck(65);}
        catch(ArithmeticException e){
        System.out.println(e);
        }
  }
}
```

OUTPUT:

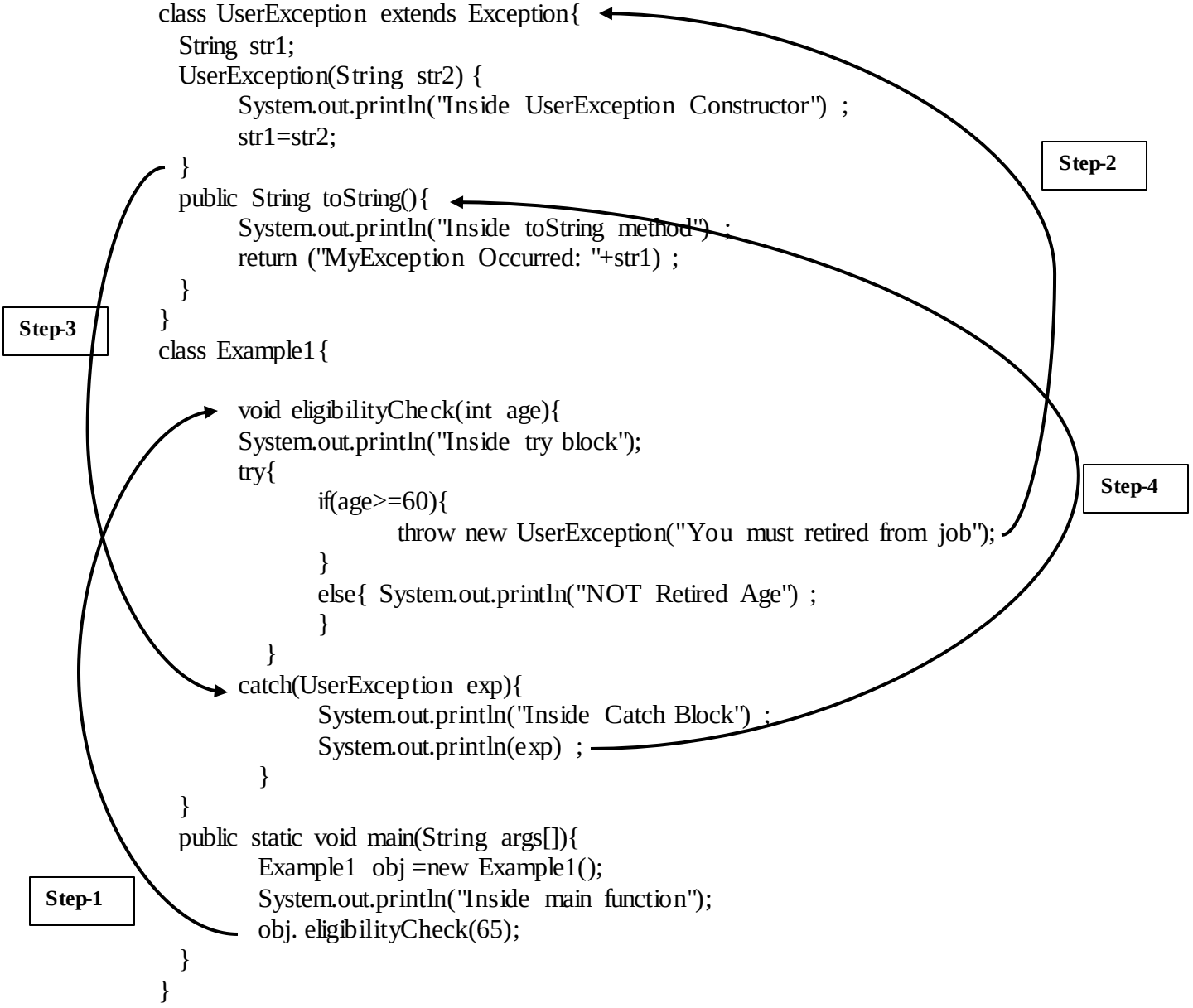
```
java.lang.ArithmeticException: its Your retirement age!
Press any key to continue . . .
```

User defined exception class

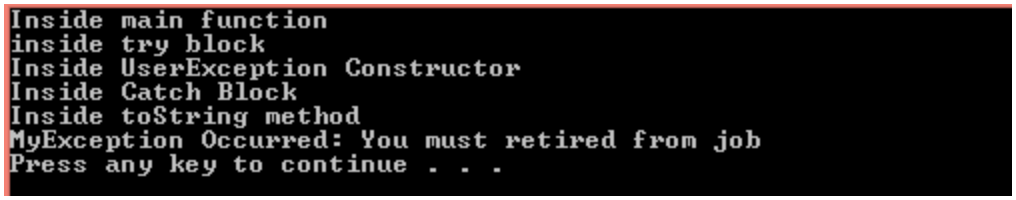
Most of the time there might be situation where user wants to trigger an exception created by programmer that helps us to figure out what exactly went wrong in the program that gives more information and details about what error we have committed in program or user input went wrong in the program. So the process of creating these exceptions that a programmer creates of its own to help them understand where the programmer went wrong or what error occurred is called **User defined exception.**

Example:

```
class UserException extends Exception{
  String str1;
  UserException(String str2) {
        System.out.println("Inside UserException Constructor") ;
        str1=str2;
  }
  public String toString(){
        System.out.println("Inside toString method") ;
        return ("MyException Occurred: "+str1) ;
  }
}
class Example1{

        void eligibilityCheck(int age){
        System.out.println("Inside try block");
        try{
                if(age>=60){
                        throw new UserException("You must retired from job");
                }
                else{ System.out.println("NOT Retired Age") ;
                }
          }
        catch(UserException exp){
                System.out.println("Inside Catch Block") ;
                System.out.println(exp) ;
          }
  }
  public static void main(String args[]){
          Example1 obj =new Example1();
          System.out.println("Inside main function");
          obj. eligibilityCheck(65);
  }
}
```



OUTPUT:



2.4.4 Using throws Keyword

There are two types of exception in java i.e. checked exception and unchecked exception. Programmer have to handle the checked exception, if we don't handle them then the program will show error at compile time.

So we have an option to handling checked exceptions using Throws keyword.

By using throws we can declare multiple exceptions at a time.

Throws keyword is used to declare the exception, it provides information to programmer that there may occur an exception so during call of that method, programmer must use exception handling mechanism.

Let's try to understand the concept of throws keyword with an example
Suppose we have a program with three method named m1(), m2() and m3(). Method m2() calling the m1() method and m4() method calling the m2() method. If there will be any error in method m1() then method m2() will abnormally terminated which is not a good sign for programmer. In this situation java come up with the concept of throws keyword to overcome from this problem.

It says when m1() method declares, it also mention that method m1() may through an exception by writing

m1() throws Exception

Now, it is the responsibility of method m2() to handle this situation by keeping the method call statement inside try…. Catch block.

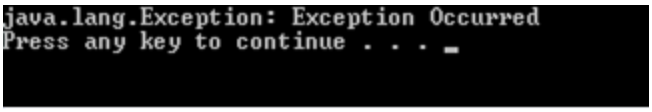
Example:

```
int method1() throws Exception
{
        // statement1
}
void method2()
{
        try{
         method1()
        }
catch(Exception e)
{
// statement }
```

Example:

```
class ThrowsExample {
  void Method1(int n)throws Exception{
    if(n==1)
      throw new Exception("Exception Occurred");
    else
      System.out.println("Every thing is OK");
  }
}

public class Example1{
  public static void main(String args[]){
   try{
    ThrowsExample obj=new ThrowsExample();
    obj. Method(1);
   }catch(Exception ex){
    System.out.println(ex);
   }
  }
}
```

OUTPUT:



In this example the method Method1() is throwing an checked exceptions so we have declared these exceptions in the method signature using throws Keyword. If we do not declare these exceptions then the program will throw a compilation error.

2.4.5 finally Keyword

A finally block contains all the statements that has to be executed whether exception occurs or not such as closing a connection, closing a file etc.
As we know if there is any exception inside the try block then it is being handle by catch block. There is another block in java called finally block, now the code inside finally block always executes whether or not exception has occurred. Basically we keep that type of code inside the finally block that is used for cleanup purpose.

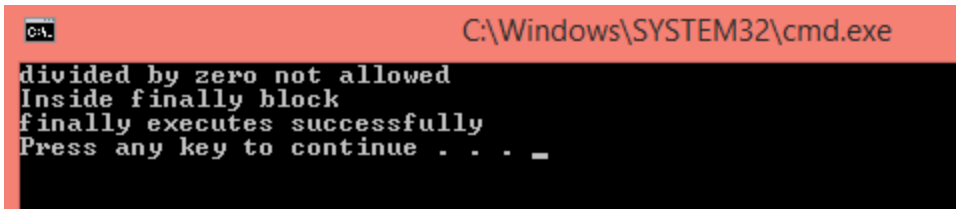
Syntax:

```
try {
   //Exception Statements
}
catch {
  // Exception Handled
}
finally {
  //Statements that has to be executed }
```

Example:

```
class FinallyEx
{
  public static void main(String args[]) {
    try{
          int n=568/0;
          System.out.println(n);
    }
    catch(ArithmeticException ex){
      System.out.println("divided by zero not allowed");
    }
    finally{
          System.out.println("Inside finally block");
    }
    System.out.println("finally executes successfully");
  }
}
```

OUTPUT:



Explanation:

When we run the above program an exception will occurs in side try block i.e. ArithmeticException and control will move to catch block and handled it then as we mention above control will go to finally block and statements will get executed.

Important points for finally block

1. We cannot use finally without a try block.
2. Finally block is optional.
3. In normal case the sequence of execution is try block and finally block but if an exception occurs then the catch block is executed before finally block.
4. An exception in the finally block, behaves exactly like any other exception.
5. The code inside finally block execute even if the try block contains return, break or continue statement.

2.4.6 Difference between throw and throws

Throw	throws
The throw keyword is used to throw/transfer the exception object to JVM manually.	The throws keyword is used to give the charge of exception handling to the method who is calling it.

throw Throwable-instance;	return_type method_name(val-list)
	throws ExceptionClass_list
	{
	// body of method
	}

The throw keyword is followed by the object of exception class.	The throws keyword is followed by name of the exception classs.
The throw keyword can throw a single exception at a time.	The throws keyword can handle more than one exception by declaring them separated by a comma.
throw keyword also be used to break a switch statement without using break keyword.	This is not possible here

2.4.7 Difference final finally and finalize

final: final is a keyword, final can be used with variable, method or class. final variable value can't be changed, final method can't override, In case of a class, it means that the class can't be inherited.

finally: finally block is used in exception handling. Regardless of whether an exception is thrown or not, the finally-block will always be executed. Even if we get some exception during the execution of the catch-block then also finally-block will get executed. Normally system resources release statement are written inside finally.

finalize(): finalize() method is used in garbage collection. finalize() method is called just before the object is garbage collected. The finalize() method can be used to perform any cleanup processing.

2. 5 Lets Us Sum Up

In this unit we learned the basic concepts of exception handling in Java, and got the idea of how to handle exceptions in different ways. We have seen the types of Exception i.e. checked and unchecked Exception. This unit provides a brief description on hierarchy of exception class in java. We have seen how to create user defined exception class and its implementation and details about build in exception class.
We have also discussed the proper use of try, catch, throw and throws keyword in exception handling.

2. 6 Self-Assessment Questions

Q.1. What is Exception in java and how it is different from error?
A. ……

……

……

……

……

Q.2. Difference between Checked and unchecked Exception?
A. ……

……

……

……

……

……

……

Q.3. Difference between throw and throws keyword.
A. ……

……

……

……

……

……

Q.4. What is finally keyword in Java? Explain.
A. ……

……

……

……

……

……

……

Q.5.What is user defined Exception? How to create an user defined exception?
A. ……

……

……

……

……

……

……

2. 7 Model Questions

Q.1 What are the Exception Handling Keywords in Java?

Q.2. What are important methods of Java Exception Class?

Q.3. What happens when exception is thrown by main method?

Q.4. How to write custom exception in Java?

Q.5. What are different scenarios causing "Exception in thread main"?

2. 8 References & Further References

1. www.fresh2refresh.com
2. http://sofia.fhda.edu/gallery/java/
3. http://greenteapress.com/thinkpython/thinkpython.html
4. http://www.dmc.fmph.uniba.sk/public_html/doc/Java/ch16.htm
5. https://beginnersbook.com/2013/04/user-defined-exception-in-java/
6. https://www.geeksforgeeks.org/
7. http://geekexplains.blogspot.com/2008/05/final-finally-finalize.html

UNIT-3 Applets

Learning Objective

By the end of this chapter we will learn about:

- It helps to learn how to create applets
- To learn web based programming
- How HTML works
- How to execute an applet program
- How to embed applet with Html
- Difference between Application program and applet program.

Unit Structure

3.1 Introduction

In JAVA we can write two types of programs. They are Application program and applet program.

- An Application program is one which runs in the local disk and the output is not sharable. Every Application program contains main method along with System.out.println statements.
- An applet program is one which runs in the browser or World Wide Web and it can be accessed from anywhere. It does not contain main methods and System.out.println statements.

Definition of applet
An applet is a java program which can be executed on a java enabled browser.

Applets are the dominant tools to creating the programs in java. A Java applet is a small dynamic Java program that can be transferred via the Internet and run by a Java-compatible Web browser. Applet can be embedded into HTML pages. Java applets run on the java enables web browsers such as Mozilla and Internet Explorer. Applet is designed to run remotely on the client browser, so there are some restrictions on it. Applet can't access system resources on the local computer. Applets are used to make the web site more dynamic and entertaining.

To write an applets program, we must import a package called java.applet.*. This package contains only one class called Applet whose fully qualified name is **java.applet.Applet.**

3.2 Local and remote applets

Local Applet: - An applet program written in local system and stored in a local disk is known as a local applet. When a Web page is trying to discover a local applet, it does not required any Internet connectivity. It simply searches the directories in the local system and locates and loads the specified applet.

Identifying a Local Applet:

```
<applet codebase="path"
        code="NewApplet.class"
        width=120 height=120 >
</apple>
```

We will learn about local applet more in details in rest of the chapter.

Remote Applets: - A remote applet is written by someone else and stored on a distributed computer system connected to the Internet. If our system is connected to the internet, we can download the remote applet onto our system via Internet and run it.

To find and load a remote applet, we must have the URL and must be specified in the applet's HTML document as the value of the CODEBASE attribute.

Identifying a Romote Applet:

```
<applet
codebase="http://www.osou.ac.in/applets/"
code="NewApplet.class"
width=120
height=120 >
</applet>
```

3.3 Applet vs. Application Program

All Java programs can be of two type i.e. Applications and Applets. The main difference between these two are that applications program contain main() method whereas applets do not. One more is, applications can be executed at command prompt and applets in a web enabled browser or in an appletviewer.

Applet Program	Application Program
Normally Program size is small	Normally Program size is large
Used to run a program on web enabled Browser like chrome, Mozilla, IE	Can be executed on standalone computer system/local computer system
Applet is portable and can be executed by any JAVA supported browser.	Need JDK, JRE, JVM installed on client machine to run
Applet applications are executed in a Restricted Environment	Application can access all the resources of the computer
Applets are created by extending the java.applet.Applet	Applications are created by writing public static void main(String[] s) method.
Applet application has 5 methods which will be automatically invoked on event of specific task. **init(), start(), stop(), destroy(), paint()**	Application has a single start point which is main method **public static void main(String args[])**

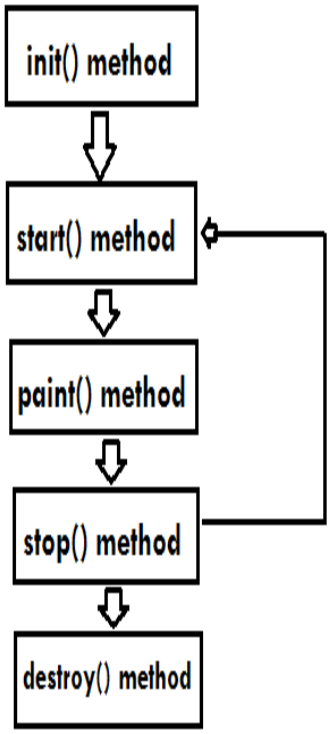
Example:
import java.applet.*;
public class AppletEx extends Applet
{
public void init() { }
public void start() { }
public void stop() {}
public void destroy() {}
public void paint(Graphics g) {}
}
 	
	Example:
	public class ApplicationEx
	{
	public static void main(String args[])
	{
	//statement
	}
	}

3.4 Life cycle of an Applet

Life cycle is the procedure followed by the JVM to execute a program. Different stages, an applet undergoes between its object creation to destruction is known as Applet Life Cycle. Each stage is represented by a method. In the life span of an applet it undergoes between five stages which is represented by 5 method.
The methods are

1. init() method
2. start() method
3. paint() method
4. stop() method
5. destroy() method

These methods are Applet lifecycle method which are defined in java.applet.Applet class except paint() method. paint() method is defined in java.awt.Component class



init() method: The life cycle start in the init() method. As soon as browser encounters the <applet> tag in the html file, the browser handover the .class file to its JVM. JVM create an object of that class and this object would be assigned to a reference of the Applet class. Then JVM calls init() method. This is the first method to be called where variable initialization is done. It is called once during its life time.

start(): this method called just after the init() method on the reference of the Applet class. This method is called to restart an applet after it has been stopped. Normally business logic are written in this method.

stop(): This method is get called when we minimize the window. If the control has to come out of the applet program, then JVM calls stop(). This method stores the code that will temporarily releases the resources which are obtained in init() method. this method is always called before destroy() method.

destroy(): if the browser is closed, then before deleting the object of the .class file, JVM called the destroy() method on the reference of the Applet class.

paint(): This method is called by the browser after completion of start method to display the data on the browser. It internally called drawString() of Graphics class.

Example:
/* file name: FirstApplet.java */

```
import java.awt.*;
import java.applet.*;
public class AppletTest extends Applet
{
 public void init()
 {
        System.out.println("inside init() method");
 }
 public void start()
 {
        System.out.println("inside start() method");
 }
 public void stop()
 {
          System.out.println("inside stop() method");
}
public void destroy()
 {
        System.out.println("inside destroy() method");
 }
 public void paint (Graphics g)
 {
        g.drawString("HELLO WORLD",200,200);
 }
}
```

Save the above program.

Compile: javac AppletTest.java
Run the applet: To run the applet we have two ways. They are using HTML program and using applet viewer tool.

Using HTML program
To run the applet program, we need to write the HTML program
Syntax:
<applet code =".class file of the applet" height = height value width = width value>
</applet>
Example:
/* file name: MyHtml.html */

```
<HTML>
<HEAD></HEAD>
        <BODY>
        <APPLET CODE="AppletTest.class" CODEBASE="."
         WIDTH=400 HEIGHT=300>
        </APPLET>
        </BODY> </HTML>
```

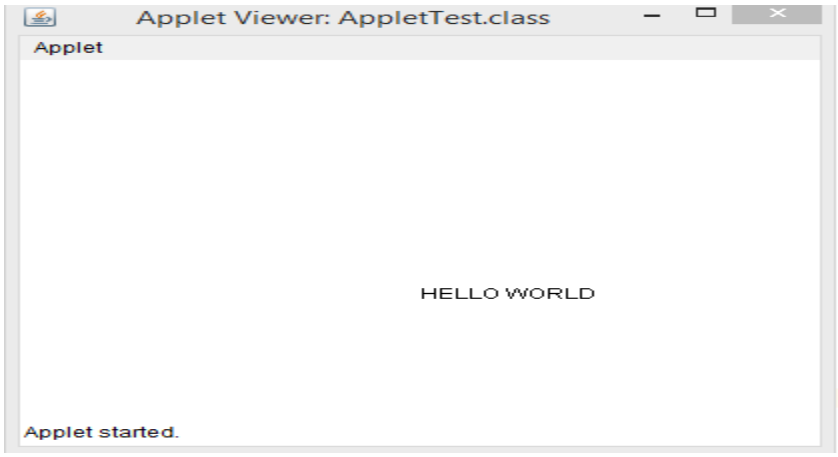
To run our Applet in browser window. type the complete url path for your HTML file, for instance c:\java\osou\MyHtml.html.

Using applet viewer tool

To run applet program using applet viewer we need to create the above html file (My.Html.html) and open cmd and go to source folder where the .class file and html file is present

Write the following code and press enter to get output

- **appletviewer MyHtml.html**




```
C:\Users\osou-18\Documents\Applet>appletviewer run.html
inside init() method
inside start() method
inside stop() method
inside start() method
inside stop() method
inside destroy() method

C:\Users\osou-18\Documents\Applet>
```

When the program runs JVM automatically call the init() and start() method, when we minimize the applet viewer it calls stop() method and when we resume the applet viewer it again call the start() method and when we close the applet viewer windows finally it call the stop() an ddestroy() method.

3.5 Creating an executable applet

Executable applet is nothing but the .class file of applet program, which can be created by compiling the source code of the applet program.

Syntax:

- **javac AppletProgram.java**

The output of the above command is a .class file i.e. AppletProgram.class which must be placed in the same directory where the source file is present.

3.6 Creating applet tag

In html, applet with in an angle bracket i.e. <applet> is a tag. The <applet...> tag send the name of the applet and tells to browser about the height and width required for the applet.
Every tag have certain attributes that need to specify.
The minimum required attributes for applet tag <applet> is code, height and width.

```
<applet code="filename.class"
        Width="100"
        Height= "100" >
</applet>
```

Height and width attributes specifies the display area for the applet as 100 pixels width and 100 pixels height.

3.7 Adding applet tag to html

To run an applet program in a browser, we have to create a HTML file with extention .html that contains the APPLET tag along with the appropreprate attributes and its value. This <applet> tag need to be written inside the <body> tag of the html file as follows.
/* filename: MyHtml.html */

<html> <head> <title></title> </head> <body> **//add the applet tag here** </body> </html>	<html> <head> <title></title> </head> <body> **<applet code="filename.class"** **width="100" height= "100" >** **</applet>** </body> </html>
Default html code	**After adding applet tag in html file**

3.8 Running the applet

To run an applet with an applet viewer, we need to execute the HTML file in command prompt.
Go to the source location where the html file and .class file is present and run the command
c:\document/applet/Example> **appletviewer Myhtml.html**

3.9 Passing parameters to applet

Parameters means the information that can be passed to an applet from the HTML page. Parameters are specified using the HTML's param tag with name and value attributes.
Param tag is a sub tag of applet tag having name and value as an attributes. Name attribute is used to specify the name of the parameter and value attribute is used to specify the value that will be passed as parameter.

Syntax:

<param name="var_name" value="val1" />
Example:
<param name="age" value="25" />
<param name="uni" value="OSOU" />

getParameter() Method
The getParameter() is a method present in Applet class of applet package. it can be used to retrieve the parameters passed from the HTML page.
Syntax:
String getParameter(String param-name);

Example:
A simple example of param tag and getParameter() method in applet program
/* file name: FirstApplet.java */

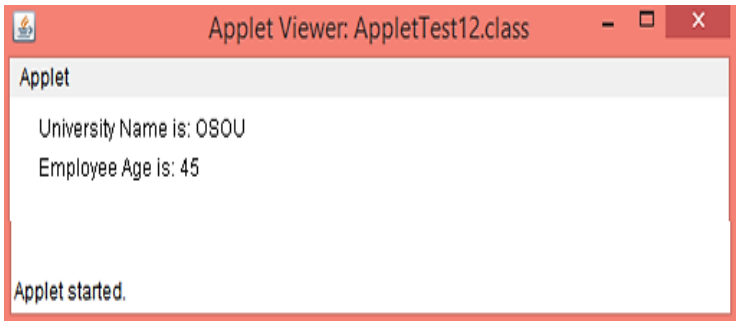
```
import java.awt.*;
import java.applet.*;
public class AppletTest extends Applet
{
String uname;
String age;
 public void init()
 {
        uname = getParameter("msg");
        age = getParameter("emp_age");
 }

 public void paint (Graphics g)
 {
        g.drawString("University Name is: " + uname, 20, 20);
        g.drawString("Employee Age is: " + age, 20, 40); }
}
```

Compile: javac AppletTest.java

```
/* filename: MyHtml.html */
<html>
   <body>
<applet code="MyApplet" height="300" width="500">
                <param name="msg" value="OSOU" />
                <param name="emp_age" value="45" />
        </applet>
    </body>
</html>
```

Execute the program: appletviewer MyHtml.html

OUTPUT:



3.10 Getting input from user in applet

Applets work in a GUI environment. Therefore, we have to first create text filed using TextField class of the applet package. Once text fields is created for receiving input, we can take input value in the text field do operation as per our requirement.

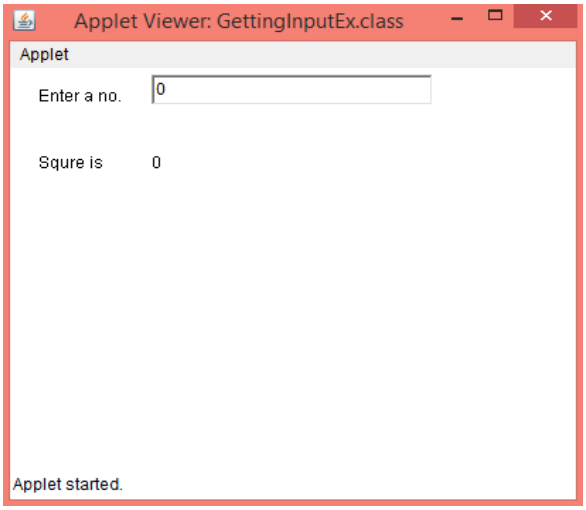
Normally text fields contain items in string form. They need to be converted to the write form before they are used in any applications. The results are then converted back to string for display.

Example:

```
import java.awt.*;
import java.applet.*;
public class GettingInputEx extends Applet
{
                TextField tf1;
                public void init()
    {
        tf1 = new TextField(25);
        add(tf1);
        tf1.setText("0");
    }
```

```
    public void paint(Graphics g)
    {
        int p=0,q=0;
        String str,str1;

        g.drawString("Enter a no.",20,25);
        str1=tf1.getText();
        p=Integer.parseInt(str1);
        q=p*p;
        str=String.valueOf(q);

        g.drawString("Square is",20,75);
        g.drawString(str,100,75);
    }
}
```

OUTPUT:



After compiling this applet program will appear but not calculate the square of the variables p because for this we need event programming.

3.11 Let Us Sum Up

Applets are probably the most common use of the Java language today. Applets are more complicated than many Java applications because they are executed and drawn inline within Web pages. In this Unit we learned the basics of creating applets, including the following things:

- All applets we develop by extending Applet class, of the java.applet package. The Applet class provides basic behavior for how the applet will be integrated with and react to the browser and various forms of input from that browser and the person running it.
- Applets have five main methods, which are used for the basic activities an applet performs during its life cycle: init(), start(), stop(), destroy(), and paint().
- To run a compiled applet class file, we include it in an HTML Web page by using the <APPLET> tag. When a Java-capable browser comes across <APPLET>, it loads and runs the applet described in that tag.
- Unlike applications, applets do not have a command line on which to pass arguments, so those arguments must be passed into the applet through the HTML file that contains it. You indicate parameters in an HTML file by using the <PARAM> tag inside the opening and closing <APPLET> tags. <PARAM> has two attributes: NAME for the name of the parameter, and VALUE for its value.

3.12 Self-Assessment Questions

Q.1. What is applet? How applet is different from Application Program?

A. …….....

……

……

……

……

……

Q.2. Explain the life cycle of applet with example.

A. …….....

……

……
……
……
……

Q.3. what is applet tag? How to execute an applet program using applet tag?

A. ……**.....**

……

……

……

……

……

Q.4. How to take an input from html page in an applet program?

A. ……**.....**

……

……

……

……

……

Q.5. Write an applet program to take input using param tag and find the sqrt of the number.

A. ……**.....**

……

……

……

……

……

3.13 Model Questions

Q.1. What is the difference between an Applet and a Java Application?

Q.2. Difference between local and remote applet.

Q.3. What is the order of method invocation in an Applet?

Q.4. Can we pass parameters to an applet from HTML page to an applet? How?

Q.5. What are the steps involved in Applet development?

3.14 References

1. https://www.roseindia.net/java/example/java/applet/
2. http://www.dmc.fmph.uniba.sk/public_html/doc/Java/ch8.htm#Summary
3. https://edurev.in/studytube/Core-Java-Important-Long-Questions-for-Viva-and-In/3617b2d3-64e0-4ed2-a046-a1cbd663a72b_t#
4. https://docs.oracle.com/javase/tutorial/
5. https://www.geeksforgeeks.org/java/

Unit - 4 Multithreading in Java

Learning objectives

After the Completion of this unit you should be able to know

- The concept of Threading
- Thread life cycle
- Create and use threads in program
- Describe how to set the thread priorities
- Thread synchronization
- Inter-thread communication in programs

4.1 Introduction

LIKE PEOPLE, COMPUTERS can multitask. That is, they can be working on several different tasks at the same time. A computer that has just a single central processing unit can't literally do two things at the same time. It is increasingly common for computers to have more than one processing unit, and such computers can literally work on several tasks simultaneously. To use the full power of these multiprocessing computers, a programmer must do parallel programming, which means writing a program as a set of several tasks that can be executed simultaneously.

In Java, a single task is called a thread. The term "thread" means a sequence of instructions that are executed one after another. A thread is single sequence of execution that can run independently in an application.

An executing Java program is called a Process. That process executes your instructions (program) sequentially following the path you created when you wrote your program. This single path is called a thread. As such, your program is only doing one activity (Java statement) at a time and working on one task (your programs list of statements) at a time. For most situations, this works fine. But there are times when you would like to have your program doing more than one thing at time. Using Java Threads it is possible to create additional threads or paths of execution that run in parallel with the main (or first) thread.

Every Java program has at least one thread; when the Java virtual machine runs your program, it creates a thread that is responsible for executing the main routine of the program. This main thread can in turn create other threads that can continue even after the main thread has terminated.

This unit covers the very important concept of multithreading in programming. Uses of thread in programs are good in terms of resource utilization of the system on which application(s) is running. Multithreaded programming is very useful in network and Internet applications development. In this unit you will learn what is multithreading, how thread works, how to write programs in Java using multithreading. Also, in this unit will be explained about thread-properties, synchronization, and inter-thread communication.

4.2 Multithreading

Our PC has only a single CPU; you might ask how it can execute more than one thread at the same time? In single processor systems, only a single thread of execution occurs at a given instant. But multiple threads in a program increase the

utilization of CPU. The CPU quickly switches back and forth between several threads to create an illusion that the threads are executing at the same time.

You know that single-processor systems support logical concurrency only. Physical concurrency is not supported by it. Logical concurrency is the characteristic exhibited when multiple threads execute with separate, independent flow of control. On the other hand on a multiprocessor system, several threads can execute at the same time, and physical concurrency is achieved.

Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

A multithreading is a specialized form of multitasking. Multitasking threads require less overhead than multitasking processes.

There is one important term with threads which is process, a process consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non-daemon threads are done executing.

Multithreading enables us to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

4.3 Uses of Multithreading

Threads are mostly used in server side programming to serve the needs of multiple client in a network. On internet, server has to cater the needs of thousands of clients, at a time. For this purpose, threads are used to accomplish various task at a time, so that they can handle several clients.

Threads are also used in Games and animation programming or GUI Programming. In many games, generally we have to perform more than one task simultaneously. For example, in a game a flight may be moving from right to left. A rocket should shoot it, releasing the rockets at the flight. These two task should happen simultaneously. For this purpose we can use two threads, one thread will move the flight and the other one will move the rocket towards the flight simultaneously.

Now let check how java implements multithreading

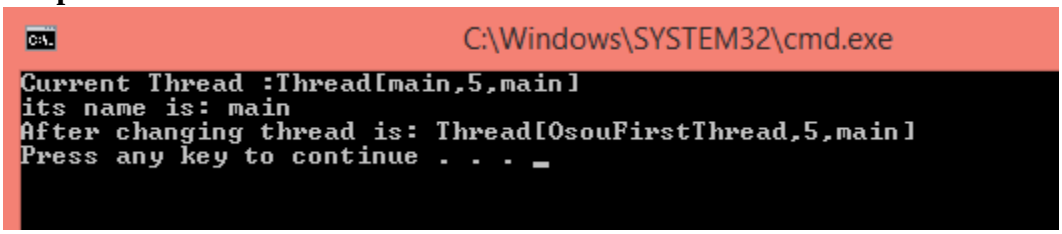
4.4 The Main Thread

Whenever we write a Java program, one thread starts running immediately, which is called the main thread of that program. Within the main thread we can create 'child' threads. The main thread is created automatically when your program is started. The main thread of Java programs is controlled through an object of Thread class.

Let understand with an example what is main thread is?

```
//program

class Osou_Thread
{
        public static void main (String args [ ])
        {
        Thread t = Thread.currentThread( );
        System.out.println("Current Thread :"+ t);
        System.out.println ("its name is: "+t.getName());

        //change the internal name of the thread
        t.setName("OsouFirstThread");
        System.out.println ("After changing thread is: "+t);
        }
}
```

Output:



Explanation:

In the above program, currentThread() is a static method in Thread class. So we called it as Thread.currentThread(). Which returns an object of Thread class i.e. "t", which display **Thread[main,5,main]** .

Here,

Thread point to the object of Thread class i.e. "t".

main: The first main shows the name of the current thread running in the current code.

5: this number represents the priority of the thread with is default priority value. These priority value ranges from 1 to 10.

main: main is also the name of the group of threads to which this thread belongs. A thread group is a data structure that controls the state of a collection of threads as a whole.

getName(): it is a method of Tread class which returns the name of the Current Thread associated with Thread object.

setName(): This method is used to change the name of the current thread associated with the Thread object.

4.5 Creating and Running Threads

As we know, every java program has a thread in it i.e. main Thread but we can also create our own threads in a program.

There are two ways in which a thread can be created:

- Using Runnable interface
- By extending the Thread class, itself

Thread can be created by implementing Runnable interface or by extends Thread class which is found in **java.lang** package.
To implement Runnable, a class needs only implement a single method called run( ), which is declared like this:

```
class Osou implements Runnable{
        public void run()
        {
                //statements
        }
}
Or
class Osou extends Thread{
        public void run()
        {
                //statements
        }
```

Create an object of class that implements Runnable, so that run() method will be availbale for execution.
Osou obj = new Osou();

After that, create an object of the Thread and attached the object of the class Osou.
Thread t= new Thread (obj);
Or

Thread t= new Thread (obj, threadName);

The name of the new thread is specified by threadName.
After the new thread is created, it will not start running until you call its start( ) method, which is declared within Thread. The start( ) method is shown here:
t.start( );

Now, the thread will start execution on the obj object of Osou class.

Example:

```
class Osou implements Runnable
{
  public void run() {
        try{
                for(int i = 1; i <= 1000; i++) {
                Thread.sleep(500);
                System.out.println("Child Thread: " + i);
                }
        }
        catch (InterruptedException e) {
                System.out.println("Child interrupted.");
                }
   }
}
class Example1
{
public static void main(String args[]) {
        Osou obj = new Osou();
        Thread t =new Thread(obj, "MyFirstThread");
        t.start();
}
}
```

Explanation:
In the above example we have created an object of class Osou that has extends the Runnable interface. Here, the object obj of class Osou contains the run() method. Then we have attached the object to thread t and assign a new name "MyFirstThread" as follows

Thread t =new Thread(obj, "MyFirstThread");
Whenever we call the start() method of thread class then the thread object calls the run() method where we have written code to display 1 to 1000.
If we try to terminate the program abnormally in the middle. We can do this externally by pressing **Ctrl + C**. it will terminate the whole program.
This abnormal termination may lead to loss of data and undependable results. So we need to terminate the thread to resolve the above problem.

4.6 How to terminate a Threads

Normally threads terminate itself automatically whenever the job of run() method completes or when the controls comes out from the run() method.
The alternate way is to write you own code to terminate the thread manually.
Let's understand the manual process of terminating a thread.

Example:

```
import java.io.*;
class Osou implements Runnable
{
boolean flag=false;
  public void run() {
        try{
                for(int i = 1; i <= 1000; i++) {
                Thread.sleep(500);
                System.out.println("Child Thread: " + i);
                if(flag) return;          // terminate the thread
                }
        }
        catch (InterruptedException e) {
                System.out.println("Child interrupted.");
                }
   }
}
class Example1
{
public static void main(String args[])throws IOException
{
        Osou obj = new Osou();
        Thread t =new Thread(obj, "MyFirstThread");
        t.start();
        System.in.read();               // wait till Enter key press
        obj.flag=true;
}
}
```

Explanation:
Here, whenever we press the enter key the flag value will change to "true" and the status will be checked in run() method and if it is true, it terminates the Thread.

4.7 Multiple Task using Single Threads

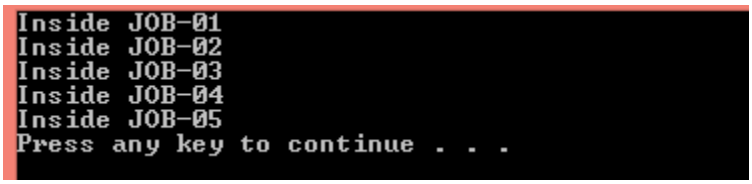
One thread can execute one task at a time. Suppose we have five job to execute. Then we will create a thread and assign all the job to it one by one.

For this we need to create five different method for each job and assign these methods to thread by calling it inside run() method.

Example:

```
class Osou implements Runnable
{
  public void run()
  {
                job1();
                job2();
                job3();
                job4();
                job5();
        }
        void job1(){
                System.out.println("Inside JOB-01");
        }
        void job2(){
                System.out.println("Inside JOB-02");
        }
        void job3(){
                System.out.println("Inside JOB-03");
        }
        void job4(){
                System.out.println("Inside JOB-04");
        }
        void job5(){
                System.out.println("Inside JOB-05");
        }
}
class Example2
{
public static void main(String args[])throws IOException
{
        Osou obj = new Osou();
        Thread t =new Thread(obj);
        t.start();

}
}
```

OUTPUT:



Explanation:

In the above program we had created five method for five different job i.e. job1, job2, job3, job4 and job5. Then all these methods has be called inside the run() method. Whenever we call the start() method it calls the run() method and execute these jobs one by one.

As there is a single Thread i.e. "t" is used to execute all five job so it will execute one by one.

4.8 Multiple Task using Multiple Threads

In multitasking, multiple job or task executes simultaneously so here in the above program if we want to execute the above five job simultaneously then we need to assign all task to different Threads. For example suppose we have two job and if we assign these task to two different thread then both the thread executes simultaneously.
Using of more than one thread is called Multi-Threading.

Example:

```
class Osou implements Runnable
{ String str;
      Osou(String str)
      {
            this.str=str;
      }
  public void run()
  {
            for(int i = 1; i <=5; i++) {
                  System.out.println(str +" number: " +i);
                  try{
                        Thread.sleep(500);
                        }
                  catch (InterruptedException e) {
                        System.out.println("Child interrupted.");
                        }
                  }
      }
}
```

```
class Example3
{
public static void main(String args[])
{
        Osou obj = new Osou("Message sent");
        Osou obj1 = new Osou("Message Received");

        Thread t =new Thread(obj);
        Thread t1 =new Thread(obj1);

        t.start();
        t1.start();

}
}
```

OUTPUT:

```
Message sent number: 1
Message Received number: 1
Message sent number: 2
Message Received number: 2
Message sent number: 3
Message Received number: 3
Message sent number: 4
Message Received number: 4
Message sent number: 5
Message Received number: 5
```

Explanation: In the above example we have passed two string "Message sent" and "Message Received" to the variable str of Osou class which will be assigned to str variable during constructor call. When obj object of class Osou created the value "Message Sent" will be assigned to str variable and when obj1 will be created "Message Received" will assigned to str variable of Osou class.

When t.start() executes it calls the run() method and print "Message Sent" and then encounter with Thread.sleep(500) which keep ideal the thread for .5 second. During this ideal time t1.start() executes and calls the run() method which displays "Message Received" and again it goes to ideal state for .5 second. In this .5 second the first thread t get active and executes its code. This way they executes both threads t and t1 simultaneously.

Here, sleep is a static method of Thread class that halt the execution of the thread for specified time.

4.9 Multiple Task using Multiple Threads and Single Object

In the Multiple task using multiple Thread concept we need to create two object of Osou class and two thread to run the thread simultaneously. But it is possible to run two thread on one object. But the problem is some we may get unpredictable results.

When two thread want to perform same task then we can create one object and

execute the same run() method twice.

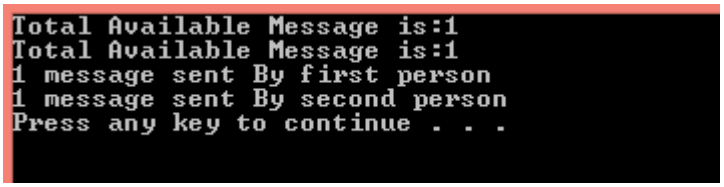
Let's take an example of message sending. Suppose we have only 10 message to send by two different thread. As all the message need to send from a single stock of 10 message so both the thread can access the same run() method for message sending.

Lets understand with an example

Example:

```
class Osou implements Runnable
{
int total=1;
int need;
        Osou(int n)
        {
                need=n;
        }
  public void run()
  {
          try{
                System.out.println("Total Available Message is:"+total);
                if(total>=need){
                        String name= Thread.currentThread().getName();
                        System.out.println(+need+" message sent "+name);
                        Thread.sleep(1000);
                        total=total-need;
        }
                else{System.out.println("Sorry Available bearth is less than required");
        }
        }
        catch (InterruptedException e) {System.out.println("Child interrupted.");}
        }
}
class Example5
{
public static void main(String args[])
{
        Osou obj = new Osou(1);

                Thread t1 =new Thread(obj,"By first person");
                Thread t2 =new Thread(obj,"By second person");

                t1.start();
                t2.start();

}
}
```

OUTPUT:



Explanation: Please notice the output of the above example. Here, though we have 2 message to send by both the thread one by one but still it shows available message is 1 thwice and message is sent twice, one for thread 1 and one for thread 2 .
This happens because because when thread t1 executes the run() method, the thread2 also executes the same run() method at the same time. Which is a wrong result and it was not our requirements.

So How can we solve this problem?
The Answer is, if we keep the thread2 (second person) wait till the first thread t (first Thread) completes and comes out. Let us not allow other thread to access the object till thread "t" comes out. This is called **Thread Synchronization** or **Thread Safe**.

4.10 Threads Syncronization

The synchronization means that a thread must wait for another thread to leave the critical section and to enter into this section using some security measures, e.g. locking the critical section when entering it.
Java uses word synchronized to synchronize threads and intercommunicate to each other. It is basically a mechanism which allows two or more threads to share all the available resources in a sequential manner. Java's synchronized is used to ensure that only one thread is in a critical region. Critical region is a lock area where only one thread is run (or lock) at a time. Once the thread is in its critical section, no other thread can enter to that critical region. In that case, another thread will has to wait until the current thread leaves its critical section.

There are many situations in which multiple threads must share access to common objects. There are times when you might want to coordinate access to shared resources. For example, in a database system, you would not want one thread to be updating a database record while another thread is trying to read from the database. Java enables you to coordinate the actions of multiple threads using synchronized methods and synchronized statements. Synchronization provides a simple monitor facility that can be used to provide mutual-exclusion between Java threads.
So how can we do this in programming?

Once you have divided your program into separate threads, you need to define how they will communicate with each other. Synchronized methods are used to coordinate access to objects that are shared among multiple threads. These methods are declared with the

synchronized keyword. Only one synchronized method at a time can be invoked for an object at a given point of time. When a synchronized method is invoked for a given object, it acquires the monitor for that object. In this case no other synchronized method may be invoked for that object until the monitor is released. This keeps synchronized methods in multiple threads without any conflict with each other.

There are two way to accomplish this.
Using Synchronized block: here, we add a number of codes of the object within a synchronized block as follows
synchronized(object){
// statements to be synchronized }
Here, object represents the object that will be locked or synchronized.

Using Synchronized Keyword: we can use synchronized keyword for a method. if we want to synchronized the statements of Job() method then just add the synchronized keyword before the method as follows

Synchronized void job()
{
// statements
}
Now the statement inside job() method are no more available to more than one thread at a time.
Lets understand the above an example with synchronize block:
Example:

```
class Osou implements Runnable
{
int total=1;
int need;
        Osou(int n)
        {
                need=n;
        }
  public void run()
  {
          synchronized(this){
          try{
                  System.out.println("Total Available Message is:"+total);
                if(total>=need){
                        String name= Thread.currentThread().getName();
                        System.out.println(+need+" message sent "+name);
                        Thread.sleep(1000);
                        total=total-need;
        }
                else{System.out.println("Sorry Available bearth is less than required");
        }
```

```
        }
        catch (InterruptedException e) {System.out.println("Child interrupted.");}
        }
}

}
class Example5
{
public static void main(String args[])
{
        Osou obj = new Osou(1);

                Thread t1 =new Thread(obj,"By first person");
                Thread t2 =new Thread(obj,"By second person");

                t1.start();
                t2.start();

}
}
```

OUTPUT:

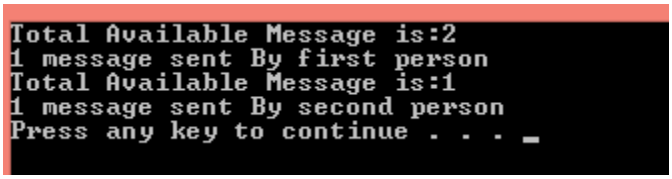
```
Total Available Message is:1
1 message sent By first person
Total Available Message is:0
Sorry Available bearth is less than required
Press any key to continue . . .
```

Example:

```
class Osou implements Runnable
{int total=2;
 int need;
        Osou(int n)
        {need=n;}
  public void run()
  {
          job();
  }
  synchronized void job()   {
          try{
                  System.out.println("Total Available Message is:"+total);
                if(total>=need){
                        String name= Thread.currentThread().getName();
                        System.out.println(+need+" message sent "+name);
                        Thread.sleep(1000);
                        total=total-need;
                        }
                else{System.out.println("Sorry Available bearth is less than required");
```

```
                        }
                }
        catch (InterruptedException e) {System.out.println("Child interrupted.");}
        } // end of synchronized block
}
class Example5
{
public static void main(String args[])
{
        Osou obj = new Osou(1);

                Thread t1 =new Thread(obj,"By first person");
                Thread t2 =new Thread(obj,"By second person");
                t1.start();
                t2.start();
}
}
```

OUTPUT:



Explanation: Now we got the proper output as per our requirements. Now the synchronized block allow only one thread to execute its run() method at time so once the total number of available message is updated, the second thread gets the access of run() method.

4.11 Threads Priority

Java assigns to each thread a priority. Thread priority determines how a thread should be treated with respect to others. A priority that helps the operating system determine the order in which threads are scheduled. Thread priority is an integer that specifies the relative priority of one thread to another. The highest- priority thread gets the chance to use the CPU.

A higher-priority thread can pre-empt a low priority thread. In this case, a lower-priority thread that does not yield the processor is forcibly pre-empted. In cases where two threads with the same priority are competing for CPU cycles, the situation is handled differently by different operating systems.
Java priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5). Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependent.

When a thread is created, it inherits its priority from the thread that created it. In addition, by using the **setPriority** method, thread's priority can be modified at any time after its creation.
The chosen thread runs until one of the following conditions is true:

- A higher-priority thread becomes runnable.
- The thread yields, or its run method exits.
- On systems that support time-slicing, the thread's time allotment has expired.

Example:

```
class Osou implements Runnable
{ String str;
        Osou(String str)
        {this.str=str;
        }
  public void run()
  {
        for(int i = 1; i <=25; i++) {
        System.out.println(str +" number: " +i);
        }
  }}
class Example3
{
public static void main(String args[])
{
        Osou obj = new Osou("Message sent");
        Osou obj1 = new Osou("Message Received");
        Thread t =new Thread(obj);
        Thread t1 =new Thread(obj1);
        t1.setPriority(2);
        t.setPriority(8);
        t1.start();
        t.start();
} }
```

OUTPUT:

```
Message Received number: 1
Message Received number: 2
Message Received number: 3
Message Received number: 4
Message sent number: 1
Message sent number: 2
Message Received number: 5
Message sent number: 3
Message sent number: 4
Message Received number: 6
Message sent number: 5
Message Received number: 7
Message Received number: 8
Message sent number: 6
Message sent number: 7
Message sent number: 8
Message Received number: 9
Message sent number: 9
Message Received number: 10
Message sent number: 10
Message Received number: 11
Message sent number: 11
Message Received number: 12
Message sent number: 12
Message Received number: 13
Message sent number: 13
Message Received number: 14
Message sent number: 14
Message sent number: 15
Message Received number: 15
Message sent number: 16
Message Received number: 16
Message sent number: 17
Message Received number: 17
Message sent number: 18
Message Received number: 18
Message sent number: 19
Message sent number: 20
Message sent number: 21
Message sent number: 22
Message Received number: 19
Message sent number: 23
Message Received number: 20
Message sent number: 24
Message Received number: 21
Message sent number: 25
Message Received number: 22
Message Received number: 23
Message Received number: 24
Message Received number: 25
Press any key to continue . . .
```

Explanation: Here**,** in the above example we have two object and two thread to perform two different task that is “Message Sent” and “Message Received”**.**
Though we have executed t1 thread first then t thread b
“Message Sent” has completed first compared to “Message Received” because of its priority level. The priority of thread t is 8 whereas priority of thread t1 is 2. So thread t execution completes first then thread t1.

4.12 Thread Methods

Following is the list of important methods available in the Thread class.

Methods with Description

public void start()
Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.

public void run()
If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.

public final void setName(String name)
Changes the name of the Thread object. There is also a getName() method for retrieving the name.

public final void setPriority(int priority)
Sets the priority of this Thread object. The possible values are between 1 and 10.

public final void setDaemon(boolean on)
A parameter of true denotes this Thread as a daemon thread.

public final void join(long millisec)
The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.

public void interrupt()
Interrupts this thread, causing it to continue execution if it was blocked for any reason.

public final boolean isAlive()
Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread.

Methods with Description
public static void yield()
Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled

public static void sleep(long millisec)
Causes the currently running thread to block for at least the specified number of milliseconds

public static boolean holdsLock(Object x)
Returns true if the current thread holds the lock on the given Object.

public static Thread currentThread()
Returns a reference to the currently running thread, which is the thread that invokes this method.

public static void dumpStack()
Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

4.13 Threads Communication

Java provides a very efficient way through which multiple-threads can communicate with each-other. This way reduces the CPU's idle time.
In some situation, more than one thread need to communicate with each other for completion of task. If both are dependent to each other.
For Example: A customer thread waits for a Tailor thread to complete its task (stitching of shirt). When the tailor complete its stitching of shirt then the customer use that shirt.
This technique is known as Inter-thread communication which is implemented by some methods. These methods are defined in "java.lang" package and can only be called within synchronized code shown as:

wait( ) It indicates the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls method notify() or notifyAll().

notify( ) It wakes up the first thread that called wait() on the same object.

notifyAll( ) Wakes up (Unlock) all the threads that called wait( ) on the same object. The highest priority thread will run first.
These methods are implemented as final methods. These three methods can be called only from within a synchronized method.
These methods enable you to place threads in a waiting pool until resources become available to satisfy the Thread's request. A separate resource monitor or controller process can be used to notify waiting threads that they are able to execute.

Let us see this example program written to control access of resource using wait() and notify () methods.

Example:

```
class Tailor extends Thread
{
StringBuffer sb;
Tailor()
{
        sb=new StringBuffer();
}
public void run()
{
        synchronized(sb)
        {
        for(int i=1; i<=10; i++)
        {
        try{
                sb.append(i+":");
                Thread.sleep(1000);
                System.out.println("Tailor compleates "+i+" Stitching ");
          }
        catch(Exception e){}
        } // end of for loop
        sb.notify();
    } // end of synchronized block
  } // end of run() method
} // end of Tailor class

class Customer extends Thread
{
        Tailor shirt;
        Customer(Tailor shirt)
        {
                this.shirt=shirt;
        }
        public void run()
        {
                synchronized(shirt.sb)
                {
                try{
                System.out.println("Tailor Has not Yet started Stitching");
                shirt.sb.wait();
                }
                catch(Exception e){}
                System.out.print("Now Customer Receives all 10 Shirts i.e.: ");
                System.out.println(shirt.sb);
                }
        }
}
```

```
class Tailor_Customer
{
	public static void main(String args[])throws Exception
	{
		Tailor obj= new Tailor();
		Customer obj1=new Customer(obj);

		Thread t1=new Thread(obj);
		Thread t2=new Thread(obj1);

		t2.start();
		t1.start();
	}
}
```

OUTPUT:

```
Customer has given 10 Shirt for Stiching...
Tailor compleates 1 Stitching
Tailor compleates 2 Stitching
Tailor compleates 3 Stitching
Tailor compleates 4 Stitching
Tailor compleates 5 Stitching
Tailor compleates 6 Stitching
Tailor compleates 7 Stitching
Tailor compleates 8 Stitching
Tailor compleates 9 Stitching
Tailor compleates 10 Stitching
Now Customer Receives all 10 Shirts i.e.: 1:2:3:4:5:6:7:8:9:10:
Press any key to continue . . .
```

Explanation: In the above example of Tailor and customer, customer has giver 10 shirt for stitching to tailor. Initially customer will give his cloth to Tailor for stitching and it will wait till completion. When Tailor complete his job he notify to customer to receive and then finally customer receives all the shirts.

So in the program, first we have call the run() method for customer to submit the cloth to tailor and wait till it completes this archives by following code.

System.out.println("Tailor Has not Yet started Stitching");

shirt.sb.wait();
Customer will remain in waiting till it gets a notification of completion.
Then we call the run() method for Tailor thread. The tailor method complete its task one by one and store in a StringBuffer by appending into it.
Then when all 10 stitching completes he sends a notification to customer that he has completed histask and enter in to sleep state. The code for this is

```
for(int i=1; i<=10; i++)
	{
	try{
```

```
                sb.append(i+":");
                Thread.sleep(1000);
                System.out.println("Tailor compleates "+i+" Stitching ");
            }
          catch(Exception e){}
          }
sb.notify();
```

Then customer thread wakeup from the waiting state and receives its product from the StringBuffer as follows

System.out.println(shirt.sb);

4.14 Threads Life Cycle

The benefit of Java's multithreading is that a thread can pause without stopping other parts of your program. A paused thread can restart. A thread will be referred to as dead when the processing of the thread is completed. After a thread reaches the dead state, then it is not possible to restart it.

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.

These state are:

Ready State

When a thread is created, it doesn't begin executing immediately. You must first invoke start ( ) method to start the execution of the thread. In this process the thread scheduler must allocate CPU time to the Thread. A thread may also enter the ready state if it was stopped for a while and is ready to resume its execution.

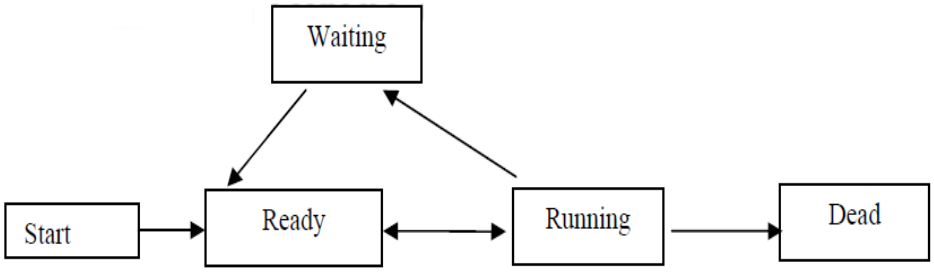
Running State

Threads are born to run, and a thread is said to be in the running state when it is actually executing. It may leave this state for a number of reasons, which we will discuss in the next section of this unit.

Waiting State

A running thread may perform a number of actions that will cause it to wait. A common example is when the thread performs some type of input or output operations.

As you can see in Figure given below, a thread in the waiting state can go to the ready state and from there to the running state. Every thread after performing its operations has to go to the dead state.



A thread begins as a ready thread and then enters the running state when the thread scheduler schedules it. The thread may be prompted by other threads and returned to the ready state, or it may wait on a resource, or simply stop for some time. When this happens, the thread enters the waiting state. To run again, the thread must enter the ready state. Finally, the thread will cease its execution and enter the dead state.

4.15 Let Us Sum Up

This unit described the working of multithreading in Java. Also you have learned what the main thread is and when it is created in a Java program. Different states of threads are described in this unit. This unit explained how threads are created using Thread class and Runnable interface. It explained how thread priority is used to determine which thread is to execute next. This unit explains concept of synchronization, creating synchronous methods and inter thread communication. It is also explained how object locks are used to control access to shared resources.

4.16 Self-Assessment Questions

Q.1. what is Thread? What are the advantage of Multithreading?
A ……..

……..

……..

……..

……..

Q.2. How does multithreading take place on a computer with a single CPU?
A ……..

……..

……..

……..

……..

Q.3. How a thread is created by extending the Thread class and implementing Runnable interface.
A ……..

……..

……..

……..

……..

Q.4. Why do we need of synchronized method? Explain.
A ……..
……..
……..

……..

……..

……..

……..

……..

Q.5. Describe the life cycle of a thread.
A ……..

……..

……..

……..

……..

……..

Q.6. What is inter thread communication?
A ……..

……..

……..

……..

……..

……..

4.17 References

1. http://math.hws.edu/javanotes/c12/s1.html
2. http://egyankosh.ac.in/bitstream/123456789/10102/1/Unit-1.pdf (IGNOU)
3. Core Java: An Integrated Approach By R. Nageswara Rao
4. Programming in JAVA, Vardhaman Mahaveer Open University, Kota