Third Year Diploma Courses in Computer Science & Engineering, Computer Engineering, Computer Technology and Information Technology Branch.

Java Programming

As per MSBTE 'I' Scheme Syllabus

JPR-22412

Unit IV

Exception Handling & Multithreading

Total Marks- 12

Contents:

Prof. Gunwant V. Mankar
B.E(IT), M.Tech(CSE), AMIE, MIAEng, MSCI
HOD in CO Dept.
(BTC- School of Diploma in Engineering, Ballarpur)
e-mail:- info@gunwantmankar.com
Website:- www.gunwantmankar.com

Topic 4 Exception Handling & Multithreaded Programming

1. Error and Exception:-

1.1. Types of Error [W-14]

Errors are broadly classified into two categories:-
1. Compile time errors
2. Runtime errors

1.1.1. Compile time error-

All syntax errors will be detected and displayed by java compiler and therefore these errors are known as compile time errors.

The most of common problems are:
- Missing semicolon
- Missing (or mismatch of) bracket in classes & methods
- Misspelling of identifiers & keywords
- Missing double quotes in string
- Use of undeclared variables.
- Bad references to objects.

1.1.2. Runtime error

Sometimes a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow. When such errors are encountered java typically generates an error message and aborts the program.

The most common run-time errors are:
- Dividing an integer by zero
- Accessing an element that is out of bounds of an array
- Trying to store value into an array of an incompatible class or type
- Passing parameter that is not in a valid range or value for method
- Trying to illegally change status of thread
- Attempting to use a negative size for an array
- Converting invalid string to a number
- Accessing character that is out of bound of a string

1.2. Exception [W-15, S-15]

An exception is an event, which occurs during the execution of a program, that stop the flow of the program's instructions and takes appropriate actions if handled.

1.2.1. Exception Handling [W-15, S-15, W-14]

Java handles exceptions with 5 keywords:
1) try 2) catch 3) finally 4) Throw 5) throws

1) **try**:

This block applies a monitor on the statements written inside it. If there exist any exception, the control is transferred to catch or finally block.

Syntax:
```
try
{
    // block of code to monitor for errors
}
```

2) **catch**:

This block includes the actions to be taken if a particular exception occurs.

Syntax:

```
catch(ExceptionType1 exOb)
{
    // exception handler for ExceptionType1
}
```

3) **finally**:

finally block includes the statements which are to be executed in any case, in case the exception is raised or not.

Syntax:

```
finally
{
        // block of code to be executed before try block ends
}
```

4) **throw**:

This keyword is generally used in case of user defined exception, to forcefully raise the exception and take the required action.

The general form of throw is :

```
throw new ThrowableInstance;
        or
throw Throwableinstance;
```

throw statement explicitly throws an built-in /user- defined exception. When throw statement is executed, the flow of execution stops immediately after throw statement, and any subsequent statements are not executed.

5) **throws**:**[S-16]**

throws keyword can be used along with the method definition to name the list of exceptions which are likely to happen during the execution of that method. In that case , try … catch block is not necessary in the code.

General form of method declaration that includes "Throws" clause

```
Type method-name (parameter list) throws exception list
    {
            // body of method} Here exception list can be separated by a comma.
    }
```

Eg.

```
class XYZ
        {
        XYZ();
        {
        // Constructor definition
        }
        void show() throws Exception
        {
        throw new Exception()
        }
```

```
                }
```

Example for Exception handling

```
class DemoException
{
        public static void main(String args[])
         {
                try
                {
                int b=8;
                int c=b/0;
                System.out.println("answer="+c);
                }
                catch(ArithmeticException e)
                 {
                System.out.println("Division by Zero");
                }
        }
 }
```

1.3 Build in Exceptions

Inside the standard package java.lang, Java defines several exception classes. A few have been used by the preceding examples. The most general of these exceptions are subclasses of the standard type RuntimeException.

These exceptions need not be included in any method's throws list, these are called *unchecked exceptions* because the compiler does not check to see if a method handles or throws these exceptions. The unchecked exceptions defined in java.lang are listed in Table 1.

Table 2 lists those exceptions defined by java.lang that must be included in a method's throws list if that method can generate one of these exceptions and does not handle it itself. These are called *checked exceptions.*

Table 1. Unchecked Runtime Exceptions

Exception	**Meaning**
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread
IllegalStateException	Environment or application is in incorrect state.

IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered

Table 2. Checked Exceptions

Exception	Meaning
ClassNotFoundException	Class not found
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable interface.
IllegalAccessException	Access to a class is denied
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

1.4 Chained Exceptions

- When java program causes an exception, then the exception responds to an exception by throwing another exception.
- Therefore, the first generated exception causes for throwing another exception in turn. This technique is known as chained exception which is introduced in JDK1.4.
- With the help of chained exception it may be easier to know that when one exception causes another exception.
- The two constructor of Throwable class supports chained exception are listed below:

1) Throwable(Throwable cause):
   The parameter cause specifies the actual cause of exception
2) Throwable(String str, Throwable cause):

The string str specifies exception description and cause specifies the actual cause of exception.

Two methods of Throwable class that supports the chained exception.

1) Throwable gtCause( ):
   This method returns the actual cause of the currently generated exception.
2) Throwable initCause( ):
   This method sets the fundamental exception with invoking exception.

Example :-

The following java program shows that the program generates ArithmeticException which in turn generates NumberFormatException.

```
class Demo
        {
        public static void main(String args[])
                {
                int n=10, result=0;
                try
                {
                System.out.println("The result is"+result);
                }
                catch(ArithmeticException ex)
                {
                System.out.println("Arithmetic exception occoured:"+ex);
                try
                {
                throw new NumberFormatException( );
                }
                catch(NumberFormatException ex1 )
                {
                System.out.println("Chained exception thrown manually:"+ex1)
                }
                }
        }

        }
```

Output-

```
Arithmetic exception occurred:
Java.lang.ArithmeticException:/ by zero
Chained exception thrown manually
Java.lang.NumberFormatException
```

1.5 Creating own Exception Subclasses

- If we want to create our own exception types to handle situations specific to our applications. This is quite easy to do: just define a subclass of Exception (which is a subclass of Throwable).

- Our subclasses don't need to actually implement anything—it is their existence in the type system that allows us to use them as exceptions.
- The Exception class does not define any methods of its own. It does, inherit those methods provided by Throwable.
- Thus, all exceptions, including those that we create, have the methods defined by Throwable available to them.

Exception defines four constructors. Two were added by JDK 1.4 to support chained exceptions.

The other two are shown here:
Exception( )
Exception(String *msg*)

The first form creates an exception that has no description. The second form specify a description of the exception.

- The following example declares a new subclass of Exception and then uses that subclass to signal an error condition in a method. It overrides the toString( ) method, allowing a carefully tailored description of the exception to be displayed.

// This program creates a custom exception type.

```
class MyException extends Exception
{
private int detail;
MyException(int a)
{
detail = a;
}
public String toString()
{
return "MyException[" + detail + "]";
}
}
class ExceptionDemo
{
static void compute(int a) throws MyException
{
System.out.println("Called compute(" + a + ")");
if(a > 10)
throw new MyException(a);
System.out.println("Normal exit");
}
public static void main(String args[ ])
{
try
{
compute(1);
compute(20);
}
catch (MyException e)
{
System.out.println("Caught " + e);
}
}
}
```

1.6 Important Programs on Exceptions

1. **Write a program to input name and age of a person and throws an user define exception if entered age is negative. [S-16, S-15]**

```
import  java.io.*;

class Negative extends Exception
{
        Negative(String msg)
        {
        super(msg);
        }
}

class NegativeDemo
{
public static void main(String args[])
{
int age=0;
String name;
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("enter age and name of person");
try
{
        age=Integer.parseInt(br.readLine());
        name=br.readLine();
        {
        if(age<0)
        throw new Negative("age is negative");
        else
        throw new Negative("age is positive");
        }
        }
catch(Negative n)
        {
         System.out.println(n);
        }
        catch(Exception e)
        {
        }
}
}
```

2. **Write a program to accept password from user and throw 'Authentication failure' exception if password is incorrect. [W-15]**

```
import java.io.*;

class PasswordException extends Exception
        {
        PasswordException(String msg)
        {
        super(msg);
        }
        }
class PassCheck
{
 public static void main(String args[])
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
try
{
System.out.println("Enter Password : ");
        if(br.readLine().equals("EXAMW16"))
        {
        System.out.println("Authenticated ");
        }
        else
        {
        throw new PasswordException("Authentication failure");
        }
        }
catch(PasswordException e)
        {
        System.out.println(e);
        }
catch(IOException e)
        {
        System.out.println(e);
        }
}
}
```

2. 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. Thus, multithreading is a specialized form of multitasking.

2.1 Creating thread

We can create a thread by instantiating an object of type Thread. Java defines two ways in which this can be accomplished:

1. Implement the Runnable interface.
2. Extends the Thread class, itself.

2.1.1. Implement the Runnable interface

First define a class that implements the Runnable interface. The Runnable interface contains only one method, run(). The run() method is defined in method with the code to be executed by the thread.

Threads are implemented in form of objects that contain a method called run(). The run() method makes up entire body of thread and is the only method in which the threads behaviour can be implemented.

A run method appears as follows

```
public void run( )

        {
            ---------
        }
```

2.1.2. Extends the Thread class itself.

Define a class that extends Thread class and override its run() method.

2.2 Extending the Threads class

- To create thread is to create new class that extends Thread and then to create an instance of that class.
- Extending the Thread class includes following steps

i) Declare the class extending the Thread class.

The Thread class can be extended as follows

```
class MyFirstThread extends Thread
{
   --------
}
```

ii) Implements the run() method

That is responsible for executing the sequence of code that the thread with execute. The extending class must override the run() method, which is the entry point of new thread.

```
public void run()
{
-------- //Thread code
}
```

iii) Create a thread object and call the start() method to initiate the thread execution.

Following statement create new thread

a) MyFirstThread t= new MyFirstThread();
   Run instance of thread class or involve run()

b) T.start( );

- **Example for creating and extending the Thread class**

```
class A extends Thread
   {
   public void run()
           {
           for(int i=1;i<=5; i++)
           {
           System.out.println("From thread A: i=" +i);
           }
           System.out.println("Exit from A");
           }
   }

class B extends Thread
   {
   public void run()
           {
   for(int j=1; j<=5; j++)
           {
           System.out.println("From thread B: j=" +j);
           }
           System.out.println("Exit from B");
           }
   }

class ThreadDemo
   {
        public static void main(String args[])
        {
         new A().start();
        new B().start();
        }
   }
```

Output –

```
From Thread A:i=1
From Thread A:i=2
From Thread A:i=3
From Thread A:i=4
From Thread A:i=5
Exit from A
From Thread B:j=1
From Thread B:j=2
From Thread B:j=3
From Thread B:j=4
From Thread B:j=5
```

2.3 Implementing the “Runnable” Interface

The easiest way to create a thread is to create a class that implements the Runnable interface. Runnable abstracts a unit of executable code. Threads can be created by implementing the Runnable interface. The Runnable interface declares the run() method i.e required for implementing threads in program.

Follow given steps:

1. Declare the class as it is implementing Runnable interface.
2. Implement run() method.
3. Create thread by defining an object that instantiated from this “runnable” class as the target of the thread.
4. Call the thread’s start() method to run the thread.
5. The run call other methods, use other classes and declare variables, just like the main thread class.

Example for implementing ‘Runnable’ interface

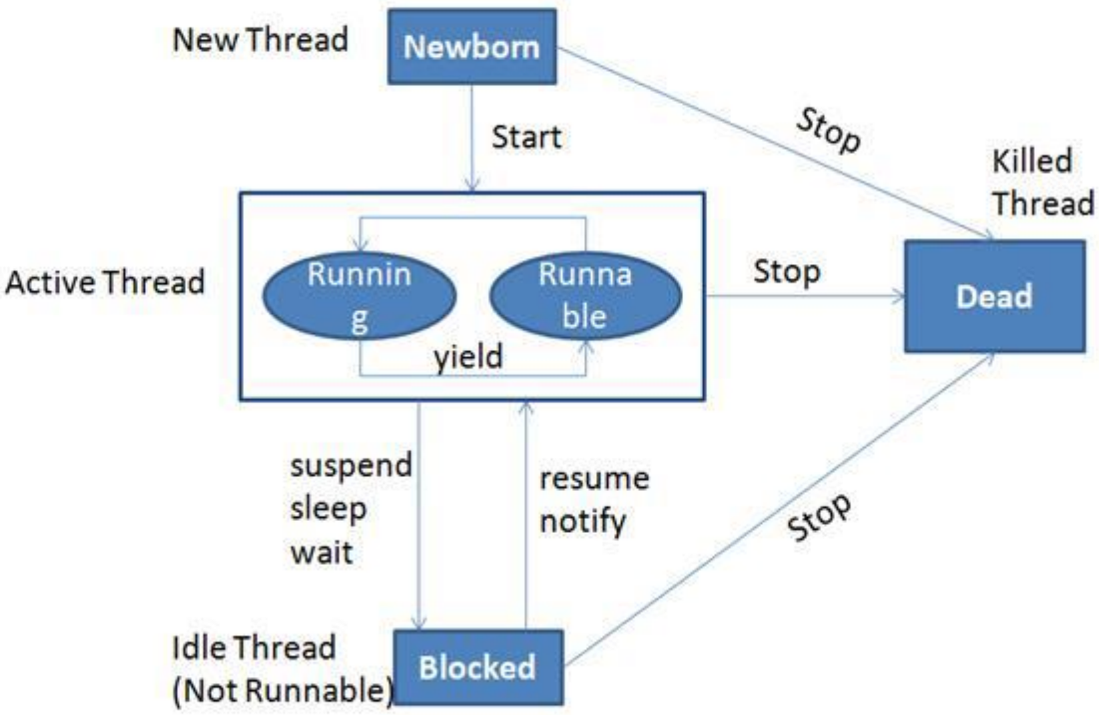
```
class A implements Runnable
        {
        public void run()
                {
                 for(int i=1; i<=10; i++)
                {
                System.out.println(“Thread A:” +i);
                }
                System.out.println(“End of Thread A:”);
                }
        }
class RunnableDemo
        {
        public static void main(String args[])
        {
        A obj = new A();
        Thread  t= new Thread(obj);
        A.start();
        System.out.println(“End of main Thread:”);
        }
        }
```

2.4 Life Cycle of Thread [S-15, S-16, W-14, W-15]

Thread Life Cycle Thread has five different states throughout its life.

1. Newborn State
2. Runnable State
3. Running State
4. Blocked State
5. Dead State

Thread should be in any one state of above and it can be move from one state to another by different methods and ways.



1. **Newborn state:**
   - When a thread object is created it is said to be in a new born state. When the thread is in a new born state it is not scheduled running from this state it can be scheduled for running by start() or killed by stop(). If put in a queue it moves to runnable state.
2. **Runnable State:**
   - It means that thread is ready for execution and is waiting for the availability of the processor i.e. the thread has joined the queue and is waiting for execution. If all threads have equal priority then they are given time slots for execution in round robin fashion.
   - The thread that relinquishes control joins the queue at the end and again waits for its turn. A thread can relinquish the control to another before its turn comes by yield().
3. **Running State:**
   - It means that the processor has given its time to the thread for execution. The thread runs until it relinquishes control on its own or it is pre-empted by a higher priority thread.
4. **Blocked state:**

- A thread can be temporarily suspended or blocked from entering into the runnable and running state by using either of the following thread method.
- **suspend()** : Thread can be suspended by this method. It can be rescheduled by resume().
- **wait():** If a thread requires to wait until some event occurs, it can be done using wait method and can be scheduled to run again by notify().
- **sleep():** We can put a thread to sleep for a specified time period using sleep(time) where time is in ms. It re-enters the runnable state as soon as period has elapsed /over

5. **Dead State:**

- Whenever we want to stop a thread form running further we can call its stop(). The statement causes the thread to move to a dead state. A thread will also move to dead state automatically when it reaches to end of the method. The stop method may be used when the premature death is required

2.5 Thread Methods: [W-14]

1) suspend(), 2) resume(), 3) yield(), 4) wait()

1) **suspend() -**

syntax : public void suspend()

This method puts a thread in suspended state and can be resumed using resume() method.

2) resume()

syntax : public void resume()

This method resumes a thread which was suspended using suspend() method.

3) yield()

syntax : public static void yield()

The yield() method causes the currently executing thread object to temporarily pause and allow other threads to execute.

4) wait()

syntax : public final void wait()

This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

2.6 Thread Exceptions

Mostly call to sleep() method is enclosed within a try block and followed by catch block. This is because the sleep() method threads an exception, which must be caught. If exception thread is not catched, program, will not compile.

Java run system will throw IllegelThreadStateException whenever attempt is made to invoke a method that a thread cannot handle in given state.

For example, a sleeping thread cannot deal with the resume() method because a sleeping thread cannot receive any instructions. Even for suspend()

method when it is used on a block or NotRunnable thread. When thread method is called which is throwing an exception the appropriate exception handler must be supplied which catch exception thrown by thread.
The catch statement can take any one of the following four forms

1. catch(ThreadDeath e)
```
{
-------- //Killed thread
}
```

2. catch(InterruptedException e)
```
{
-------- //cannot handle it in current state
}
```

3. catch(IllegalArgumentException e)
```
{
---------- //Illegal method argument
}
```

4. catch(Exception e)
```
{
--------- // any other
}
```

2.7 Thread priority & methods [S-15, S-16]

Thread Priority:

Threads in java are sub programs of main application program and share the same memory space. They are known as light weight threads. A java program requires at least one thread called as main thread. The main thread is actually the main method module which is designed to create and start other threads in java each thread is assigned a priority which affects the order in which it is scheduled for running. Thread priority is used to decide when to switch from one running thread to another. Threads of same priority are given equal treatment by the java scheduler.

Thread priorities can take value from 1-10. Thread class defines default priority constant values as

```
MIN_PRIORITY = 1
NORM_PRIORITY = 5 (Default Priority)
MAX_PRIORITY = 10
```

Thread Methods:

1. setPriority:

Syntax: public void setPriority(int number);
This method is used to assign new priority to the thread.

2. getPriority:

Syntax: public int getPriority();
It obtain the priority of the thread and returns integer value.

Example- Demonstrating thread priority

```
class A extends Thread
   {
   public void run()
            {
            System.out.println("Thread A started ");
            for(int i=1; i<=5; i++)
            {
            System.out.println("From thread A: i=" +i);
            }
            System.out.println("Exit from A");
            }
   }

class B extends Thread
   {
   public void run()
            {
            System.out.println("Thread B started ");
            for(int j=1; j<=5; j++)
            {
            System.out.println("From thread B:j=" +j);
            }
            System.out.println("Exit from B");
            }
   }

class C extends Thread
   {
   public void run()
            {
            System.out.println("Thread C started ");
            for(int k=1; k<=5; k++)
            {
            System.out.println("From thread C: k=" +k);
            }
            System.out.println("Exit from C");
            }
   }
class ThreadPriorityDemo
   {
   public static void main(String args[])
            {
                     A  objA= new A( ).;
                     B  objB= new B( ).;
                     C  objC= new C( ).;
```

```
                objC.setPriority(Thread.MAX_PRIORITY);
                objB.setPriority(objA.getPriority() +1);
                objA.setPriority(Thread.MIN_PRIORITY);

                System.out.println("Start Thread A");
                objA.start();
                System.out.println("Start Thread B");
                objB.start();
                System.out.println("Start Thread C");
                objC.start();
                System.out.println("End of  main Thread");

        }
}
```

2.8 Synchronization

When two or more threads wants to access to a shared resources, there is requirement of the shared resource must be used by only one thread at a time. The synchronization process is used to achieve this.

Syntax-

```
synchronized  returntype methodname()
{
------ //code here is synchronized
}
```

2.9 Inter thread communication

When a two or more thread exits in an application and they are communicating each other by exchanging information, then it is called as inter thread communication.

2.10 Deadlock

When two threads have a circular dependency on a pair of synchronized objects, then there is deadlock. When two or more threads are waiting to gain control of resource. Due to some reasons, the condition on which the waiting threads rely on to gain control does not happen. The result is called as deadlock.

2.11 Important Programs

1) Write a program to create two threads; one to print numbers in original order and other to reverse order from 1 to 50. [W-14]

```
class original extends Thread
{
        public void run()
        {
        for(int i=1; i<=50;i++)
```

```
          {
          System.out.println(" First Thread="+i);
          }
          }
}
class reverse extends Thread
{
          public void run()
          {
          for(int  j=50; i >=1; i--)
          {
          System.out.println(" Second Thread="+i);
          }
          }
}
class orgrevDemo
{
          public static void main(String args[])
          {
          System.out.println("first thread printing 1 to 50 in ascending order");
          new  original().start();
          System.out.println("second thread printing 50 to 1 in reveres order");
          new  reverse().start();
          System.out.println("Exit from Main");
          }
}
```

2) Write a program to create two threads so one thread will print 1 to 10 numbers whereas other will print 11 to 20 numbers.

```
class A extends Thread
{
          public void run()
          {

          for(int i=1; i<=10;i++)
          {
          System.out.println(" First Thread ="+i);
          }
          }
}
class B extends Thread
{
          public void run()
          {
          for(int  j=11; j<=20; j++)
          {
          System.out.println(" Second Thread="+j);
          }
          }
}
```

```
class orgrevDemo
{
        public static void main(String args[])
        {
        System.out.println("first thread printing 1 to 10 ");
        new  A().start();
        System.out.println("second thread printing 11 to 20 ");
        new  B().start();
        System.out.println("Exit from Main");
        }
}
```

Important Questions:-

1) Explain thread priority and method to get and set priority values.
2) Describe life cycle of thread.(4 times)
3) Describe use of ‘throws’ with suitable example.
4) Explain following methods related to threads : 1) suspend ( ) 2) resume ( ) 3)yield( ) 4) wait ( )
5) What is exception ? How it is handled ? Explain with suitable example.(2 times)
6) Explain the following clause w.r.t. exception handling : (i) try (ii) catch (iii) throw (iv) finally
7) Write a program to input name and age of a person and throws an user define exception if entered age is negative.
8) What is thread priority ? How thread priority are set and changed ? Explain with example.(2 times)
9) Write a program to input name and age of person and throws user defined exception, if entered age is negative .
10) Write a program to create two threads ; one to print numbers in original order and other to reverse order from 1 to 50.
11) What are different types of error ? What is use of throw, throws and finally statement?
12) Write a program to accept password from user and throw ‘Authentication failure’ exception if password is incorrect.