

Unit III: Exception Handling and Multithreading

Exception Handling Basics

- Exception in Java is an indication of some unusual event. Usually it indicates the error. Let us first understand the concept. In Java, exception is handled, using five keywords try, catch, throw, throws and finally.
- The Java code that you may think may produce exception is placed within the try block. Let us see one simple program in which the use of try and catch is done in order to handle the exception divide by zero.

Java Program [Exception Demo.Java]

```
class ExceptionDemo

{

public static void main(String args[])

{

try

{

int a,b;

a=5;

b=a/0;

}

catch(ArithmeticException e)

{

System.out.println("Divide by Zero\n");

}
```

```
System.out.println("...Executed catch statement");  
  
}  
  
}
```

Output

Divide by Zero

...Executed catch statement

Inside a try block as soon as the statement: $b=a/0$ gets executed then an arithmetic exception must be raised, this exception is caught by a catch block. Thus there must be a try-catch pair and catch block should be immediate follower of try statement. After execution of catch block the control must come on the next line. These are basically the exceptions thrown by java runtime systems.

Benefits of Exception Handling

Following are the benefits of exception handling -

1. Using exception the main application logic can be separated out from the code which may cause some unusual conditions.
2. When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.
3. The working code and the error handling code can be separated out due to exception handling mechanism. Using exception handling, various types of errors in the source code can be grouped together.
4. Due to exception handling mechanism, the errors can be propagated up the method call stack i.e. problems occurring at the lower level can be handled by the higher up methods.

Exception Hierarchy

The exception hierarchy is derived from the base class Throwable. The Throwable class is further divided into two classes - Exceptions and Errors.

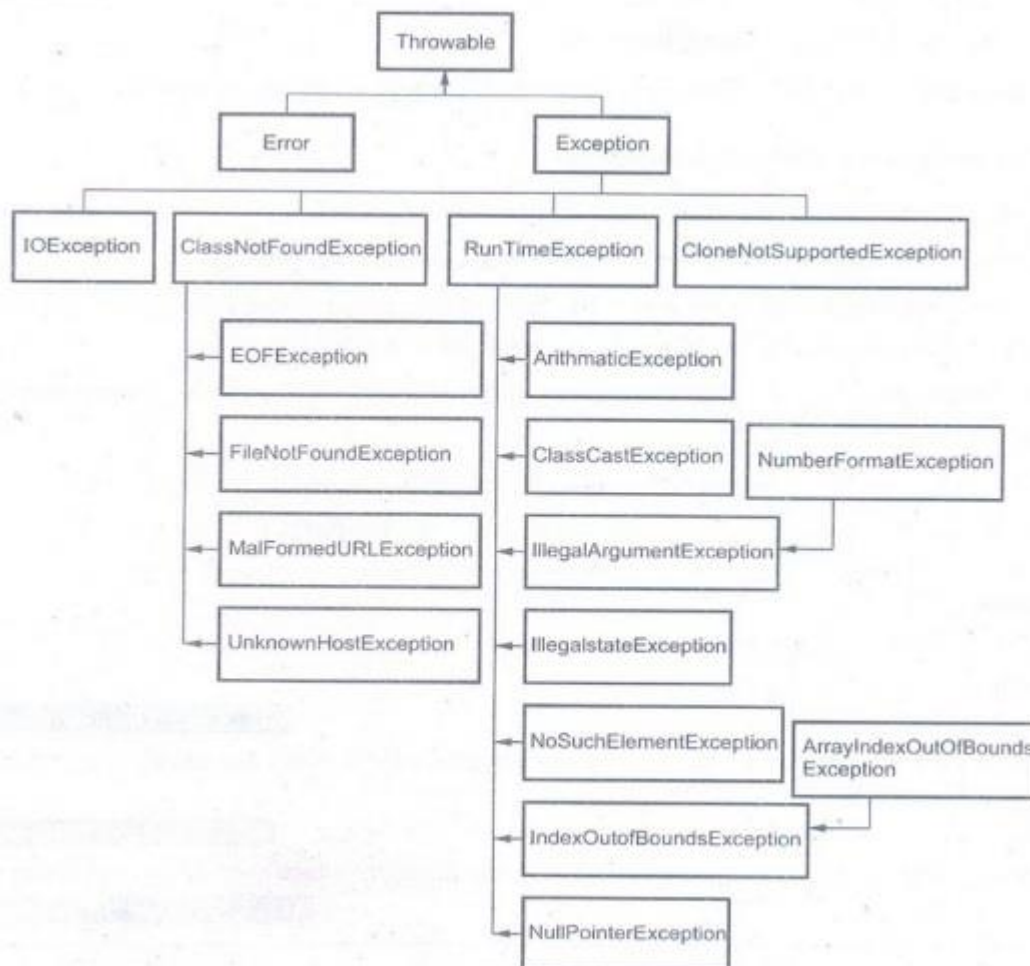


Fig. 3.1.1 Exception hierarchy

Exceptions: Exceptions are thrown if any kind of unusual condition occurs that can be caught. Sometimes it also happens that the exception could not be caught and the program may get terminated.

Errors: When any kind of serious problem occurs which could not be handled easily like OutOfMemoryError then an error is thrown.

Types of Exception

- There are two type of exceptions in Java

- **Checked Exception:** These types of exceptions need to be handled explicitly by the code itself either by using the try-catch block or by using throws. These exceptions are extended from the java.lang.Exception class.

For example: IOException which should be handled using the try-catch block.

- **Unchecked Exception:** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions. These exceptions are extended from java.lang.RuntimeException class.

For example: ArrayIndexOutOfBoundsException, NullPointerException, RuntimeException.

Keywords used in Exception Handling

- Various keywords used in handling the exception are -

try - A block of source code that is to be monitored for the exception.

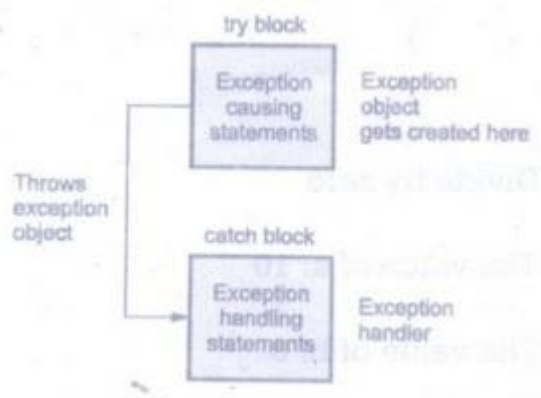
catch - The catch block handles the specific type of exception along with the try block. Note that for each corresponding try block there exists the catch block.

finally - It specifies the code that must be executed even though exception may or may not Occur.

throw - This keyword is used to throw specific exception from the program code. **throws** - It specifies the exceptions that can be thrown by a particular method.

try-catch Block

- The statements that are likely to cause an exception are enclosed within a try block. For these statements the exception is thrown.
- There is another block defined by the keyword catch which is responsible for handling the exception thrown by the try block.
- As soon as exception occurs it is handled by the catch block.



- The catch block is added immediately after the try block.
- Following is an example of try-catch block

```
try
{
//exception gets generated here
}

catch(Type_of_Exception e)
//exception is handled here
}
```

- If any one statement in the try block generates exception then remaining statements are skipped and the control is then transferred to the catch statement.

Java Program[RunErrDemo.java]

```
class RunErrDemo

{

public static void main(String[] args)

{

int a,b,c;

a=10;

b=0;

try

{

Exception occurs because the element is divided by 0.

{

c=a/b;

}

catch(ArithmeticException e)

{

System.out.println("\n Divide by zero");"

}

System.out.println("\n The value of a: "+a);

System.out.println("\n The value of b: "+b);

}

}
```

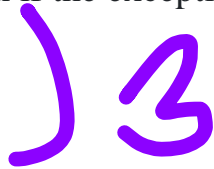
Output

Divide by zero

The value of a: 10

The value of b: 0

Note that even if the exception occurs at some point, the program does not stop at that point.



Multiple catch Clauses

- It is not possible for the try block to throw a single exception always.
- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.
- The syntax for single try and multiple catch is -

try

{

...//exception occurs

}

catch(Exception_type e)

{

...//exception is handled here

}

```
catch (Exception_type e)
{
...//exception is handled here
}

catch(Exception_type e)
{
...//exception is handled here
}
```

Example

Java Program[MultipleCatchDemo.java]

```
class MultipleCatchDemo
{
public static void main(String args[])
{
int all = new int [3];

try
{
for (int i = 1; i <=3; i++)
{
a[i] = i *i;
}

for (int i = 0; i <3; i++)
```



```
{  
a[i] = i/i;  
}  
  
}  
  
catch (ArrayIndexOutOfBoundsException e)  
{  
System.out.println ("Array index is out of bounds");  
}  
  
catch (ArithmeticException e)  
{  
System.out.println ("Divide by zero error");  
}  
  
}  
  
}
```

Output

Array index is out of bounds

Note: *If we comment the first for loop in the try block and then execute the above code we will get following output*

Divide by zero error

Nested try Statements

- When there are chances of occurring multiple exceptions of different types by the same set of statements then such situation can be handled using the nested try statements.
- Following is an example of nested try-catch statements -

Java Program[NestedtryDemo.java]

```
class NestedtryDemo

{

public static void main(String[] args)

{

try

{

int a = Integer.parseInt (args [0]);

int b = Integer.parseInt (args [1]);

int ans = 0;

try

{

ans = a/b;

System.out.println("The result is "+ans);

}

catch (ArithmeticException e)

{

System.out.println("Divide by zero");
```

```
}  
  
}  
  
catch (NumberFormatException e)  
  
{  
  
System.out.println ("Incorrect type of data");  
  
}  
  
}  
  
}
```

Output

D:\>javac NestedtryDemo.java

D:\>java NestedtryDemo 20 10

The result is 2

D:\>java NestedtryDemo 20 a

Incorrect type of data

D:\>java NestedtryDemo 20 0

Divide by zero

Using finally

- Sometimes because of execution of try block the execution gets break off. And due to this some important code (which comes after throwing off an exception) may not get executed. That means, sometimes try block: may bring some unwanted things to happen.
- The finally block provides the assurance of execution of some important code that must be executed after the try block.

- Even though there is any exception in the try block the statements assured by finally block are sure to execute. These statements are sometimes called as clean up code. The syntax of finally block is

```
finally {  
  
    //clean up code that has to be executed finally  
  
}
```

The finally block always executes. The finally block is to free the resources.

Java Program [finallyDemo.java]

```
/*  
  
This is a java program which shows the use of finally block for handling  
exception */  
  
class finallyDemo  
  
{  
  
    public static void main(String args[])  
  
    {  
  
        int a 10,b=-1;  
  
        try  
  
        {  
  
            b=a/0;  
  
        }  
  
        catch(ArithmeticException e)  
  
        {  
  
            System.out.println("In catch block: "+e);  
  
        }  
  
    }  
  
}
```

```
}  
  
finally  
  
{  
  
if(b!= -1)  
  
System.out.println("Finally block executes without occurrence of exception");  
  
else  
  
System.out.println("Finally block executes on occurrence of exception");  
  
}  
  
}  
  
}
```

Output

In catch block: java.lang.ArithmeticException: /by zero

Finally block executes on occurrence of exception

Program Explanation

- In above program, on occurrence of exception in try block the control goes to catch block, the exception of instance ArithmeticException gets caught. This is divide by zero exception and therefore /by zero will be printed as output.

Following are the rules for using try, catch and finally block

1. There should be some preceding try block for catch or finally block. Only catch block or only finally block without preceding try block is not at all possible.
2. There can be zero or more catch blocks for each try block but there must be single finally block present at the end.



Using throws

When a method wants to throw an exception then keyword throws is used. The syntax is

```
method_name(parameter_list) throws exception_list
```

```
{
```

```
}
```

- Let us understand this exception handling mechanism with the help of simple Java program.

Java Program

```
/* This programs shows the exception handling mechanism using throws
```

```
*/
```

```
class ExceptionThrows
```

```
{
```

```
static void fun(int a,int b) throws ArithmeticException
```

```
{
```

```
int c;
```

```
try
```

```
{
```

```
c=a/b;
```

```
}
```

```
catch(ArithmeticException e)
```

```
{
```

```
System.out.println("Caught exception: "+e);
```

```

    }

    }

    public static void main(String args[])

    {

    int a=5;

    fun(a,0);

    }

    }

```

Output

Caught exception: java.lang.ArithmeticException: /by zero

- In above program the method fun is for handling the exception divide by zero. This is an arithmetic exception hence we write

```
static void fun(int a,int b) throws ArithmeticException
```

- This method should be of static type. Also note as this method is responsible for handling the exception the try-catch block should be within fun.

Using throw

- For explicitly throwing the exception, the keyword throw is used. The keyword throw is normally used within a method. We can not throw multiple exceptions using throw.

Java Programming Example

```
class Exception Throw
```

```
{
```

```
static void fun(int a,int b)

{

int c;

if(b==0)

throw new ArithmeticException("Divide By Zero!!!");

else

c=a/b;

}

public static void main(String args[])

{

int a=5;

fun(a,0);

}

}
```

Output

Exception in thread "main" java.lang.ArithmeticException: Divide By Zero!!!

at ExceptionThrow.fun(ExceptionThrow.java:7)

at ExceptionThrow.main(ExceptionThrow.java:14)



Difference between throw and throws

Throw	Throws
For explicitly throwing the exception, the keyword throw is used.	For declaring the exception the keyword throws is used.
Throw is followed by instance.	Throws is followed by exception class.
Throw is used within the method.	Throw is used with method signature.
We cannot throw multiple exceptions.	It is possible to declare multiple exceptions using throws.
Example - Refer section 3.7.	Example - Refer section 3.6.

Java's Built-in Exceptions: • Various common exception types and causes are enlisted in the following table-

Exception	Description
ArithmeticException	This is caused by error in Math operations. For e.g. Divide by zero.
NullPointerException	Caused when an attempt to access an object with a Null reference is made.
IOException	When an illegal input/output operation is performed then this exception is raised.
IndexOutOfBoundsException	An index when gets out of bound ,this exception will be caused.
ArrayIndexOutOfBoundsException	Array index when gets out of bound, this exception will be caused.
ArrayStoreException	When a wrong object is stored in an array this exception must occur.
EmptyStackException	An attempt to pop the element from empty stack is made then this exception occurs
NumberFormatException	When we try to convert an invalid string to number this exception gets caused.
RuntimeException	To show general run time error this exception must be raised.
Exception	This is the most general type of exception
ClassCastException	This type of exception indicates that one tries to cast an object to a type to which the object can not be casted.
IllegalStateException	This exception shows that a method has been invoked at an illegal or inappropriate time. That means when Java environment or Java application is not in an appropriate state then the method is invoked.

User defined Exception

- We can throw our own exceptions using the keyword throw.
- The syntax for throwing out own exception is

throw new Throwable's subclass

- Here the Throwable's subclass is actually a subclass derived from the Exception class.

For example -

```
throw new ArithmeticException();
```

Throwable's subclass

- Let us see a simple Java program which illustrates this concept.

Java Program[MyExceptDemo.java]

```
import java.lang.Exception;

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

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

```
{  
  
int age;  
  
age=15;  
  
try  
  
{  
  
if(age<21)  
  
throw new MyOwnException("Your age is very less than the condition");  
  
}  
  
catch (MyOwnException e)  
  
{  
  
System.out.println ("This is My Exception block");  
  
System.out.println (e.getMessage());  
  
}  
  
finally  
  
{  
  
System.out.println ("Finally block:End of the program");  
  
}  
  
}  
  
}
```

Output

This is My Exception block

Your age is very less than the condition

Finally block:End of the program

Program Explanation

- In above code, the age value is 15 and in the try block exception if the value is less than 21. As soon as the exception is thrown the catch block gets executed. Hence as an output we get the first message "This is My Exception block". Then the control is transferred to the class MyOwnException(defined at the top of the program). The message is set and it is "Your age is very less than the condition". This message can then printed by the catch block using the System.out.println statement by means of e.message.
- At the end the finally block gets executed.

Ex. 3.9.1: Write an exception class for a time of day that can accept only 24 hour representation of clock hours. Write a java program to input various formats of timings and throw suitable error messages.

Sol.:

```
import java.lang.Exception;

import java.io.*;

import java.util.*;

class MyException extends Exception

{

    MyException(String msg)

    {

        super(msg);

    }

}

class Clock
```

```
{  
  
private int hour;  
  
private int minute;  
  
public void input() throws IOException  
{  
  
    BufferedReader br=new BufferedReader (new InputStreamReader(System.in));  
  
    System.out.println("\n Enter the time in hh:mm format");  
  
    String str=br.readLine();  
  
    StringTokenizer tokn=new StringTokenizer(str,":");  
  
    String h=tokn.nextToken();  
  
    String m=tokn.nextToken();  
  
    hour=Integer.parseInt(h);  
  
    minute= Integer.parseInt(m);  
  
    try  
    {  
  
        System.out.println("Hour: "+hour);  
  
        if((hour < 0) || (hour>24))  
  
            throw new MyException("Fatal error: invalid hour");  
  
    }  
  
    catch(MyException e)  
    {  
  
        System.out.println(e.getMessage());  

```

```
}

try

{

System.out.println("Minute: "+minute);

if((minute <0) || (minute > 59))

throw new MyException("Fatal error: invalid minute");

}

catch(MyException e)

{

System.out.println(e.getMessage());

}

}

}

class ClockDemo

{

public static void main(String[] args) throws IOException

{

Clock c=new Clock();

c.input();

}

}
```

Output(Run1)

Enter the time in hh:mm format

25:80

Hour: 25

Fatal error: invalid hour

Minute: 80

Fatal error: invalid minute

Output(Run2)

Enter the time in hh:mm format

10:70

Hour: 10

Minute: 70

Fatal error: invalid minute

Output(Run3)

Enter the time in hh:mm format

30:40

Hour: 30

Fatal error: invalid hour

Minute: 40

Review Question

1. With suitable Java program, explain user defined exception handling.

Java Thread Model

Thread life cycle specifies how a thread gets processed in the Java program. By executing various methods. Following figure represents how a particular thread can be in one of the state at any time.

Java Thread Model

Thread life cycle specifies how a thread gets processed in the Java program. By executing various methods. Following figure represents how a particular thread can be in one of the state at any time.

Method	Purpose
start()	The thread can be started and invokes the run method.
run()	Once thread is started it executes in run method.
setName()	We can give the name to a thread using this method.
getName()	The name of the thread can be obtained using this name.
join()	This method waits for thread to end.

New state

- When a thread starts its life cycle it enters in the new state or a create state.

Runnable state

- This is a state in which a thread starts executing.
- Waiting state
- Sometimes one thread has to undergo in waiting state because another thread starts executing.

Timed waiting state

- There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state.

Blocked state

- When a particular thread issues an Input/Output request then operating system sends it in blocked state until the I/O operation gets completed. After the I/O completion the thread is sent back to the runnable state.

Terminated state

After successful completion of the execution the thread in runnable state enters the terminated state.

Review Questions

- 1.What is thread? Explain its state and methods.
- 2.Write about various threads states in Java.
- 3.Explain: States of a thread with a neat diagram.
- 4.Define threads. Describe in detail about thread life cycle.
5. Explain in detail the different states of a thread.

Creating a Thread

In Java we can implement the thread programs using two approaches -

- o Using Thread class
- o sing runnable interface.

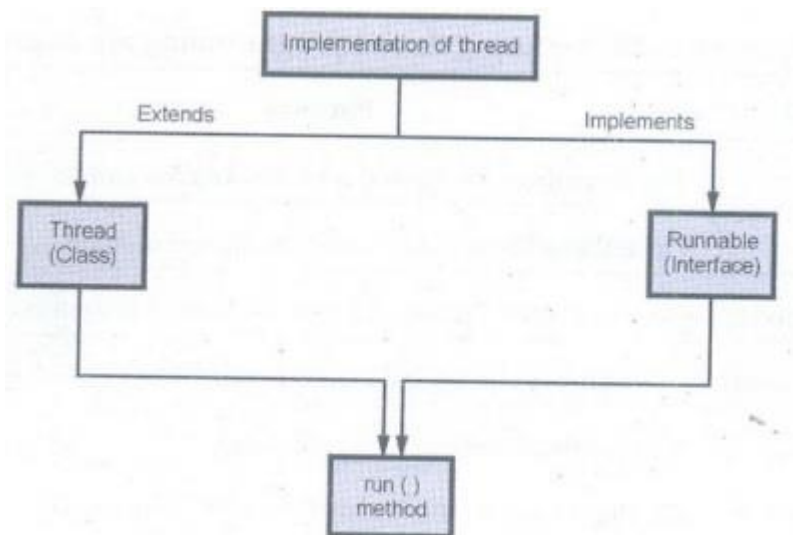


Fig. 3.12.1 : Creation of Java thread

As given in Fig. 3.12.1, there are two methods by which we can write the Java thread programs one is by extending thread class and the other is by implementing the Runnable interface.

- The run() method is the most important method in any threading program. By using this method the thread's behaviour can be implemented. The run method can be written as follows -

```
public void run()
{
    //statements for implementing thread
}
```

For invoking the thread's run method the object of a thread is required. This object can be obtained by creating and initiating a thread using the start() method.

Extending Thread Class

- The Thread class can be used to create a thread.
- Using the extend keyword your class extends the Thread class for creation of thread. For example if I have a class named A then it can be written as

class A extends Thread

- Constructor of Thread Class: Following are various syntaxes used for writing the constructor of Thread Class.

Thread()

Thread(String s)

Thread (Runnable obj)

Thread(Runnable obj, String s);

- Various commonly used methods during thread programming are as given below -

Method	Purpose
start()	The thread can be started and invokes the run method.
run()	Once thread is started it executes in run method.
setName()	We can give the name to a thread using this method.
getName()	The name of the thread can be obtained using this name.
join()	This method waits for thread to end.

Following program shows how to create a single thread by extending Thread Class,

Java Program

class MyThread extends Thread doin

{

```
public void run()

{

System.out.println("Thread is created!!!");

}

}

class ThreadProg

{

public static void main(String args[])

{

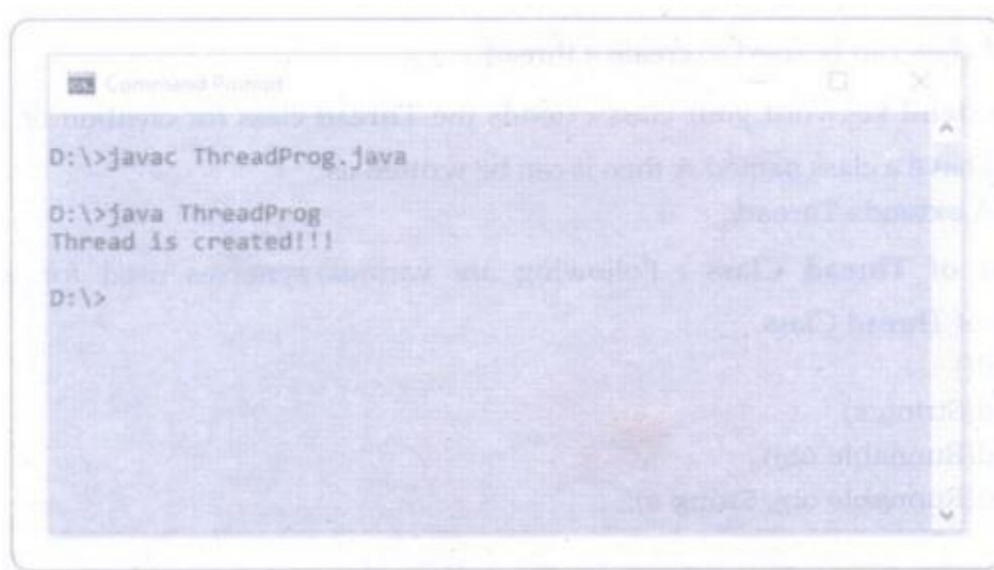
MyThread t=new MyThread();

t.start();

}

}
```

Output



Program Explanation :

In above program, we have created two classes. One class named MyThread extends the Thread class. In this class the run method is defined. This run method is called by t.start() in main() method of class ThreadProg.

The thread gets created and executes by displaying the message Thread is created.

Ex. 3.12.1: Create a thread by extending the Thread Class. Also make use of constructor and display message "You are Welcome to Thread Programming".

Sol.:

```
class MyThread extends Thread
{
String str=""; //data member of class MyThread

MyThread(String s)//constructor
{
this.str=s;
}

public void run()
{
System.out.println(str);
}
}

class ThreadProg
{
```

```
public static void main(String args[])
{
    MyThread t=new MyThread("You are Welcome to Thread Programming");
    t.start();
}
}
```

Output

You are Welcome to Thread Programming

Implementing Runnable Interface

- The thread can also be created using Runnable interface.
- Implementing thread program using Runnable interface is preferable than implementing

it by extending the thread class because of the following two reasons -

1. If a class extends a thread class then it can not extends any other class which may be required to extend.
2. If a class thread is extended then all its functionalities get inherited. This is an expensive operation.se

- Following Java program shows how to implement Runnable interface for creating a single thread.

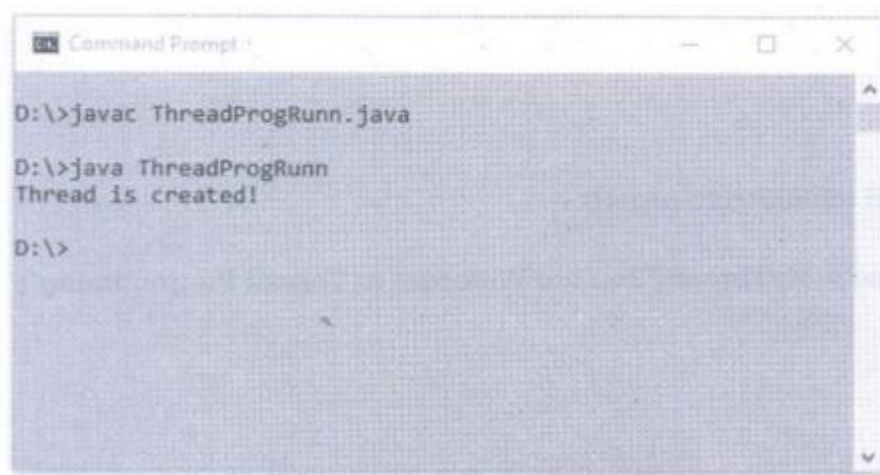
Java Program

class MyThread implements Runnable

```
{
    public void run()
```

```
{  
  
System.out.println("Thread is created!");  
  
}  
  
}  
  
class ThreadProg Runn  
  
{  
  
public static void main(String args[])  
  
{  
  
MyThread obj = new MyThread();  
  
Thread t=new Thread(obj);  
  
t.start();  
  
}  
  
}
```

Output



```
Command Prompt  
D:\>javac ThreadProgRunn.java  
D:\>java ThreadProgRunn  
Thread is created!  
D:\>
```

Program Explanation :

- In above program, we have used interface Runnable.

- While using the interface, it is necessary to use implements keyword.
- Inside the main method
 1. Create the instance of class MyClass.
 2. This instance is passed as a parameter to Thread class.
 3. Using the instance of class Thread invoke the start method.
 4. The start method in-turn calls the run method written in MyClass.
- The run method executes and display the message for thread creation.

Ex. 3.12.2 Create a thread by extending the Thread Class. Also make use of constructor and display message "You are Welcome to Thread Programming".

Sol. :

```
class MyThread implements Runnable
```

```
{
```

```
String str;
```

```
MyThread(String s)
```

```
{
```

```
this.str=s;
```

```
}
```

```
public void run()
```

```
{
```

```
System.out.println(str);
```

```
}
```

```
}
```



```

class ThreadProgRunn
{
public static void main(String args[])
{
MyThread obj = new MyThread("You are Welcome to Thread Programming");
Thread t=new Thread(obj);
t.start();
}
}

```

Output

You are Welcome to Thread Programming

Ex. 3.12.3: Write a Java program that prints the numbers from 1 to 10 line by line after every 10 seconds.

Sol. :

Java Program[MultiThNum.java]

```

class NumPrint extends Thread
{
int num;

NumPrint()

{
start();//directs to the run method
}

```

```
public void run()//thread execution starts

{

try

{

for(int i=1;i<=10;i++)

{

System.out.println(i);

Thread.sleep(10000);//10 sec,b'coz 1000millisec=1 sec.

}

}

catch (InterruptedException e)

{

System.out.println("Exception in Thread handling");

}

}

}

public class MultiThNum

{

public static void main(String args[])

{

NumPrint t1;

t1=new NumPrint();
```

```
}
```

```
}
```

Output

```
C:>javac MultiThNum.java
```

```
C:>java MultiThNum
```

1

2

3

4

5

6

7

7

8

9

10

Review Questions

1. Explain how threads are created in Java.
2. How to extend the thread class? Give an example.
3. How to implement runnable interface for creating and starting threads?
4. What is meant by concurrent programming? Define thread. Discuss the two ways of implementing thread using example.



Multithreading

The multiple threads can be created both by extending Thread class and by implementing the Runnable interface.

1. Java Program for creating multiple threads by extending Thread Class

```
class A extends Thread
```

```
{
```

```
public void run()
```

```
{
```

```
for(int i=0;i<=5;i++)//printing 0 to 5
```

```
{
```

```
System.out.println(i);
```

```
}
```

```
}
```

```
}
```

```
class B extends Thread
```

```
{
```

```
public void run()
```

```
{
```

```
for(int i=10;i>=5;i--)//printing 10 to 5
```

```
{
```

```
System.out.println(i);
```

```
}
```

```
}  
  
}  
  
class ThreadProg  
  
{  
  
public static void main(String args[])  
  
{  
  
A t1=new A();  
  
B t2=new B();  
  
t1.start();  
  
t2.start();  
  
}  
  
}
```

Output

0

1

2

3

4

5

10

9

8

7

6

5

2. Java Program for creating multiple threads by implementing the Runnable interface

class A implements Runnable

{

public void run()

{

for(int i=0;i<=5;i++)//printing 0 to 5

{

System.out.println(i);

}

}

}

class B implements Runnable

{

public void run()

{

for(int i=10;i>=5;i--)//printing 10 to 5

{

System.out.println(i);

```
}  
  
}  
  
}  
  
class ThreadProg Runn  
{  
  
public static void main(String args[])  
{  
  
A obj1=new A();  
  
B obj2=new B();  
  
Thread t1=new Thread(obj1);  
  
Thread t2=new Thread(obj2);  
  
t1.start();  
  
t2.start();  
  
}  
  
}
```

Output

0

1

2

3

4

5

10

9

8

7

6

5.

Ex. 3.13.1: Write a Java application program for generating four threads to perform the following operations - i) Getting N numbers as input ii) Printing the numbers divisible by five iii) Printing prime numbers iv) Computing the average.

Sol. :

```
import java.io.*;

import java.util.*;

class FirstThread extends Thread

{

public void run() //generating N numbers

{

int i;

System.out.println("\nGenerating Numbers: ");

for (i=1;i<=10;i++)

{

System.out.println(i);

}
```



```
}
```

```
}
```

```
class SecondThread extends Thread
```

```
{
```

```
public void run() //Displaying the numbers divisible by five
```

```
{
```

```
int i;
```

```
System.out.println("\nDivisible by Five: ");
```

```
for (i=1;i<=10;i++) //10 can be replaced by any desired value
```

```
{
```

```
if (i%5==0)
```

```
System.out.println(i);
```

```
}
```

```
}
```

```
}
```

```
class ThirdThread extends Thread
```

```
{
```

```
public void run() //generating the prime numbers
```

```
{
```

```
int i;
```

```
System.out.println("\n Prime Numbers: ");
```

```
for (i=1;i<=10;i++) //10 can be replaced by any desired value
```

```
{  
  
int j;  
  
for (j=2; j<i; j++)  
  
{  
  
int n = i%j;  
  
if (n==0)  
  
break;  
  
}  
  
if(i == j)  
  
System.out.println(i);  
  
}  
  
}  
  
}  
  
class FourthThread extends Thread  
  
{  
  
public void run() //generating the prime numbers  
  
{  
  
int i,sum;  
  
double avg;  
  
sum=0;  
  
System.out.println("\nComputing Average: ");  
  
for (i=1;i<=10;i++) //10 can be replaced by any desired value
```

```
sum=sum+i;

avg=sum/(i-1);

System.out.println(avg);

}

}

class MainThread

{

public static void main(String[] args) throws IOException

{

FirstThread T1 = new FirstThread(); //creating first thread

SecondThread T2 = new SecondThread(); //creating second thread

ThirdThread T3 = new ThirdThread(); //creating Third thread

FourthThread T4 = new FourthThread(); //creating Fourth thread

T1.start(); //First Thread starts executing

T2.start();//Second Thread starts executing

T3.start();//Third Thread starts executing

T4.start();//Fourth Thread starts executing

}

}
```

Output

Generating Numbers:

1

2

3

4

5

6

7

8

9

10

Divisible by Five:

5

10

Computing Average:

5.0

Prime Numbers:

2

3

5

7

Ex. 3.13.2: Create a Bank Database application program to illustrate the use of multithreads

Sol. :

```
public class BankAppl implements Runnable {  
  
    private Account acc = new Account();  
  
    public static void main(String[] args) {  
  
        BankAppl obj = new BankAppl();  
  
        Thread t1 = new Thread(obj);  
  
        Thread t2 = new Thread(obj);  
  
        Thread t3= new Thread(obj);  
  
        t1.setName("Mr.XYZ");  
  
        t2.setName("Mr.ABC");  
  
        t3.setName("Mr.POR");  
  
        t1.start();  
  
        t2.start()  
  
        t3.start();  
  
    }  
  
    public void run() {  
  
        for (int x = 0; x < 10; x++) {  
  
            makeWithdrawal(10);  
  
            if (acc.getBalance() < 0) {  
  
                System.out.println("Account Overdrawn!!!");  
  
            }  
  
        }  
  
    }  
  
}
```

```
void makeWithdrawal(int amt) {

    int bal;

    bal = acc.getBalance();

    if (bal >= amt) {

        System.out.println("\t"+Thread.currentThread().getName()+"withdraws
        Rs."+amt);

        try {

            Thread.sleep(100);

        } catch (InterruptedException ex) {

        }

        acc.withdraw(amt);

        bal = acc.getBalance();

        System.out.println("\t. The Balance: "+bal);

    } else {

        System.out.println("Insufficient Balance in account for " +
        Thread.currentThread().getName() + " to withdraw " + acc.getBalance());

    }

}

}

}

class Account {

    private int balance = 100;

    public int getBalance() {

        return balance;

    }

}
```

```
}  
  
public void withdraw(int amount) {  
  
    balance = balance - amount;  
  
}  
  
}
```

Review Questions

1. What is multithreading? What are the methods available in Java for inter-thread communication? Discuss with example
2. Write a Java program to illustrate multithreaded programming.
3. Write notes on multi-threaded programming.
4. Write a java application that illustrate the use of multithreading. Explain the same with sample input.
5. Describe the creation of a single thread and multiple threads using an example
6. Create a simple real life application program in Java to illustrate the use of multithreads.

Priorities

- In Java threads scheduler selects the threads using their priorities.
- The thread priority is a simple integer value that can be assigned to the particular thread.
- These priorities can range from 1 (lowest priority) to 10 (highest priority).
- There are two commonly used functionalities in thread scheduling -
 - o `setPriority`
 - o `getPriority`
- The function `setPriority` is used to set the priority to each thread

Thread_Name.setPriority (priority_val);

Where, `priority_val` is a constant value denoting the priority for the thread. It is defined as follows

1. `MAX_PRIORITY = 10`
2. `MIN_PRIORITY = 1`
3. `NORM_PRIORITY = 5`

- The function `getPriority` is used to get the priority of the thread.

Thread_Name.getPriority();

What is Preemption?

- Preemption is a situation in which when the currently executed thread is suspended temporarily by the highest priority thread.
- The highest priority thread always preempts the lowest priority thread.

Let us see the illustration of thread prioritizing with the help of following example

Java program[Thread_PR_Prog.java]

```
class A extends Thread

{

public void run()

{

System.out.println("Thread #1");

for(int i=1;i<=5;i++)

{

System.out.println("\tA: "+i)

}

System.out.println("\n-----End of Thread #1-----");

}

}

class B extends Thread

{

public void run()

{

System.out.println("Thread #2");

for(int k=1;k<=5;k++)

{

System.out.println("\tB: "+k);

}

}
```

```
System.out.println("\n-----End of Thread #2-----");
```

```
}
```

```
}
```

```
class Thread_PR_Prog
```

```
{
```

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

```
{
```

```
A obj1=new A();
```

```
B obj2=new B();
```

```
obj1.setPriority(1);
```

```
obj2.setPriority(10);//highest priority
```

```
System.out.println("Strating Thread#1");
```

```
obj1.start();
```

```
System.out.println("Strating Thread#2");
```

```
obj2.start();
```

```
}
```

```
}
```

Output

Strating Thread #1

Strating Thread#2

Thread #1

Thread #2

B: 1

B: 2

B: 3

B: 4

B: 5

Thread 2 preempts the execution of thread 1

-----End of Thread #2-----

A: 1

A: 2

A: 3

A: 4

A: 5

-----End of Thread #1-----

Program explanation

In above program,

- 1) We have created two threads - the Thread#1 and Thread#2.
- 2) We assign highest priority to Thread#2.
- 3) In the main function both the threads are started but as the Thread #2 has higher priority than the Thread#1, the Thread #2 preempts the execution of Thread#1.
- 4) Thus Thread #2 completes its execution and then Thread #1 completes.

Synchronization

- When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization. The synchronization is the concept which is based on monitor. Monitor is used as mutually exclusive lock or mutex. When a thread owns this monitor at a time then the other threads can not access the resources. Other threads have to be there in waiting state.
- In Java every object has implicit monitor associated with it. For entering in object's monitor, the method is associated with a keyword synchronized. When a particular method is in synchronized state then all other threads have to be there in waiting state.
- There are two ways to achieve the synchronization -

1. Using Synchronized Methods 2. Using Synchronized Blocks (Statements).

Let us make the method synchronized to achieve the synchronization by using following Java program -

1. Using Synchronized Method

```
class Test
{
    synchronized void display(int num)
    {
        System.out.println("\nTable for "+num);
    }
    System.out.print(" "+num*i);
}
System.out.print("\nEnd of Table");
try
```

any 1 program as example

```
{  
    Thread.sleep(1000);  
} catch (Exception e) {}  
}  
}  
  
class A extends Thread  
{  
    Test th1;  
    A(Test t)  
    {  
        th1=t;  
    }  
    public void run()  
    {  
        th1.display(2);  
    }  
}  
  
class B extends Thread  
{  
    Test th2;  
    B(Test t)  
    {
```

```
th2=t;

}

public void run()

{

th2.display(100);

}

}

class MySynThread

{

public static void main(String args[])

{

Test obj=new Test();

A t1=new A(obj);

B t2=new B(obj);

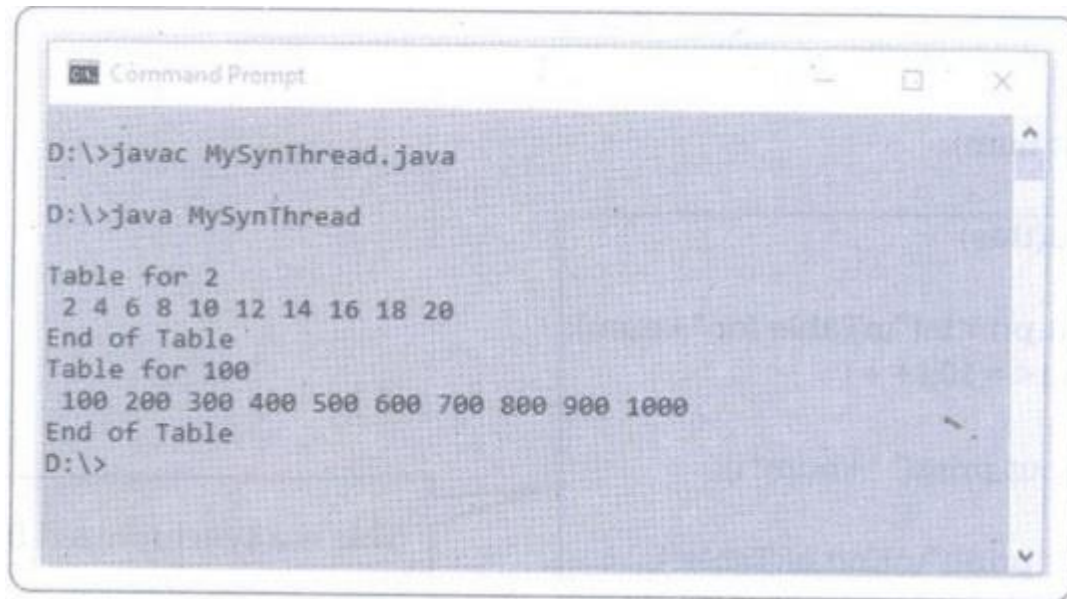
t1.start();

t2.start();

}

}
```

Output



```
Command Prompt
D:\>javac MySynThread.java
D:\>java MySynThread
Table for 2
 2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

Program Explanation:

- In above program we have written one class named Test. Inside this class the synchronized method named display is written. This method displays the table of numbers.
- We have written two more classes named A and B for executing the thread. The constructors for class A and Class B are written and to initialize the instance variables of class Test as th1 (as a variable for class A) and th2 (as a variable for class B)
- Inside the run methods of these classes we have passed number 2 and 10 respectively and display method is invoked.
- The display method executes firstly for thread t1 and then for t2 in synchronized manner.

2. Using Synchronized Block

- When we want to achieve synchronization using the synchronized block then create a block of code and mark it as synchronized.
- Synchronized statements must specify the object that provides the intrinsic lock.

Syntax

The syntax for using the synchronized block is

```
synchronized(object reference)
{
    statement;
    statement; //block of code to be synchronized
    .
    .
    .
}
```

Java Program

```
class Test
{
    void display(int num)
    {
        Synchronized(this)
        {
            System.out.println("\n Table for"+num);
            for( int i=1;i<=10;i++)
            {
                System.out.println(" "+num*i);
            }
            System.out.println("\n End of Table");
        }
    }
}
```



```
try
{
Thread.sleep (1000);
} catch(Exception e){ }
}
```

```
class A extends Thread
```

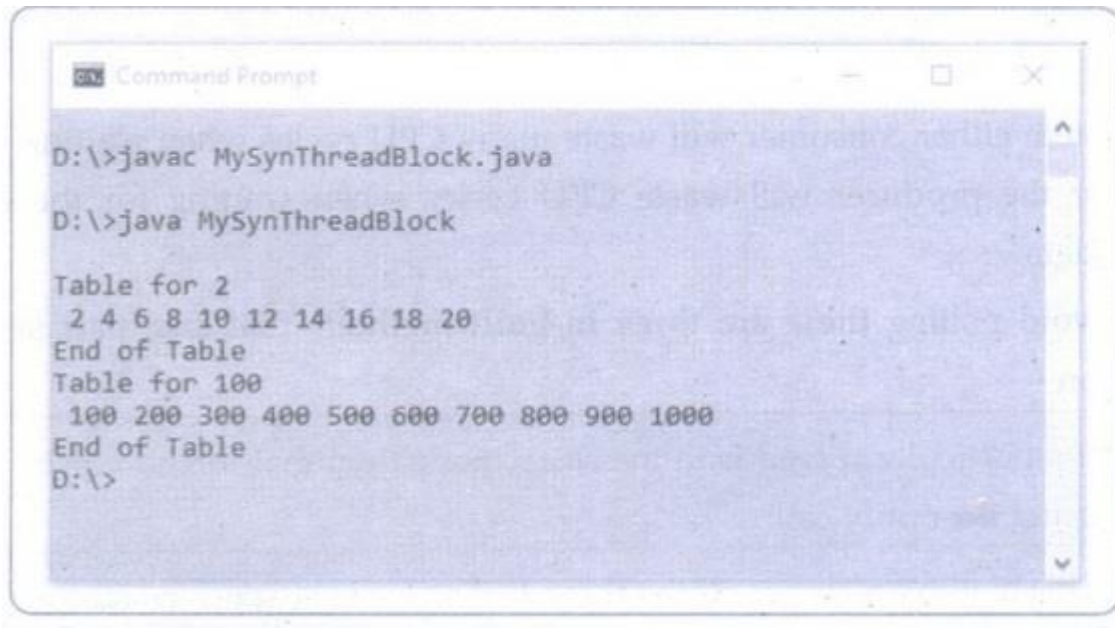
```
{
Test th1;
A(Test t)
{
th1=t;
}
public void run()
{
th1.display(2);
}
}
```

```
class B extends Thread
```

```
{
Test th2;
B(Test t)
```

```
{  
th2=t;  
}  
  
public void run()  
{  
th2.display(100);  
}  
}  
  
class MySynThreadBlock  
{  
public static void main(String args[])  
{  
Test obj=new Test();  
A t1=new A(obj);  
B t2=new B(obj);  
t1.start();  
t2.start();  
}  
}
```

Output



```
D:\>javac MySynThreadBlock.java
D:\>java MySynThreadBlock

Table for 2
 2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

Points to Remember about Synchronization

1. Only methods can be synchronized but the variables and classes can not be synchronized.
2. Each object has one lock.
3. A class contains several methods and all methods need not be synchronized.
4. If two threads in a class wants to execute synchronized methods and both the methods are using the same instance of a class then only one thread can access the synchronized method at a time.
5. We can not synchronize the constructors
6. A thread can acquire more than one lock.

Review Questions

- 1.What is thread synchronization ? Discuss with an example.
2. What is synchronization? Explain the different types of synchronization in java.

Inter Thread Communication

- Two or more threads communicate with each other by exchanging the messages. This mechanism is called interthread communication.
- Polling is a mechanism generally implemented in a loop in which certain condition is repeatedly checked.
- To better understand the concept of polling, consider producer-consumer problem in which producer thread produces and the consumer thread consumes whatever is produced.
- Both must work in co-ordination to avoid wastage of CPU cycles.
- But there are situations in which the producer has to wait for the consumer to finish consuming of data. Similarly the consumer may need to wait for the producer to produce the data.
- In Polling system either consumer will waste many CPU cycles when waiting for producer to produce or the producer will waste CPU cycles when waiting for the consumer to consume the data.
- In order to avoid polling there are three in-built methods that take part in inter-thread communication -

notify()	If a particular thread is in the sleep mode then that thread can be resumed using the notify call.
notifyall()	This method resumes all the threads that are in suspended state.
wait()	The calling thread can be send into a sleep mode.

Example Program

Following is a simple Java program in which two threads are created one for producer and

another is for consumer.

The producer thread produces(writes) the numbers from 0 to 9 and the consumer thread consumes(reads) these numbers.

The wait and notify methods are used to send particular thread to sleep or to resume the thread from sleep mode respectively.

Java Program[InterThread.java]

```
class MyClass

{

int val;

boolean flag = false;

synchronized int get() //by toggling the flag synchronised read and write is
performed

{

if(!flag)

try

{

wait();

}

catch (InterruptedException e)

{

System.out.println("InterruptedException!!!");

}

System.out.println("Consumer consuming: " + val);

flag = false;

notify();

return val;
```

```
}

synchronized void put(int val)

{

if(flag)

try

{

wait();

}

catch (InterruptedException e)

{

System.out.println("InterruptedException!!!");

}

this.val = val;

flag = true;

System.out.println("Producer producing " + val);

notify();

}

}

class Producer extends Thread

{

MyClass th1;

Producer(MyClass t).
```

```
{  
  
th1 = t;  
  
}  
  
public void run()  
  
{  
  
for(int i =0;i<10;i++)  
  
{  
  
th1.put(i);  
  
}  
  
}  
  
}
```

Class Consumer Extends Thread

```
{  
  
MyClass th2;  
  
Consumer(MyClass t)  
  
{  
  
th2 = t;  
  
}  
  
public void run()  
  
{  
  
for(int I = 0;i<10;i++)  
  
{
```

```
th2.get();  
  
}  
  
}  
  
}  
  
class InterThread  
  
{  
  
public static void main(String[] arg)  
  
{  
  
MyClass TObj = new MyClass();  
  
Producer pthread=new Producer(TObj);  
  
Consumer cthread=new Consumer(TObj);  
  
pthread.start();  
  
cthread.start();  
  
}  
  
}
```

Output

Producer producing 0

Consumer consuming: 0

Producer producing 1

Consumer consuming: 1

Producer producing 2

Consumer consuming: 2

Producer producing 3

Consumer consuming: 3

Producer producing 4

Consumer consuming: 4

Producer producing 5

Consumer consuming: 5

Producer producing 6

Consumer consuming: 6

Producer producing 7

Consumer consuming: 7

Producer producing 8

Consumer consuming: 8

Producer producing 9

Consumer consuming: 9

Program Explanation

- **Inside get() -**

The wait() is called in order to suspend the execution by that time the producer writes the value and when the data gets ready it notifies other thread that the data is now ready. Similarly when the consumer reads the data execution inside get() is suspended. After the data has been obtained, get() calls notify(). This tells producer that now the producer can write the next data in the queue.

- **Inside put() -**

The wait() suspends execution by that time the consumer removes the item from the queue. When execution resumes, the next item of data is put in the queue,

and notify() is called. When the notify is issued the consumer can now remove the corresponding item for reading.

Suspending - Resuming, and Stopping Threads

‘• **Stopping a thread:** A thread can be stopped from running further by issuing the following statement -

th.stop();

By this statement the thread enters in a dead state. From stopping state a thread can never return to a runnable state.

‘• **Blocking a thread:** A thread can be temporarily stopped from running. This is called blocking or suspending of a thread. Following are the ways by which thread can be blocked

1. sleep()

By sleep method a thread can be blocked for some specific time. When the specified time gets elapsed then the thread can return to a runnable state.

2. suspend

By suspend method the thread can be blocked until further request comes. When the resume() method is invoked then the thread returns to a runnable state.

3. wait

The thread can be made suspended for some specific conditions. When the notify method is called then the blocked thread returns to the runnable state.

• **The difference between the suspending and stopping thread** is that if a thread is suspended then its execution is stopped temporarily and it can return to a runnable state. But in case, if a thread is stopped then it goes to a dead state and can never return to runnable state.

Resuming a thread: The resume() method is only used with suspend() method. This method is only to resume a thread which was suspended using suspend() method. The syntax is -

```
public final resume()
```

Ex. 3.17.1 Write a java program for inventory problem to illustrates the usage of thread synchronized keyword and inter thread communication process. They have three classes called consumer, producer and stock.



Sol.:

class Consumer implements Runnable

{

Stock obj1;

Thread t;

Consumer (Stock obj1)

{

this.obj1 = obj1;

t = new Thread(this, "Consumer thread");

t.start();

}

public void run().

{

while (true)

{

try {

t.sleep(900);

```
} catch (InterruptedException e) { }  
  
obj1.getStock((int)(Math.random()*100));  
  
}  
  
}  
  
void stop()  
  
{  
  
t.stop();  
  
}  
  
}  
  
class Producer implements Runnable  
  
{  
  
Stock obj2;  
  
Thread t;  
  
Producer(Stock obj2)  
  
{  
  
this.obj2 = obj2;  
  
t = new Thread(this, "Producer thread");  
  
t.start();  
  
}  
  
public void run()  
  
{  
  
while(true)
```

```
{  
  
try {  
  
t.sleep(900);  
  
} catch (InterruptedException e) { }  
  
obj2.addStock((int)(Math.random()*100));  
  
}  
  
}  
  
void stop()  
  
{  
  
t.stop();  
  
}  
  
}  
  
class Stock  
  
{  
  
int items = 0;  
  
public synchronized void addStock(int i)  
  
{  
  
items = items + i;  
  
System.out.println("Stock added:" + i);  
  
System.out.println("present stock :" + items);  
  
notify();  
  
}
```

```
public synchronized int getStock(int j)

{
while(true)

{
if(items >= i)

{
items = items - j;

System.out.println("Item is removed from stock : " + j);

System.out.println("Current stock is : " + items);

break;

}

else

{

System.out.println("\tStock is not enough!!!" );

System.out.println ("\t Waiting for items to get added...");

try {

wait();

}catch(InterruptedException e) { }

}

}

return items;

}
```

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

{

    Stock j = new Stock();

    Producer p = new Producer(j);

    Consumer c = new Consumer(j);

    try

    {

        Thread.sleep(10000);

        p.stop();

        c.stop();

        p.t.join();

        c.t.join();

        System.out.println("Thread stopped");

    } catch (InterruptedException e) { }

    System.exit(0);

}
```

Wrappers

- Wrapper classes are those classes that allow primitive data types to be accessed as objects.
- The wrapper class is one kind of wrapper around a primitive data type. The wrapper classes represent the primitive data types in its instance of the class.
- Following table shows various primitive data types and the corresponding wrapper classes -

Primitive data type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short
void	Void

- Methods to handle wrapper classes are enlisted in the following table -

Method used	Description
Integer val=new Integer(int_var)	An object is created for the integer variable int_var.
Float val=new Float(f_var)	An object is created for the Float variable f_var.
Double val=new Double(d_var)	An object is created for the double variable d_var.
Long val=new Long(Long_var)	An object is created for the Long variable Long_var.

• Suppose an object for holding an integer value is created then we can retrieve the integer value from it using typevalue() method. For instance the object obj contains an integer value then we can obtain the integer value from obj. It is as follows -

int num=obj.intValue();

- Similarly we can use floatValue(), doubleValue() and longValue().
- Similarly in order to convert the numerical value to string the toString() method can be used. For instance

str=Integer.toString(int_val)

- The variable int_val can be converted to string str.
- For converting the string to numerical value the parseInt or parseLong methods can be used.

Points to remember about wrapper classes

1. The wrapper classes do not contain the constructors.
2. The methods of the wrapper classes are static.
3. After assigning the values to the wrapper class we cannot change them.

Example: Java Program

```
import java.io.*;

import java.lang.*;

class WrapperDemo

{

public static void main(String[] args)

{

System.out.println("Creating an object for value 10");

Integer i=new Integer(10);

System.out.println("Obtaining the value back from the object: "+i.intValue());

String str="100";

System.out.println("The string is: "+str);

System.out.println("Obtaining the numeric value from the string: "+
Integer.parseInt(str));

}

}
```

Output

Creating an object for value 10

Obtaining the value back from the object: 10

The string is: 100

Obtaining the numeric value from the string: 100

Ex. 3.18.1 Define wrapper class. Give the following wrapper class methods with

syntax and use :

1) To convert integer number to string.

2) To convert numeric string to integer number.

3) To convert object numbers to primitive numbers sing type value () method.

Sol. Wrapper class - Refer section 3.18.

```
class WrapperDemo1

{

public static void main(String args[]).

{

System.out.println("\tInteger to String Conversion");

int i=100;

String str=Integer.toString(i);

System.out.println("int Value: "+i);

System.out.println("Equivalent String: "+str);.

System.out.println("\tString to Integer Conversion");

str="500";

int j= Integer.parseInt(str);

System.out.println("String: "+str);

System.out.println("Equivalent int: "+j);

System.out.println("\tobject to Primitive number

using typeValue Conversion");

Integer a new Integer(1000);//creating object for float value int val a.intValue();
```

```
System.out.println("Integer: "+a);

System.out.println("Equivalent int value: "+val);

Float b=new Float(2000);//creating object for float value

float f=b.floatValue();

System.out.println("Float: "+b);

System.out.println("Equivalent float value: "+f);

}

}
```

Output

Integer to String Conversion

int Value: 100

Equivalent String: 100

String to Integer Conversion

String: 500

Equivalent int: 500

object to Primitive number using typeValue Conversion

Integer: 1000

Equivalent int value: 1000

Float: 2000.0

Equivalent float value: 2000.0

Uses of Wrapper Class

1) Wrapper classes are used to convert numeric strings into numeric values.

- 2) Wrapper classes are used to convert numeric value to string using wrapper class.
- 3) Wrapper class is a medium to store primitive data type in an object.
- 4) Using the `intValue()` method we can retrieve value of the object as its primitive data type.

Review Questions

1. Explain the use of wrapper classes in Java.
2. Explain the concept of wrapper classes with illustrative example.

Autoboxing

Definition: An automatic conversion of primitive data type into equivalent wrapper type is called as autoboxing.

This is a new feature of Java5

Example Program

```
class AutoboxExample
{
    public static void main(String args[])
    {
        //auto-boxing, an int is boxed into Integer object

        Integer obj = 111;

        //unboxing

        int val = obj.intValue();
    }
}
```

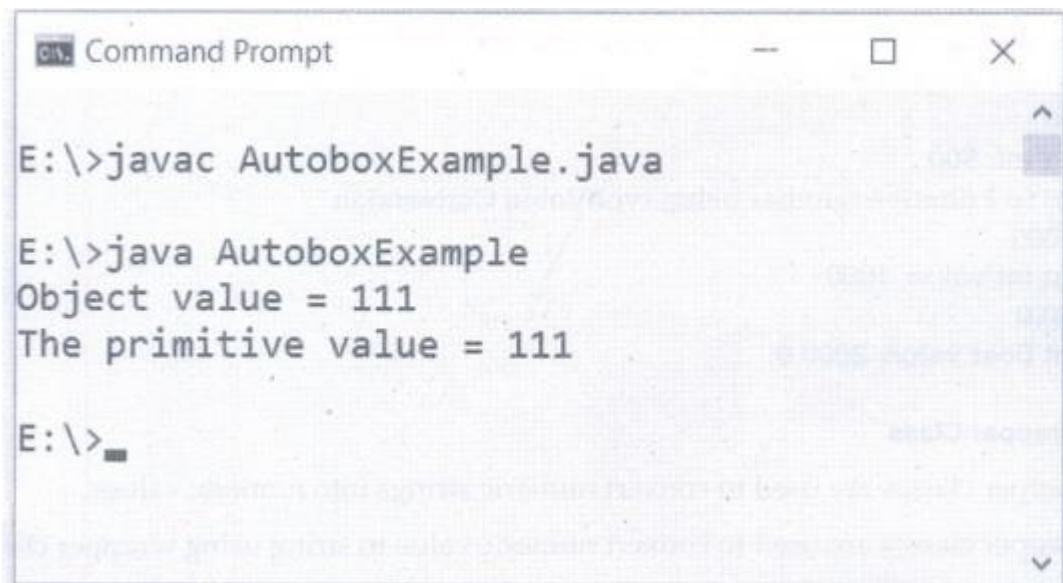
```
System.out.println("Object value = "+obj);

System.out.println("The primitive value = "+val);

}

}
```

Output



```
Command Prompt

E:\>javac AutoboxExample.java

E:\>java AutoboxExample
Object value = 111
The primitive value = 111

E:\>
```

Program Explanation

In above program,

- (1) We are autoboxing an integer type value.
- (2) Then using the `byteValue()` method we are unboxing this value. This method converts the given number into a primitive byte type and returns the value of integer object as byte.
- (3) Finally we are displaying both the object value and primitive value

Two Marks Questions with Answers

Q.1 What is an exception? Give example.

Ans: Exception is a mechanism which is used for handling unusual situation that may occur in the program. For example -

ArithmeticException: This exception is used to handle arithmetic exceptions such as divide by zero,

IOException: This exception occurs when an illegal input/output operation is performed.

Q.2 What will happen if an exception is not caught?

Ans. An uncaught exception results in invoking of the `uncaughtException()` method. As a result eventually the program will terminate in which it is thrown.

Q.3 What is the benefit of exception handling?

Ans. When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.

Q.4 What is the difference between error and exception in java?

Ans. Errors are mainly caused by the environment in which an application is running. For example, `OutOfMemoryError` happens there is shortage of memory. Where as exceptions are mainly caused by the application itself. For example, `NullPointerException` occurs when an application tries to access null object.

Q.5 What is compile time and run time error?

Ans;

1.The errors that are detected by the Java compiler during the compile time are called compiler time errors.

2. The runtime errors are basically the logically errors that get caused due to wrong logic.

Q.6 What is the use of try, catch keywords?

Ans. : try - A block of source code that is to be monitored for the exception.

catch - The catch block handles the specific type of exception along with the try block.

Q.7 What is the difference between throw and throws?

Ans. :

1. The throw keyword is used to explicitly throw an exception.
2. The throws keyword is used to declare an exception.

Q.8 What is ArrayIndexOutOfBoundsException?

Ans. When we use an array index which is beyond the range of index then ArrayIndexOutOfBoundsException occurs.

Q.9 What is the need of multiple catch?

Ans.:

- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.

Q.10 There are three statements in a try block - statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?

Ans. No. Once a try block throws an exception, remaining statements will not be executed. Control comes directly to catch block.

Q.11 Explain the types of exceptions.

Ans. There are two types of exceptions:

1. Checked Exception: These types of exceptions need to be handled explicitly by the code itself either by using the try-catch block or by using throws. These exceptions are extended from the java.lang.Exception class.

For example: IOException which should be handled using the try-catch block.

2. Unchecked Exception: These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions. These exceptions are extended from java.lang.RuntimeException class.

For example: ArrayIndexOutOfBoundsException, NullPointerException, RuntimeException.

Q.12 Does finally block get executed If either try or catch blocks are returning the control?

Ans. Yes, finally block will be always executed no matter whether try or catch blocks are returning the control or not.

Q.13 Can we have empty catch block?

Ans. Yes we can have empty catch block, but it is an indication of poor programming practice. We should never have empty catch block because if the exception is caught by that block, we will have no information about the exception and it will create problem while debugging the code.

Q.14 Which class is the super class for all types of errors and exceptions in java ?

Ans. The java.lang.Throwable is the super class for all types of errors and exceptions in java.

Q.15 What is runtime exceptions?

Ans. RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine. Runtime Exception is the parent class in all exceptions of the Java.

Q.16 What is exception handling?

Ans. Exception handling is a mechanism in which the statements that are likely to cause an exception are enclosed within try block. As soon as exception occurs it is handled using catch block.

Thus exception handling is a mechanism that prevents the program from crashing when some unusual situation occurs.

Q.17 What happens when the statement `int value = 25/0` is executed?

Ans. The exception Arithmetic Exception: Divide by zero will occur.

Q.18 What do you mean by threads in Java ?

Ans. Thread is tiny program running continuously. It is a light weight process in Java..

Q.19 Give the difference between process and thread.

Ans.

Sr. No.	Thread	Process
1.	Thread is a light-weight process.	Process is a heavy weight process.
2.	Threads do not require separate address space for its execution. It runs in the address space of the process to which it belongs to.	Each process requires separate address space to execute.

Q.20 What are different stages in thread?

Ans. Various stages in thread are

1. New state
2. Time waiting state
3. Runnable state
4. Waiting state

5. Blocked state

6. Terminated state

Q.21 What do you mean by synchronization ?

Ans. When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization.

Q.22 What are the three ways by which the thread can enter in waiting stage?

Ans. :

i) Waiting state: Sometimes one thread has to undergo in waiting state because another thread starts executing. This can be achieved by using wait() state.

ii) Timed waiting state: There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state. This can be achieved by using the sleep() command.

iii) Blocked state : When a particular thread issues an input/output request then operating system sends it in blocked state until the I/O operation gets completed. After

the I/O completion the thread is sent back to the runnable state. This can be achieved by using suspend() method.

Q.23 What is multithreading?

Ans. Multithreading is an environment in which multiple threads are created and they can execute simultaneously. The multiple threads can be created either by extending the thread class or by implementing the runnable interface.

Q.24 Mention two mechanisms for protecting a code block from concurrent access.

Ans. There are two mechanisms for protecting a code block from concurrent access -

1. Use of reentrant code
2. Use synchronized block

Q.25 What is meant by notify methods in multithreading?

Ans. The notify() method is used for inter thread communication. If a particular thread is in the sleep mode then that thread can be resumed by using the notify call.

Q.26 What are the two ways of creating a thread?

Ans. Thread can be created using

1. Thread class
2. runnable interface.

Q.27 Why do we need run() and start() method both? Can we achieve it with only run() method?

Ans. We use start() method to start a thread instead of calling run() method because the run() method is not a regular class method. It should only be called by the JVM. If we call the run() method directly then it will simply invoke the caller's thread instead of its own thread. Hence the start() method is called which schedules the thread with the JVM.

Q.28 What is the need for thread?

Ans. In Java threads are used to handle multiple tasks together. This kind of programming is called concurrent programming.

Q.29 Name any four thread constructor.

Ans. :

1. Thread()
2. Thread(String name).
3. Thread(Runnable target)

4. Thread (Runnable target, String name)

Q.30 When thread is initiated and created, what is its initial stage?

Ans. : A thread is in "Ready" state after it has been created and started. This state signifies that the thread is ready for execution. From here, it can be in the running

state.

Q.31 "Thread is a lightweight process" - Comment

Ans.

Threads do not require separate address for its execution. It runs in the address space of the process to which it belongs to. Hence thread is a lightweight process.

Q.32 Sketch the life cycle of thread

Ans.

