UNIT 3 EXCEPTION HANDLING AND MULTITHREADING

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. List checked exceptions of Java.

(Model Paper-II, Q1(e) | May-19(R16), Q1(e))

Answer :

The various checked exceptions of Java are as follows,

1. **ClassNotFoundException**

   This exception is thrown if the requested class does not exist or if the class name is invalid.

2. **CloneNotSupportedException**

   This exception is thrown if there is an attempt to clone (or create or copy of) an object that does not implement the "Cloneable" interface.

3. **IllegalAccessException**

   This exception is thrown if the requested class cannot be accessed.

4. **InstantiationException**

   This exception is thrown if there is an attempt to instantiate (or create) an object of an abstract class or interface.

5. **InterruptedException**

   This exception is thrown if one thread interrupts the another thread.

6. **NoSuchFieldException**

   This exception is thrown if there is an attempt to access a field that does not exist.

7. **NoSuchMethodException**

   This exception is thrown if there is an attempt to a method call that does not exist.

Q2. What are the benefits of exception handling?

Nov./Dec.-16(R13), Q1(e)

Answer :

The benefits of exception handling are as a follows,

(i) Exception handling can control run time errors that occurs in the program.

(ii) Exception handling can avoid abnormal termination of the program and also shows the behavior of program to users.

(iii) Exception handling can provide a facility to handle exceptions, throws message regarding exception and completes the execution of program by catching the exception.

(iv) Exception handling can separate the error handling code and normal code by using try-catch block.

(v) Exception handling can produce the normal execution flow for a program.

Q3. What is the difference between error and an exception? Nov./Dec.-17(R16), Q1(e)

Answer :

	Error		Exception
1.	Error occur due to the environment where the application is running.	1.	Exception occur due to the applications itself.
2.	They are of unchecked type.	2.	They have checked and unchecked types.
3.	They occur at runtime and are not known to the compiler.	3.	Checked exceptions are known to the compiler and unchecked are not known to the compiler since they occur at runtime.
4.	They belong to java.lang.error.	4.	They belong to java.lang.exception.
5.	Examples of errors are java.lang.stackoverflow error, java.lang.outofmemoryerror.	5.	Examples of exceptions are checked exceptions: SQL Exception, IO Exception, Unchecked Exceptions, Class Cast Exception, ArrayIndexOutofBoundException etc.
6.	They cannot be handled.	6.	They can be handled through try-catch block.

Q4. Discuss about multiple catch clauses. Nov./Dec.-17(R13), Q1(f)

Answer :

Multiple Catch Clauses

In Java programming, multiple catch clauses can be declared for single try clause. Every catch clause can catch and manipulate a different kind of exception. The following program demonstrates the declaration of multiple catch clause.

Example

```
class MultiCatchExample
{
public static void main(String[ ] args)
{
    int[ ] val1 = {21, 22, 23, 24, 25, 26, 27};
    int[ ] val2 = {8, 9, 0, 2, 5, 13};
    int x = 0;
    while (x < val1. length)
    {
        try
        {
            System.out.println("The division of"+ val1[x] + "and" + "val2[x] + "1s" +val1[x]/val2[x]);
            x++;
        }
        catch(ArrayIndexOut of Bounds Exception ex)
        {
            System.out.println("match not found");
        }
        catch(ArithemticException ex)
        {
            System.out.println("An exception" + ex + "raised");
        }
    }
}
```

In the above program, two different types of exception such as arithmetic exception and ArrayIndexOutofBoundsException are occurred. In the two arrays, the size of array val1 is greater than the array size of val2. Thus, an exception can be raised if the size of val2 is completed while performing an operation between the arrays val1 and val2.

Q6. Differentiate process based multitasking and thread based mulitasking.

Answer :

	Process-based Multitasking		Thread-based Multitasking
1.	More overhead is required in multitasking processes.	1.	Less overhead is required in multitasking threads.
2.	Each process has its own address space. Therefore, they are heavy weight.	2.	Threads share the same address space and the same process. Therefore, they are light weight.
3.	Interprocess communication is expensive and limited.	3.	Interthread communication is expensive.
4.	Context switching between processes is costly.	4.	Context switching between threads is cheaper.
5.	It is not controlled by Java.	5.	It is controlled by Java.

Q7. Define thread.

Answer :

Thread can be defined as a set of executable instructions that are executed independently. A program can be divided into multiple subprograms and each subprogram is called a thread. Every individual thread is executed separately, thereby decreasing the execution time of a program.

Q8. Describe the Main thread.

Answer : **(Model Paper-I, Q1(e) | Nov./Dec.-17(R13), Q1(e))**

The main thread is the first thread that will begin its execution immediately when the Java program starts up. The main thread is important for two reasons. First, it allows the other child threads to be created. Second, reason is that the thread perform various shutdown actions. Therefore, it must be the last thread to finish execution.

Even though the main thread starts executing immediately, it can be controlled using an object of Thread class. The object of Thread is created by using its static method currentThread( ). This method has the below general form,

```
public static Thread currentThread( )
```

This method when called will returns a reference to currently executing thread. After a reference to the main thread is obtained it can be controlled like any other threads. For example, the methods such as getName( ), setName( ), sleep( ) etc., can be invoked on the main thread.

Q9. How to create a thread? **May-19(R16), Q1(f)**

Answer :

A thread can be created by instantiating an object of type thread. Basically, a thread can be created in two ways.

1. **Implementing Runnable Interface**

The simple and easiest way for creating a thread is to create a class which implements the 'Runnable' interface. It is possible to construct a thread irrespective of object which implements 'Runnable'. 'This interface hides a unit of executable code. The implementation of 'Runnable', interface is done by implementing a single method called run( ), which contains the creation code similar to main thread, run( ) can invoke other methods, use other classes and declare variables.

Syntax

```
public void run( )
```

2. **Extending Thread Class**

This is another way for creating a thread. Here a new class which extends 'Thread' is created later an instance of that class is created. The class which is extending should override the run( ) method to switch the thread to runnable state (which is an entry point for a new thread). In order to start execution it should invoke start( ) method.

Q10. What is thread based preemptive multitasking? **Nov./Dec.-18(R16), Q1(e)**

Answer :

Thread-based multitasking allows two or more tasks of a single program to run simultaneously by the processor. For example, the text editor formatting the text at the same time printing the text. Here both tasks are performed by two separate threads. In thread-base multitasking, a thread is the smallest dispatchable unit of code.

Q10. How do we start and stop a thread?

Answer : April/May-18(R16), Q1(e)

Starting a Thread

A thread can be started by defining start( ) method. It initiates the execution of a thread. It calls run( ) method of runnable interface internally to execute the run( ) method code in another thread.

Example

```
class NewThread extends Thread
{
    public void run( )
    {
        System.out.println("My thread :");
    }
    public static void main(String args[ ])
    {
        New Thread t1 = new NewThread( );
        t1.start( );
    }
}
```

Stopping a Thread

Stopping a thread is difficult than that of starting a thread. The thread class contains a stop( ) method which can be called to stop a thread. But it gives rise to instabilities and even errors which are hard to detect. Because of this it is deprecated. The solution to stop a thread is to place a loop inside the run() method which will terminate when the variable changes to value.

Example

```
public void run( )
{
    while(oktorun = = true)
    {
    }
}
```

Q11. Write the complete life cycle of a thread.

Answer : (Model Paper-II, Q1(f) | April/May-18(R16), Q1(f))

Every thread has a life cycle, which consists of following states,

(a) Born state

(b) Runnable state

(c) Running state

(d) Blocked state

(e) Dead state.

A thread can be present in any one of these states and can switch from one state to another state using various methods. The following figure depicts the life cycle of thread,

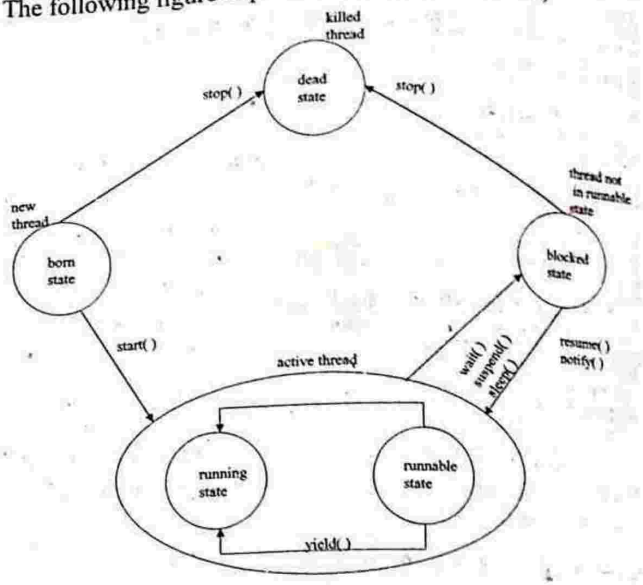

Figure: Life Cycle of a Thread

Q12. How do we set priorities for threads?

Answer : Nov./Dec.-18(R16), Q1(f)

Every thread is assigned a priority which specifies the amount of time used to access the C.P.U. The thread with highest priority thread contains less access time. While executing a lowest priority thread, if a highest priority is resumed back from its suspension, then the resumed thread can preempt the lowest priority thread. The priority of thread depends on the amount of time used to of access C.P.U and the highest potential access to C.P.U. The priority does not depends on the execution speed of thread. If a child thread is created, then the priority of it is similar to its parent thread. The thread which contain equal priority can access the C.P.U equally and the priority of thread can be altered. The syntax is,

final void setPriority(int priority)

In the above syntax, final is a keyword, void is the return type, setPriority is the method name which can be used to set priority and priority is an integer value which can store the priority of thread. Any thread can contain one of the available priority constant and these constant can be defined by thread class. The various priority constants are as follows,

MIN_PRIORITY = 1

NORM_PRIORITY = 5

MAX_PRIORITY = 10

The priority of a thread can be acquired by using the following syntax,

final int getPriority( );

13. What is synchronization and why is it important? (Model Paper-I, Q1(f) | Nov./Dec.-17(R16), Q1(f))

OR

What is thread synchronization? Explain.

Answer : Nov./Dec.-16(R13), Q1(f)

Synchronization can be defined as a process of enabling single thread to access shared resources. In multi-thread programming, synchronization threads are essential. When threads are attempting to access the shared resources synchronization, data can be altered and result an incorrect output. To overcome this situation, synchronization threads can be utilized.

In multithreaded programming, thread synchronization is used in order to maintain the data consistency. This can be done using the 'synchronized' keywords. When a method is declared as synchronized, then the Java complier creates a monitor semaphore and provides it to the threads object created for thread class which invokes the synchronized method initially. A monitor can be defined as an object which can act as a mutex or mutual exclusive lock. The thread which can be entered into the monitor can hold the lock and disables other thread to access the method. Once the execution of initial thread is complete, then the next waiting thread can enter into the monitor. While executing the thread, when another thread attempts to enter the monitor, then it can be suspended until the execution of the thread in monitor gets completed.

PART-B
ESSAY QUESTIONS WITH SOLUTIONS

3.1 EXCEPTION HANDLING

3.1.1 Fundamentals of Exception Handling, Exception Types

Q14. What is an exception? Explain how an exception can be handled in Java. Also, list the benefits of exception handling.

OR

What is an Exception? How is an Exception handled in JAVA? (Model Paper-II, Q6(a) | Nov./Dec.-18(R16), Q5(a))
(Refer Only Topics: Exception, Exception Handling)

OR

What are advantages of using Exception handling mechanism in a program? April/May-18(R16), Q5(a)
(Refer Only Topics: Benefits of Exception Handling, Example)

OR

What are the different ways to handle exceptions? Explain.
(Refer Only Topics: Exception Handling, Example)

Answer : Nov./Dec.-17(R16), Q7(a)

Exception

An exception can be defined as a error which can be occurred at runtime or at execution time.

Exception Handling

Exception handling can be defined as a mechanism of handling exceptions that can be occurred at run-time. This mechanism is commonly used to prevent the malfunctions such as computer deadlock or computer hanging some examples of runtime exception are divide by zero, array out of bounds exceptions.

The runtime, exception can be handled by using five keywords namely try, catch, throw, throws and finally. The code that can generate exception is usually placed in try block. If an exception is occurred, execution flow finds the similar catch block and leaves the try block. The catch block handles the exception and generates a message specifying the type of exception. Some of these exception types are as follows,

(i) ArithmeticException

(ii) ArrayIndexOutOfBoundsException.

However, there is no rule that the try block must contain corresponding catch block. Each try block may contain zero or multiple catch blocks.

When an exception is thrown and if it is matched with any of the catch block, then the statements of corresponding catch block can be executed, otherwise, the catch block can be skipped and the statements present next to the catch block will be executed. The finally block is declared after all the catch blocks whose code can be executed irrespective to the occurrence of exception. Finally block is optional.

If a method throw an exception, then it can be handled by catch block. This exception can be thrown using 'throws' clause. When control leaves the throws block, then it cannot return back to the 'throws' clause. If the thrown exception cannot be handled by any catch block, then it can be handled by default handler called as 'Termination Model Of Exception Handler'. The syntax of exception handling blocks is as follows,

```
try
{
    //Block of code to check exception
}
catch (Exception Type 1 exception object)
{
    //code to handle exceptions
}
```

```
catch (Exception Type 2 exception object)
{
    //code to handle exceptions
}
finally
{
    //code to be executed before/after try-catch block
}
```

Benefits of Exception Handling

The benefits of exception handling are as a follows,

i) Exception handling can control run time errors that occurs in the program.

ii) Exception handling can avoid abnormal termination of the program and also shows the behavior of program to users.

iii) Exception handling can provide a facility to handle exceptions, throws message regarding exception and completes the execution of program by catching the exception.

iv) Exception handling can separate the error handling code and normal code by using try-catch block.

v) Exception handling can produce the normal execution flow for a program.

vi) Exception handling mechanism can implement a clean way to propagate error. That is when an invoking method cannot manage a particular situations, then it throws an exception and asks the invoking method to deal with such situation.

vii) Exception handling mechanism develops a powerful coding which ensures that the exceptions can be prevented.

Example

For answer refer Unit-III, Q17, Topic: Program.

Q15. Discuss in brief about exception types.

Answer :

The class throwable is the root of all the exception types. Therefore, it is at the top of the class hierarchy. Exception class is used for exception conditions which needed to be caught by the user programs. User will create their own custom exception types as that of the subclasses of exception. The RuntimeException is the subclass of exception class. This will be automatically defined for programs written by users such as division by zero and invalid array indexing.

Error class will define the exceptions which are unexpected to be caught in normal conditions of a program. The exceptions which are of type error indicate the errors related to java run-time environment: One example of it would be stack overflow.

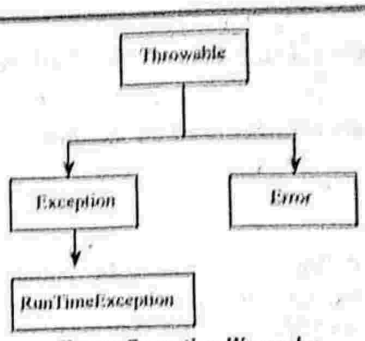

Figure: Exception Hierarchy

3.1.2 Termination or Resumptive Models, Uncaught Exceptions

Q16. What are termination and resumptive models? Write about uncaught exception.

Answer : Model Paper-I, Q6(a)

Termination Model

In this model of exception handling, the programmer will have to explicitly invoke the same method in which the error was occurred, and was transferred to the catch block so that the error can be handled. This explicit invocation is required, because few programming languages that use this kind of model do not allow the control to return back to the point where the error was occurred.

Resumptive Model

In this model of exception handling, the programmer will not have to explicitly invoke the same method in which the error was occurred and was transferred to the catch block so that the error can be handled. Because, the programming languages that use this kind of model will allow the control to return back to the point where the error was occurred.

Uncaught Exception

An exception occurred but it can not be caught in the try/catch block is known as an uncaught exception. This type of exception is handled by uncaught exception handler.

Example

```
class Divide
{
    public static void main(String args[ ])
    {
        int a = 0;
        int b = 55/a;
    }
}
```

In this example, when the exception is detected by the Java run-time system then an object is created and throws it. So, the execution of this program is halted and the thrown exception must be caught and required action are taken. But, this program do not have any exception handler, default exception handler is invoked which is provided by the Java run-time system. Hence, the output of this program displays an error (i.e., divide by zero).

3.1.3 Using Try and Catch, Multiple Catch Clauses, Nested Try Statements, Throw, Throws and Finally

Q17. Explain the usage of try statement in Java with an example program.

Answer :

Try Statement

The try statement can handle the runtime exceptions generated during the execution of a Java program. This can be done by enclosing the code which can generate exceptions in the try block. It is necessary for every try block to have one or more catch blocks. This catch block displays the type of exception which a programmer intends to catch.

Syntax

```
try
{
    //block of code which raises an exception
}
catch block
```

Program

```
public class Try_Example
{
    public static void main (String[ ] args)
    {
        int[ ] arr = new int[5];
        try
        {
            int n;
            for(x = 0 ; x < 10; x + +)
            {
                arr[ n ] = x;
            }
            System.out.println("Array [" + i +" ] is :");
            System.out.println(arr);
            }
            catch ( ArrayIndexOutofBoundsException ex)
            {
                System.out.println("Array elements exceeded boundary limits");
            }
        }
    }
```

An array of size '5' is declared, However, there are more than 5 elements that are being inserted. That is the insertion is exceeding the boundary limits. When this situation arise, the try block raises an error, which is handled by its respective catch block that specify "ArrayIndexOutofBoundsException".

8. Discuss about catch statement with an example.

Answer : Model Paper-II, Q6(b)

Catch Statement

The catch block handles the exceptions that are thrown by try block. The primary objective of well-constructed catch clause is to resolve the exceptional condition and continue with the execution part. Generally, a try and its catch statement form a single unit. The scope of the catch clause is confined to only those statements declared in its respective try block. A catch statement cannot handle the exceptions which are thrown by another try statement.

Syntax

```
try
{
    //code
}
catch(ExceptionType excep_name)
{
    //code to handle exception
}
catch(ExceptionType excep_name)
{
    //code to handle exception
}
```

Here, catch block is an exception handler that is responsible for handling the type of exception specified in the argument. The arguments, "ExceptionType" specifies the exception to be handled and "excep_name" specifies the name of exception that can be referred by an handler. The catch block contains code that is executed for handling the raised exception .The system invokes the handler, when its ExceptionType matches with the type of exception thrown.

Here, ExceptionType is the type of exception which can be handled by catch block. Exception object is the object which is by created for the exception.

Program

```
public class CatchExample
{
    public static void main (String[ ] args)
    {
        int x;
        int y;
        int z;
        x = 10;
        y = 0;
        try
        {
            z = x/y;
            System.out.println("The division of x and y is" + z );
        }
        catch(ArithmeticException ex)
        {
            System.out.println("An arithmetic exception of divide by zero is caught");
        }
    }
}
```

In the above program, an exception is raised when 10 divides 0. When the try block raises ArithmeticException, then the catch block handles it and displays an error message. "An arithmetic exception of divide by zero is caught". Meanwhile, the program enters into safe state and rest of the program can be executed without any issues. However, when try block does not throw an exception, then the code implemented in catch block cannot be executed.

Q19. Explain the concept of multiple catch clauses. Is it possible to nest the try statement? If yes, explain.

OR

Write a Java program that illustrates the application of multiple catch statements.

(Refer Only Topic: Multiple Catch Clauses, Program)

Answer :

Multiple Catch Clauses

A program code can raise more than one exception, which is basically handled by defining two or more catch clauses. Each catch statement handle a different type of exception. When an exception is raised, every catch statements defined within a particular try block is examined sequentially. If the exception raised matches with any of the ExceptionType defined on a catch statement, then that particular catch block is executed. Once a catch statement is executed, the remaining catch statements are executed in the similar fashion.

A programmer before using multiple catch clauses must always declare the exception subclasses prior to the declaration of their respective superclasses. This is because, a catch statement of a super class with not only catch the exception of that class but also catch the exceptions raised by any of its subclass.

Nested Try Statement

Yes, in Java it is possible to nest the try statement. That is, if a try statement is defined inside the block of another try statement, then the block is referred as Nested try block. The execution of this block is done in the following manner.

(i) Initially, the exception raised by any incoming try statement is pushed on the stack.

(ii) Then, the catch statement are inspected. If an inner try statement doesn't have its respective catch handler, then the stack of that particular try statements is not executed and the next try catch statement of another try statement is inspected for a match.

(iii) This inspection of catch statement continues, until one of the catch statement of a try block is executed successfully or until the nested try statements are exhausted.

Program

```
import java.io.*;
class NestedTry
{
    public static void main(String args[ ])
    {
        try
        {
            int x = args.length;
            int y = 55/x;          //case1: When no command-line argument is available then this statement will
                                   //produce a divide by zero exception
            System.out.println("The value of x is" + x);
            try          //nested block
            {
                if(x = = 1)
                x = x/(x – x);     //case2: when one command-line argument is available, this statement
                                   // will produce divide by zero exception.
                if(x = = 2)        //case3: when two command-line arguments are available these following statements
                                   //will produce an out of bounds exception.
                {
                    int z[ ] = {5};
                    z[55] = 88;
                }
            }
```

```
            catch(ArrayIndexOutOfBoundsException ex)
            {
                System.out.println("The array index out-of-bounds:" + ex);
            }
            catch(ArithmeticException ex)
            {
                System.out.println("Divide by zero error:" + ex);
            }
        }
    }
```

Output

The value of x is 2

The array index out-of-bounds: java:lang.ArrayIndexOutOfBoundsException: 55.

Q20. Describe how throw statement is used in Java. Explain with an example.

Answer : **Model Paper-II, Q7(a)**

'throw'

'throw' is a Java keyword used in exception handling. Generally a try block checks if any exceptions are raised. And when an error occurs, it throws the error which is caught by the catch statement. Basically, the exceptions thrown by the Java run-time system are being caught, but throw statement allows a program to throw an exception explicitly.

Syntax

throw ThrowableInstance;

The ThrowableInstance should be an object of either Throwable class or any of its subclass. The int, char, String etc., type of objects cannot be thrown. The Throwable object can be created in two ways.

(i) **As a Parameter into Catch Clause**

```
catch(NullPointerException e)
        {
          //statements
          throw e;
        }
```

(ii) **Using New Operator**

```
throw new NullPointerException;
```

When an error occurs the execution gets stopped at throw statement. The next try block is checked to see if it has catch block. If no catch block exist, then control passes to the next try block. This is done until a matching catch is found. If no catch is matched, the situation is handled by default exception handler. The default exception handler terminates the program and displays them information stack trace.

Example Program

```
class ThrowUsage
{
static int x=5, y=0, z;
static int div( )
{
  try
  {
    if(y = = 0) //throws an exception throw new ArithmeticException("Division by zero");
    else
      return x/y;
  }
```

```
    catch(ArithmeticException ae)
    {
      System.out.println("ArithmeticException Exception caught at div( )");
      throw ae; // rethrows an exception
    }
  }
  public static void main(String args[ ])
  {
    try
    {
      z=div( );
      System.out.println("z=" +z);
    }
    catch(ArithmeticException ae)
    {
      System.out.println("Exception Recaught at main( ):" +ae);
    }
  }
}
```

Q21. Explain in details about throws statement along with an example.

Answer :

'throws' Statement

Throws clause handles the unhandled exceptions that are generated by a method. This clause list the types of exceptions that can be thrown by a method. The throws clause can be used for specifying any exception except the exception belonging to erro class or any of its subclass.

Syntax

```
return-type method-name(arg-list) throws exception-list
{
    //body
}
```

In the above syntax, exception-list is the list of exception separated by comma. The throws keyword is used to list the exceptions that can be thrown by a method. If the possible exceptions are not listed in the method of throws clauses, the compile time error is generated.

Example Program

```
class ThrowsExample
{
    public static char prompt (string st) throws Java.io.IOException
    {
        System.out.println(st+ " ");
        return(char) system.in.read ( );
    }
    public static void main(String[ ] args)
    {
        char c;
        try
        {
            c = promt("Enter a character");
        }
```

```
        catch(java.io.IOException en)
        {
            System.out.println("An exception occured");
            c = 'n';
        }
        System.out.println("The entered character is" +c);
        }
    }
```

The IOException can be defined in the package Java.io.

Q22. How are finally statements used in Java? Explain in detail.

Answer :

'finally' Statement

Every try block should be associated with atleast one catch or finally block. However, it is optional to have a finally block within a program. If a finally block is included in a program, then it will be executed after the try/catch block but before the code following this try/catch block. The finally block will be executed irrespective of the occurrence of exceptions. A finally block helps in closing opened files, deallocating the allocated memories etc.

Syntax

The syntax of the finally block is,

```
try
{
    //statements
}
catch(...)
{
    //statements
}
catch(...)
{
    //statements
}
finally
{
    //statements
}
```

Example

```
class Exfinally
{
static void proOne( )
{
  try
  {
    System.out.println("Process One");
    throw new RuntimeException("demo");
  }
  finally
  {
    System.out.println("Process One's finally");
  }
}
static void proTwo( )
{
  try
  {
    System.out.println("Process Two");
    return;
  }
  finally
  {
    System.out.println("Process Two's finally");
  }
}
static void proThree( )
{
  try
  {
    System.out.println("Process Three");
  }
  finally
  {
    System.out.println("Process Three's finally");
  }
}
public static void main(String args[ ])
{
  try
  {
    proOne( );
  }
  catch(Exception e)
  {
    System.out.println("Caught the exception");
  }
  proTwo( );
  proThree( );
}
}
```

3.1.4 Built-in Exceptions

Q23. Explain in detail the built-in exceptions in Java.

OR

Differentiate between Checked and Unchecked Exceptions with examples.

Answer : (Model Paper-I, Q6(b) | Nov./Dec.-17(R16), Q6(a))

Java defines several exception classes inside java.lang package. Among these exceptions, the RuntimeException is the most common. Many exceptions derived from RuntimeException are available automatically. Generally, the two types of exceptions are,

(a) Checked exceptions

(b) Unchecked exceptions.

(a) Checked Exceptions

Checked exceptions are the exceptions thrown by a method if it encounters a situation which is not handled by itself. All classes that are derived from "Exception" class but not "RuntimeException" class are checked exceptions. Whenever a method is declared or called it is checked by compiler to determine whether the method throws checked exceptions or not. The methods that throw checked exceptions must handle them by providing a throws clause containing the list of checked exceptions that it can throw in the method declaration or it must handle them by providing a try-catch block. If these exceptions are not handled the compiler issues error message indicating that the exception must be caught or declared.

The list of Java's checked exceptions defined in java.lang package is given below.

1. **ClassNotFoundException**
   This exception is thrown if the requested class does not exist or if the class name is invalid.
2. **CloneNotSupportedException**
   This exception is thrown if there is an attempt to clone (or create or copy of) an object that does not implement the "Cloneable" interface.
3. **IllegalAccessException**
   This exception is thrown if the requested class cannot be accessed.
   Example: Trying to access private or protected methods or instance variables.
4. **InstantiationException**
   This exception is thrown if there is an attempt to instantiate (or create) an object of an abstract class or interface.
5. **InterruptedException**
   This exception is thrown if one thread interrupts the another thread.
6. **NoSuchFieldException**
   This exception is thrown if there is an attempt to access a field that does not exist.
7. **NoSuchMethodException**
   This exception is thrown if there is an attempt to a method call that does not exist.

Example for Checked Exception

Program

```
import java.io.*;
class program
{
    static void func( ) throws IllegalAccessException
    {
        try
        {
            System.out.println("From func( )");
            throw new IllegalAccessException
            ("example");
        }
        finally
        {
            System.out.println("From finally block of func( )");
        }
    }
}
public static void main(String args[ ])
{
    try
    {
        func( );
    }
    catch(Exception ex)
    {
        System.out.println("Exception caught" +ex);
    }
}
```

The output generated after running the above program is shown below.

Output

```
From func( )
From finally block of func( )
Exception caught java.lang.IllegalAccess Exception: example
```

In the above program, the main( ) method will first call func( ). In the func( ) method, it is declared that it throws an IllegalAccessException. When an exception occurs within the try block, it will be thrown and the control will come out of try block. Then, the finally block is executed. The exception thrown from try block is caught in the catch block present in main( ) method. That is, after executing the finally block, the control will go to the catch block.

If an exception does not occur within the try block, the finally block will anyhow be executed after the try block. But, the control will not go to the catch block, as the exception is not thrown.

Runtime Exceptions/Unchecked Exceptions

The run-time conditions that can generally occur in any Java method are represented as Runtime Exceptions. These exceptions are also called 'unchecked exceptions'. A Java method does not require to declare that it will throw any of the run-time exceptions.

The standard run-time exception classes that are defined in java.lang package are,

1. **ArithmeticException**

   This exception will be thrown when an exceptional arithmetic condition has occurred.

   Example: Integer division by zero.

2. **ArrayIndexOutOfBoundsException**

   This exception will be thrown when an array object detects an out of range index. This condition occurs when the index is greater than or equal to the size of the array or less than zero.

3. **ArrayStoreException**

   This exception will be thrown when trying to store a value in the array that does not have a type as that of the array.

4. **ClassCastException**

   This exception will be thrown while trying to cast a reference to an object of inappropriate type.

   Example: Trying to cast integer object to string object.

5. **IllegalArgumentException**

   This exception will be thrown when an illegal argument is passed to a method.

   Example: A method trying to acquire non-empty string as parameter where input string is null.

6. **IllegalMonitorStateException**

   This exception will be thrown if any of the wait( ), notifyAll( ) or notify( ) methods of the object are called from a thread that does not possess the monitor of that object.

   Example: Trying to notify other threads that wait on an object's monitor.

7. **IllegalStateException**

   This exception will be thrown when a method is invoked while the run-time environment is not in an appropriate state to perform the operation that has been requested.

   Example: Using a scanner which is closed.

8. **IllegalThreadStateException**

   This exception will be thrown when an operation is performed on a thread that is illegal with respect to the current state of the thread.

   Example: Trying to use a dead thread.

9. **ArrayIndexOutOfBoundsException**

   When an array index is out of bounds, then the ArrayIndexOutOfBoundsException is thrown.

   Example: Trying to access an array element with illegal index.

10. **NullPointerException**

    This exception will be thrown when an object is accessed using a null object reference.

    Example: Calling a class method that has no object instance.

11. **NegativeArraySizeException**

    This exception will be thrown when an array of negative size is created.

12. **NumberFormatException**

    This exception will be thrown when the numeric information in a string cannot be parsed.

    Example: Converting an object of string to int type.

13. **SecurityException**

    This exception will be thrown when an operation is performed that does not conform to the security policy that is being implemented by the object of "Security Manager".

    Example: Using a package name that is already defined in Java.

14. **StringIndexOutOfBoundsException**

    This exception will be thrown when an Out-of-range index is detected by a 'String' object or a 'StringBuffer' object.

    Example: Trying to access a string element with illegal index.

Example

For answer refer Unit-III, Q19, Topic: Program.

Q24. What do you mean by an exception and error? Give the hierarchy of the exceptions in java.

Answer : Nov./Dec.-16(R13), Q6(b)

Exception

For answer refer Unit-III, Q14, Topic: Exception.

Error

It is a known fact that human beings are prone to errors. Obviously, we (the Java professionals) are not an exception to this. Errors occur due to mistakes in the design and coding of an application. Errors result due to bad coding these are nothing but syntax errors.

Exception Hierarchy

Exception hierarchy consists of all the possible exceptions that can occur in a Java program. In this hierarchy, "Object" is the root class and has a subclass called "Throwable". The "Throwable" class inturn has two subclasses one is the 'Exception' class and the other is the 'Error' class. The java.lang package defines both the standard exception classes and error classes.

Standard Exception Classes (or) Built-in Exception

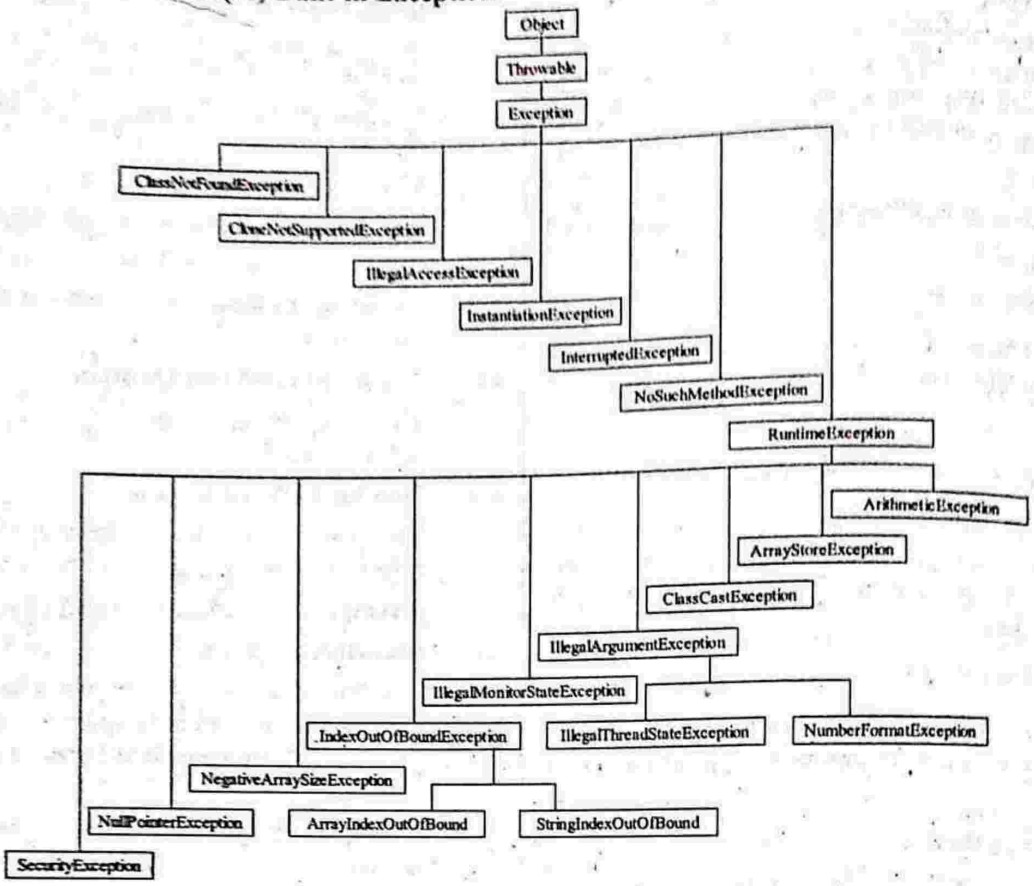

Figure (i): Standard Exception Clauses

For remaining answer refer Unit-III, Q23.

ERROR Classes

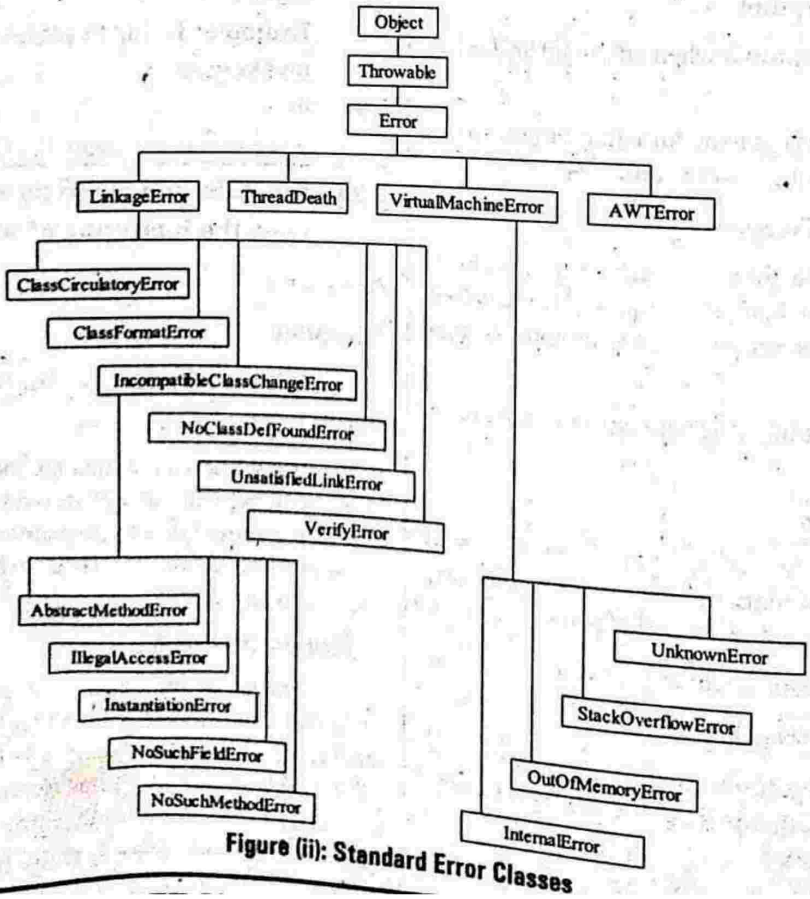

Figure (ii): Standard Error Classes

The standard error classes are as follows,

Error

This class consists of several subclasses. Whenever an unpredictable error is occurred, any of its appropriate subclasses will be thrown. As the errors have unpredictable nature, it is not required to declare them in the 'throws' clauses when the methods throw objects that are instances of 'Error' class or any one of its subclasses.

LinkageError

This error is thrown when there is a problem in resolving the class reference, if there are any appropriate subclasses.

Examples

1. When it is difficult to find class definition or,
2. When the current definition is not compatible with the excepted class definition.

ThreadDeath

The 'thread' object contains a 'stop( )' method that will throw this error in order to kill the thread.

ClassCirculatoryError

This error will be thrown if there is a circular reference between classes while initializing the class.

ClassFormatError

This error will be thrown if the file format containing the definition of a class holds an error.

IncompatibleClassChangeError

When the references to a class are made in an incompatible way by another class, this error or any one of its subclasses will be thrown.

Example: Working with Java libraries which internally depends on eglib.

NoClassDefFoundError

When there is a problem in finding the class definition, this error will be thrown.

Example: Trying to load a class at run-time using class. ForName( ) which is not present in class path.

UnsatisfiedLinkError

When there is a problem in finding the implementation of a native method, this error will be thrown.

Example: Using a native library like .dll on windows.

VerifyError

When the byte code verifier finds out that a well-formed class file holds a kind of security problem or internal inconsistency, this error will be thrown.

AbstractMethodError

When an abstract method is invoked, this error will be thrown.

(xi) IllegalAccessError

When a class does not have access to a particular field or a method but it tries to access that field or call that method, this error will be thrown. A compiler catches this error.

Example: Modifying a final property of the base class, or accessing a private method or property.

(xii) InstantiationError

When an abstract class or an interface is instantiated, this error will be thrown. A compiler catches this error.

(xiii) NoSuchFieldError

When an instance or a class variable is referenced that is not defined in the current class definition, this error will be thrown.

Example: Modification of a class definition which is incompatible with the curved definition.

(xiv) NoSuchMethodError

When a method is referenced that is not defined in the current class definition, this error will be thrown.

Example: Modification of a class definition which is incompatible with current definition.

(xv) VirtualMachineError

Whenever a Java Virtual Machine encounters an error, any of the appropriate subclasses of this error will be thrown.

(xvi) InternalError

When the virtual machine contains an internal error, this error will be thrown as a signal.

(xvii) OutOfMemoryError

When the memory allocation could not succeed, this error will be thrown.

Example: Requested array size exceeds virtual machine limit.

(xviii) StackOverflowError

When the virtual machine experiences a StackOverflow within it, this error will be thrown.

Example: Infinite Recursion

(xix) UnknownError

When an error is detected that belongs to unknown origins within the run-time system, this error will be thrown.

(xx) AWTError

AWT error is a subclass of error that is thrown when a serious problem occurs in AWT.

3.1.5 Creating Own Exception Subclasses

Q25. Explain the process of creating an own exception subclass with an example.

OR

How to create a user-defined exception? Explain with an example.

Answer : Nov./Dec.-16(R13), Q7(b)

An exception subclasses can be created if there is a need to generate own exceptions in the Java program. As a matter of fact, most of the exceptions are defined in Java's Built-in exceptions. However, sometimes program has to generate own exceptions. Therefore, Java compiler provides a flexibility to define own exceptions in the program, there by increasing the efficiency of the program.

The subclasses can be created from Exception class, which is inturn derived from another class called as Throwable. The Exception class and the subclasses created for Exception class can inherit required methods from Exception class. The exception subclasses and Exception class does not implement anything, but overrides the required methods from 'Thrownable' class.

The methods defined by Throwable class are as follows,

(a) Throwable fillInStackTrace( )

This method is used to throw the object of Throwable class which contains complete stack trace. The thrown object can be rethrown if necessary.

(b) Throwable getCauses( )

This method is used to return the exception which can be a causes to create current exception. Where the current exception is generated without any help of others exception, then null is returned.

(c) String getMessage( )

This method is used to return the local description of the exception.

(d) Stack TraceElement[ ] getStackTrace( )

This method can return an array stack track element class which contains stack trace in an array. This method can return single element at a time. The method present in top of the stack is the final method which can be invoked before the exception way thrown. The stack trace element in the trace. The data is like the method name present as stack trace element.

(e) Throwable initCauses(Throwable causes exception)

This method returns a reference to the exception and also association the causes exception with the calling exception as a reason of calling exception.

(f) void printStatement( )

This method prints the stack trace.

(g) void printStackTrace(PrintStream Stream)

This method forwards the stack trace to specified stream.

(h) void printStackTrace(PrintWriter Stream)

This method sends the required stack trace to specified stream.

(i) void setStackTrace(StackTrace Elements[ ])

This method assigns the element that are passed into stack trace elements. This method is commonly used in specialized applications.

(j) String toString( )

This method is used to return the object of strings which holds the description of exception. Generally, This method is used while outputting throwable object and can be invoked using the method println(').

The exception uses two constructors to create exception subclasses. They are as follows,

(a) Exception( )

(b) Exception(String Message).

(a) Exception( )

This constructor creates an exception with no description.

(b) Exception(String Message)

This constructor creates an exception with some description. But, this description may confuse the programmer as it even displays the exception name followed by a colon. Therefore, it is necessary to override the toString( ) method. So as to display the output in a desirable format without the name of exception and a colon.

gram

```
class ExceptionSubclass extends Exception
{
    private int info;
    ExceptionSubclass (int x)
    {
        info = x;
    }
    public String toString ( )
    {
        return"Exception Subclass["+info+"]";
    }
}
class exceptionsubclass
{
    void calculate(int x)throws ExceptionSubclass
    {
        System.out.println("calculate is invoked with("+x+")");
        if(x>20)
            throw new ExceptionSubclass(x);
        System.out.println("exit");
    }
    public static void main(String [ ] args)
    {
        try
        {
            calculate(5);
            calculate(35);
        }
        catch(ExceptionSubclass ex)
        {
            System.out.println("The generated exception is" +ex);
        }
    }
}
```

In the above program, own exception is created. The class ExceptionSubclass is defined from Exception class and it also ains an overload method called as toString( ) which displays the exception value.

ExceptionSubclass is created with a method called as calculate ( ), which can throw the object of class ExceptionSubclass. s exception can be thrown when an argument > 20 is passed to the method calculate ( ) and display the error message which efined in the catch block.

Q26. Write a java program that demonstrates how certain exception types are not allowed to be thrown.

April/May-18(R16), Q6(b)

Answer :

An exception in a Java instance initialization block (not class initialization) is not allowed to be thrown because there should be a possible code path which does not allow the program to throw an exception. A simple java program that demonstrate the exceptions that are not allowed to be thrown is as given below.

Example

```
class Example
{
    class X
    {
        int a;
        {{a = 0/0;}}
    }
    class Y
    {
        Integer b = null;
        int a;
        {{a = b intValue( );}}
    }
    class Z
    {
        {{throw new NullPointerException( );}}
    }
    class RunEx extends RuntimeException{ }
    class V
    {
        {{throw new RunEx( );}}
    }
    void run( )
    {
        try{X x = new X( );}
        catch(Exception e) {System.out.println(e);}
        try{Y y = new Y( );}
        catch(Exception e) {System.out.println(e);}
        try{Z z = new Z( );}
        catch(Exception e) {System.out.println(e);}
    }
    public static void main(String args[ ])
    {
        Example ex = new Example( );
        ex.run( );
    }
}
```

3.2 MULTITHREADING

3.2.1 Differences between Thread-based Multitasking and Process-based Multitasking

Q27. Compare and contrast process-based multitasking and thread-based multitasking. Model Paper-II, Q7(b)

OR

What are the different ways that are possible to create multiple threaded programs in java? Discuss the differences between them. April/May-18(R16), Q7(a)

OR

How many ways are possible in java to create multiple threaded programs? Discuss the differences between them.

Nov./Dec.-17(R16), Q7(b)

Answer :

Multithreading is a concept of dividing a program into two or more subprograms that can run concurrently. Each subprogram is called a thread. Each thread executes separately independent of other threads. Thus multithreading is a specialized form of multitasking.

There are two types of multitasking.

1. Process-based multitasking
2. Thread-based multitasking.

1. Process-based Multitasking

A process is a program that is in execution by the processor. Process-based multitasking allows two or more programs to run concurrently by the processor. For example, using text editor while running the Java compiler. In process-based multitasking, a program is the smallest dispatchable unit of code.

2. Thread-based Multitasking

Thread-based multitasking allows two or more tasks of a single program to run simultaneously by the processor. For example, the text editor formatting the text at the same time printing the text. Here both tasks are performed by two separate threads. In thread-base multitasking, a thread is the smallest dispatchable unit of code.

Following is the comparison between process-based and thread-based multitasking.

Process-based Multitasking	Thread-based Multitasking
1. More overhead is required in multitasking processes.	1. Less overhead is required in multitasking threads.
2. Each process has its own address space. Therefore, they are heavy weight.	2. Threads share the same address space and the same process. Therefore, they are light weight.
3. Interprocess communication is expensive and limited.	3. Inter-thread communication is expensive.
4. Context switching between processes is costly.	4. Context switching between threads is cheaper.
5. It is not controlled by Java.	5. It is controlled by Java.

3.2.2 Java Thread Model

Q28. Define thread. Explain the thread model of Java.

OR

Explain about Java thread Model.

(Refer Only Topic: Java Thread Model)

Nov./Dec.-17(R13), Q7(b)

Answer :

Thread

Thread can be defined as a set of executable instructions that are executed independently. A program can be divided into multiple subprograms and each subprogram is called a thread. Every individual thread is executed separately, thereby decreasing the execution time of a program.

Java Thread Model

The Java run-time system and its class libraries are designed in such a way that it depends on multi-threading concept. Java offers asynchronous thread environment which helps in efficient utilization of CPU cycles.

In Java, multithreading is preferred more than the single threaded system. Single threaded system make use of a method named event loop along with polling. According to this method, a single thread present in the system runs a loop infinite number of times (i.e., infinite loop). The polling mechanism is responsible for selecting an event from the queue which decides the next action to be performed while the selection of this event, the event loop gives the control to the required event handler. The CPU will be idle until the event handler returns the response due to which the CPU time is wasted. So, a single event dominates the other events and restricts them to be processed.

In a single-threaded model, a single thread is responsible for blocking the execution of other threads unless it complete its execution. Hence, the resources are wasted.

In contrast to single-threaded model, multithreading in Java offers the following benefits.

(i) It removes the loop and polling mechanism.

(ii) A single thread can be stopped without affecting the execution of other parts of the program.

Java thread model can be defined into the following three parts.

1. Thread priorities
2. Synchronization
3. Messaging.

1. Thread Priorities

For answer refer Unit-III, Q33.

2. Synchronization

For answer refer Unit-III, Q34.

3. Messaging

A program can be divided into a number of threads wherein every individual thread can communicate with each other. Java supports this messaging service with low-cost for the threads to communicate. It provides a set of methods that support inter-thread communication.

3.2.3 Creating Threads

Q29. Explain in detail the process of creating thread with an example.

OR

What does extending a thread mean? Explain by means of a program.

(Refer Only Topic: Extending Thread Class)

Answer : Nov./Dec.-16(R13), Q7(a)

Creation of Thread

A thread can be created by instantiating an object of type thread.

Basically, a thread can be created in two ways.

(a) By implementing Runnable Interface

(b) By extending Thread Class.

(a) Implementing Runnable Interface

The simple and easiest way for creating a thread is to create a class which implements the 'Runnable' interface. It is possible to construct a thread irrespective of object which implements 'Runnable'. 'This interface hides a unit of executable code. The implementation of 'Runnable', interface is done by implementing a single method called run( ), which contains the creation code similar to main thread, run( ) can invoke other methods, use other classes and declare variables.

Syntax

```
public void run( )
```

After creating a class which implements 'Runnable', an object of type 'Thread' is instantiated from within the class. One of the important constructors defined by 'Thread' is,

```
Thread(Runnable thObj, String tName)
```

Here,

thObj – Instance of a class which implements the runnable interface.

tName – Name of the new thread.

Once the new thread is created, it execution can be initiated using its start( ) method.

Example

```
class MyThread implements Runnable
{
    String t_name;
    Thread t;
    MyThread(String name)
    {
        t_name = name;
        t = new Thread(this, t_name);
        System.out.println("MyThread:" + t);
        t.start( );
    }
    public void run( )
    {
        try
            {
            for(int j = 5 ; j > 0 ; j--)
            {
                System.out.println(t.getName( )+ " " + j);
                Thread.sleep(300);
            }
        }
        catch(InterruptedException ie){
                System.out.println(t.getName( ) + "Interrupted");
        }
        System.out.println(t.getName( ) + "Exiting...");
    }
}
public class Main{
    public static void main(String[ ] args)
    {
        new MyThread("First");
        try
        {
            Thread.sleep(5000);
        }
```

```
            catch(InterruptedException ie)
            {
                System.out.println("Main thread has been interrupted!");
            }
            System.out.println("Main thread is exiting...");
        }
    }
```

(b) Extending Thread Class

This is another way for creating a thread. Here a new class which extends 'Thread' is created later an inatance of that class is created. The class which is extending should override the run( ) method to switch the thread to runnable state (which is an entry point for a new thread). In order to start execution it should invoke start( ) method.

Program

```
class NewThread extends Thread
{
    public void run( )
    {
        for (int i = 1; i <5; i++)
        {
            System.out.println(this.getName()+ "running");
            try
            {
                Thread.sleep(1000);
            }
            catch (Exception e)
            {
                System.out.println("caught exception :"+e);
            }
        }
    }
}
    public class ThreadExample
    {
        public static void main(String args[ ])
        {
            NewThread t1 = new NewThread( );
            NewThread t2 = new NewThread( );
            NewThread t3 = new NewThread( );
            t1.start( );
            t2.start( );
            t3.start( );
        }
    }
```

Q30. Differentiate between multiprocessing and multithreading. What is to be done to implement these in a program?

Answer : (Model Paper-I, Q7(a) | Nov./Dec.-18(R16), Q7(a))

	Multiprocessing		Multithreading
1.	Multiprocessing is a method in computing where different parts of a task are shared among two or more processors.	1.	Multithreading is a concept in which a program is divided into two or more subprograms. Each subprogram is called as a thread.
2.	It allows single process to run on multiple processors.	2.	It allows the programmer to run multiple threads concurrently.
3.	It utilizes multiple CPUs.	3.	It makes maximum use of the CPU and write efficient programs.
4.	There are two types of multiple processing. They are asymmetric multiprocessing and symmetric.	4.	There is no such classification in multithreading.
5.	It provides advantages such as, (i) Processor affinity (i.e., avoidance of process migration). (ii) Load balancing (i.e., equal distribution of work load amount CPUs).	5.	It provides advantages such as, (i) Idle time of the microprocessor can be reduced. (ii) Performance of processes is improved.

Multithreading can be implemented in two ways,

1. By implementing runnable interface.
2. By extending thread class.

1. **Implementing Runnable Interface**

For answer refer Unit-III, Q29, Topic: Implementing Runnable Interface.

2. **Extending Thread Class**

For answer refer Unit-III, Q29, Topic: Extending Thread Class.

Multi processing can be implemented by processing different threads concurrently with the help of multiple processors. Wherein each thread is executed independently with another thread running on some other processor.

Q31. Write a program that creates two threads. First thread prints the numbers from 1 to 100 and the other threads prints the numbers from 100 to 1.

Answer : Nov./Dec.-18(R16), Q7(b)

Program that Creates Two Threads

First Thread (1 to 100) and Second Thread (100 to 1)

```
class Thread1 extends Thread
{
    public void run( )
    {
        for(int k = 1; k<=100; k++)
        {
            System.out.println("Thread1:"+k);
            try
            {
                Thread.sleep(500);
            }
```

```
                catch(InterruptedException ie)
                {
                    ie.printStackTrace( );
                }
            }
        }
    }
    class Thread2 extends Thread
    {
        public void run( )
        {
            for(int k = 100; k>= 1; k--)
            {
                System.out.println("Thread2:" +k);
                try
                {
                    Thread.sleep(500);
                }
                catch(InterruptedException ie)
                {
                    ie.printStackTrace( );
                }
            }
        }
    }
    public class main
    {
        public static void main (String[ ] args)
        {
            Thread1 th1 = new Thread1( );
            Thread2 th2 = new Thread2( );
            th1.start( );
            th2.start( );
            try
            {
                th1.join( );
                th2.join( );
            }
            catch(InterruptedException ie)
            {
                ie.printStackTrace( );
            }
        }
```

Q. Write a program to create four threads using Runnable interface.

(April/May-18(R16), Q7(b) | Nov./Dec.-17(R16), Q6(b))

Answer :

Program

```
class FourT implements Runnable
{
@Override
public void run( )
{
String name = Thread.currentThread( ).getName( );
try
{
System.out.println("Starting%s\n", name);
Thread.sleep(1500);
System.out.println("Ending%s\n", name);
}
catch(InterruptedException e)
{
e.printStackTrace( );
}
}
}
public class ThreadJoin
{
public static void main(String[ ] args) throws InterruptedException
{
FourT tk = new FourT( );
System.out.println("1.Working with multiple threads using thread join:");
ThreadJoin(tk);
System.out.println("2. Working with multiple threads WITHOUT thread join:");
WithoutThreadJoin(tk)
}
private static void WithoutThreadJoin(FourT tk) throws InterruptedException
{
Thread t1 = new Thread(task, "Thread-1 without join");
Thread t2 = new Thread(task, "Thread-2 without join");
Thread t3 = new Thread(task, "Thread-3 without join");
Thread t4 = new Thread(task, "Thread-4 without join");
//Start thread 1
t1.start( );
//Start thread 2
t2.start( );
//Start thread 3
t3.start( );
```

```
//Start thread 4
t4.start( );
}
private static void ThreadJoin(MyTask task) throws
    InterruptedException
{
Thread t1 = new Thread(task, "Thread-1 using join");
Thread t2 = new Thread(task, "Thread-2 using join");
Thread t3 = new Thread(task, "Thread-3 using join");
Thread t4 = new Thread(task, "Thread-4 using join");
//Start thread 1
t1.start( );
t1.join( );
//Start thread 2
t2.start( );
t2.join( );
//Start thread 3
t3.start( );
t3.join( );
//Start thread 4
t4.start( );
t4.join( );
}
}
```

Output

1. Working with multiple threads using thread join:

```
Starting Thread-1 using join
Ending Thread-1 using join
Starting Thread-2 using join
Ending Thread-2 using join
Starting Thread-3 using join
Ending Thread-3 using join
Starting Thread-4 using join
Ending Thread-4 using join
```

2. Working with multiple threads WITHOUT thread join:

```
Starting Thread-1 without join
Starting Thread-2 without join
Starting Thread-3 without join
Starting Thread-4 without join
Ending Thread-1 without join
Ending Thread-2 without join
Ending Thread-3 without join
Ending Thread-4 without join
```

3.2.4 Thread Priorities, Synchronizing Threads, Inter-thread Communication

Q33. How do we set priorities for threads? Explain.

Model Paper-I, Q7(b)

OR

Discuss about "thread priorities" with examples.

Nov./Dec.-16(R13), Q6(a)

Answer :

Thread Priorities

Every thread is assigned a priority which specifies the amount of time used to access the C.P.U.The thread with highest priority thread contains less access time. While executing a lowest priority thread, if a highest priority is resumed back from its suspension, then the resumed thread can preempt the lowest priority thread. The priority of thread depends on the amount of time used to of access C.P.U and the highest potential access to C.P.U. The priority does not depends on the execution speed of thread. If a child thread is created, then the priority of it is similar to its parent thread. The thread which contain equal priority can access the C.P.U equally and the priority of thread can be altered.

Syntax

```
final void setPriority(int priority)
```

In the above syntax, final is a keyword, void is the return type, setPriority is the method name which can be used to set priority and priority is an integer value which can store the priority of thread. Any thread can contain one of the available priority constant and these constant can be defind by thread class. The various priority constants are as follows,

```
MIN_PRIORITY = 1
NORM_PRIORITY = 5
MAX_PRIORITY = 10
```

The priority of a thread can be acquired by using the following syntax,

```
final int get priority( )
```

Program

```
public class Demo extends Thread
{
public void run( )
{
    for(int i = 1; i < 10; i ++)
    {
        String str = Thread.currentThread( ).getName( );
        System.out.println(str + " : " + i);
    }
}
```

```
public static void main(String args[ ])
{
    Demo obj1 = new Demo( );
    Demo obj2 = new Demo( );
    Demo obj3 = new Demo( );
    obj1.setName("First Thread");
    obj2.setName("Second Thread");
    obj3.setName("Third Thread");
    obj1.setPriority(NORM_PRIORITY);
    obj2.setPriority(MAX_PRIORITY);
    obj3.setPriority(MIN_PRIORITY);
    obj1.start( );
    obj2.start( );
    obj3.start( );
    System.out.println("Priority of 1st thread:"+ obj1.getPriority( ));
    System.out.println("Priority of 2nd thread:"+ obj2.getPriority( ));
    }
}
```

In the above program, different priorities are set to three different threads. Each thread executes the loop for 10 times. e thread with the highest priority that is the second thread in this program will be executed first. The last 3 statements gets the ority associated with each thread.

What is synchronization? Explain with suitable example.

swer :

chronization

Synchronization can be defined as a process of enabling single thread to access shared resources. In multithread programming, chronization threads are essential. When threads are attempting to access the shared resources synchronization, data can be ted and result an incorrect output. To overcome this situation, synchronization threads can be utilized.

In multithreaded programming, thread synchronization is used in order to maintain the data consistency. This can be e using the 'synchronized' keywords. When a method is declared as synchronized, then the Java complier creates a monitor maphore and provides it to the threads object created for thread class which invokes the synchronized method initially. A monitor be defined as an object which can act as a mutex or mutual exclusive lock. The thread which can be entered into the monitor hold the lock and disables other thread to access the method. Once the execution of initial thread is complete, then the next iting thread can enter into the monitor. While executing the thread, when another thread attempts to enter the monitor, then it be suspended until the execution of the thread in monitor gets completed.

ample

```
class First
{
  synchronized void call(String str)
  {
    System.out.print("[ "+str);
    try
    {
      Thread.sleep(1000);
    }
```

```
            catch(InterruptedException e)
            {
                System.out.println("Interrupted");
            }
            System.out.println(" ] ");
        }
    }
class Second implements Runnable
{
    String str;
    First f;
    Thread t;
    public Second(First fir, String S)
    {
        f = fir;
        str = S;
        t = new Thread(this);
        t.start( );
    }
    public void run( )
    {
        f.call(str);
    }
}
class SynchDemo
{
    public static void main(String args[ ])
    {
        First f = new First( );
        Second s1= new Second(f,"SIA");
        Second s2= new Second(f, "GROUP");
        Second s3= new Second(f, "COMPANY");
        try
        {
            s1.t.join( );
            s2.t.join( );
            s3.t.join( );
        }
        catch(InterruptedException e)
        {
            System.out.println("Interrupted");
        }
    }
}
```

As shown in the program, by calling sleep( ), the call( ) method allows another thread to get executed. This will result in the mixed-up output of three message strings. To prevent this, method call( ) is declared as synchronized. When the sleep( ) is called by call( ), it makes other threads to wait until the call( ) resumes and returns to the calling method. This prevents other threads to access the shared resource at the same time.

Q35. Explain about inter-thread communications.

Answer : (Model Paper-II, Q7(a) | Nov./Dec.-17(R13), Q6(a))

Inter-thread Communication

Inter-thread communication can be defined as a means through which multiple threads can communicate with each other by exchanging messages. Any two threads can communicate before switching to others thread state or after switching to other thread state. Consider that the thread in active state can transfer a message to the thread in suspended state prior to switching to suspended state. In Java, inter-thread communication can be supported by using following methods.

(a) notify( )

(b) notifyall( )

(c) wait( ).

(a) notify( )

This method is used to resume the initial thread which is in sleep mode.

Syntax

final void notify( )

(b) notifyall( )

This method is used to resume all the threads that are in sleep mode. These threads can be executed depending on its priority.

Syntax

final void notifyall( )

wait( )

This method is used to send the invoking thread into sleep mode. The thread which is sent to sleep mode can be resumed with the help of notify( ) or notifyall ( ) method. In addition to this, user can set time to the thread by declaring the timer as an argument for wait( ) method. After the completion of timer, the thread can be resumed automatically.

Syntax

final void wait( )

These methods that are declared as final and therefore cannot be overriden and all these methods can throw InterruptedException.

Program

```
class InterThreadExample
{
    float r=0.0f;
    synchronized void result( )
    {
        System.out.println("The method called
                            to display circle area");
        if (r = =0.0)
        {
            System.out.println("Enter radius");
            try
            {
                wait( );
            }
            catch(Exception ex)
            {
            }
    }
    System.out.println("The area can be obtained
                        as"+3.14 * radius * radius);
}
synchronized void etvalue(float x)
{
        System.out.println("Enter radius...");
        r= x;
        System.out.println("Radius received");
        notify( );
    }
}
class Mainthread
{
public staic void main(String [ ] args)
{
    final InterThreadExample objThread =
                        newInterThreadExample( );
    newThread( )
    {
        public void run( )
        {
            objThread.result( );
        }
        }.start( );
        newThread( )
        {
            public void run( )
            {
                objThread setValue(10.5f);
            }
        } start( );
    }
}
```