	Hope Foundation's **Finolex Academy of Management and Technology, Ratnagiri** **Department of MCA**		
Subject name: Java Lab (SBL)			Subject Code: ITC304
Class	SEIT	Semester –III (CBCGS)	Academic year: 2022-23
Name of Student			**QUIZ Score :**
Roll No		Experiment No.	04

Title: **To study and implement the concepts of multithreading, exception handling and Input/Output streams.**

1. Lab objectives applicable:

LOB4 - To recognize usage of Exception Handling, Multithreading, Input Output streams in various applications.

2. Lab outcomes applicable:

LO4 - Construct robust and faster programmed solutions to problems using concept of Multithreading, exceptions and file handling

3. Learning Objectives:

1. To understand how exception handling and multithreading is achieved.

4. Practical applications of the assignment/experiment:

Concepts are used to improve robustness and error handling of applications,

5. Prerequisites:

1. Understanding of C programming.

6. Minimum Hardware Requirements:

1. P-IV PC with 2GB RAM, 500GB HDD, NIC

7. Software Requirements:

1. Windows / Linux operating systems, Java Developer Kit (1.8 or higher), Notepad++, Java Editors like Netbeans or Eclipse

8. Quiz Questions (if any): (Online Exam will be taken separately batch-wise, attach the certificate/ Marks obtained)

1. What is exception handling? What are its types?
2. Which are the two ways of creating user defined threads?

9. Experiment/Assignment Evaluation:

Sr. No.	**Parameters**	**Marks obtained**	**Out of**
1	Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used -		6
2	Lab Performance		2
3	Punctuality		2
Date of performance (DOP)		**Total marks obtained**		**10**

Signature of Faculty

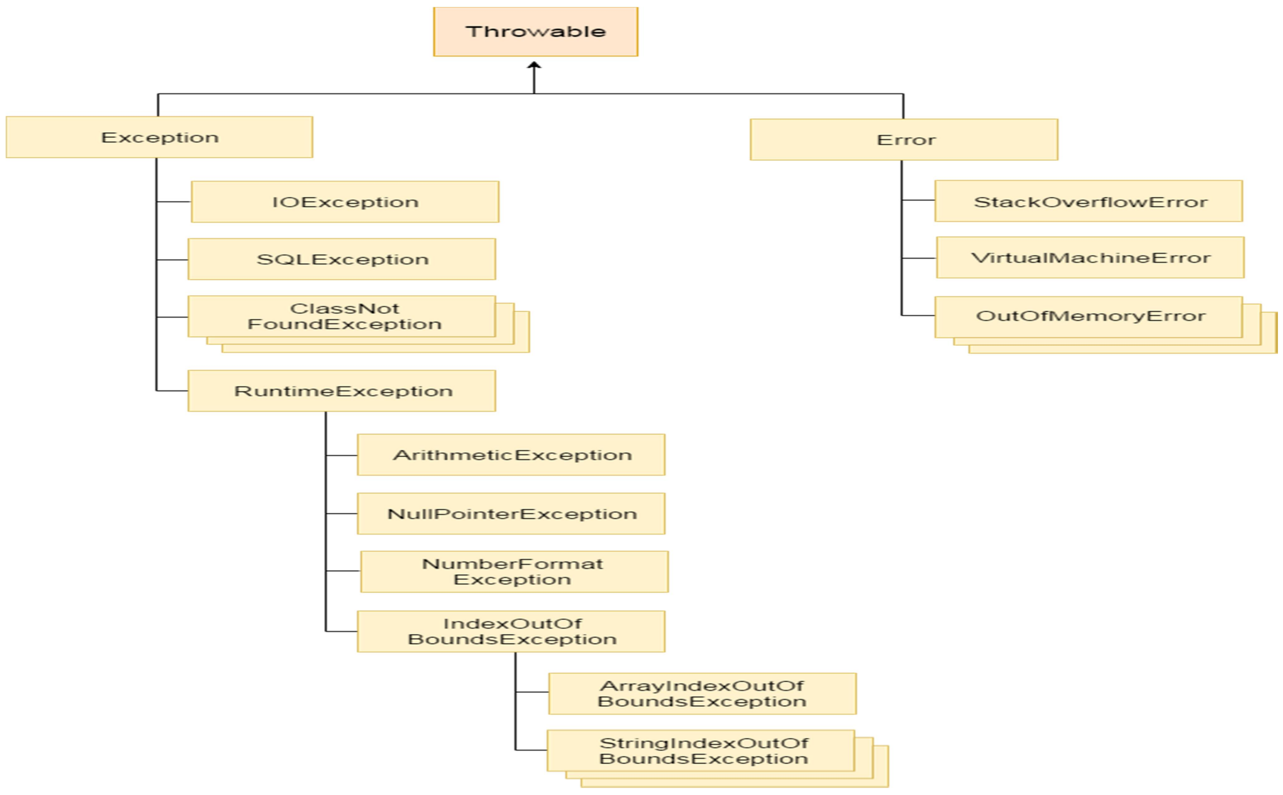
10. Theory:

Java- Checked and unchecked Exceptions

In Java, there two types of exceptions:

1) Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.
For example, consider the following Java program that opens file at locatiobn "C:\test\a.txt" and prints first three lines of it. The program doesn't compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*
2) Unchecked are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.
In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.

Hierarchy of Java Exception Classes



Java- Using try-catch blocks
Java try block
Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch or finally block.
Syntax of java try-catch
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
next →← prev
Java try block
Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch or finally block.

Syntax of java try-catch

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
try{
//code that may throw exception
}finally{}
```

Java catch block

- Java catch block is used to handle the Exception. It must be used after the try block only.
- You can use multiple catch block with a single try.

Java- Use of throw and throws keywords

Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
throw exception;
```

Let's see the example of throw IOException.

```
throw new IOException("sorry device error);
```

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

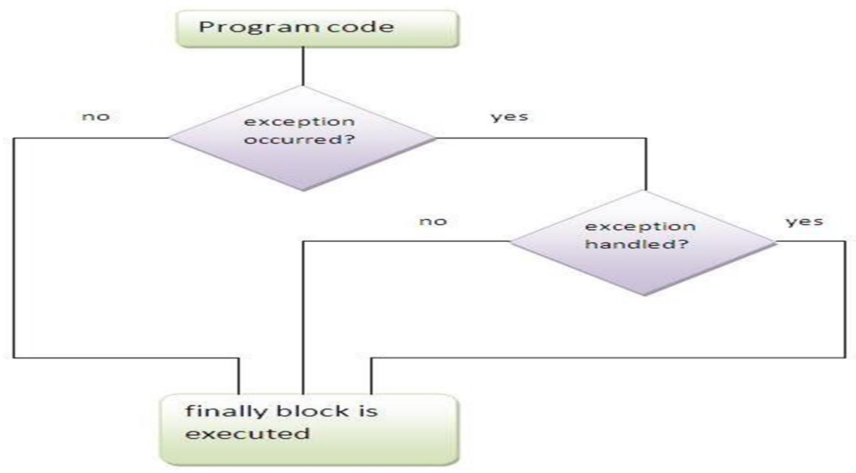
```
return_type method_name() throws exception_class_name{
//method code
}
```

Difference between throw and throws in Java

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare ar exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated witl throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions,e.g public void method()throws IOException,SQLException.

Java- Use of finally block:
Java finally block is a block that is used *to execute important code* such as closing connection, stream etc.
Java finally block is always executed whether exception is handled or not.
Java finally block follows try or catch block.



Java- Creating Threads

- **Create a Thread by Implementing a Runnable Interface**

If your class is intended to be executed as a thread then you can achieve this by implementing a **Runnable** interface. You will need to follow three basic steps −

Step 1

As a first step, you need to implement a run() method provided by a **Runnable** interface. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of the run() method −

```
public void run( )
```

Step 2

As a second step, you will instantiate a **Thread** object using the following constructor −

```
Thread(Runnable threadObj, String threadName);
```

Where, *threadObj* is an instance of a class that implements the **Runnable**interface and **threadName** is the name given to the new thread.

Step 3

Once a Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

```
void start();
```

- **Create a Thread by Extending a Thread Class**

The second way to create a thread is to create a new class that extends **Thread** class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1

You will need to override **run( )** method available in Thread class. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of run() method −

```
public void run( )
```

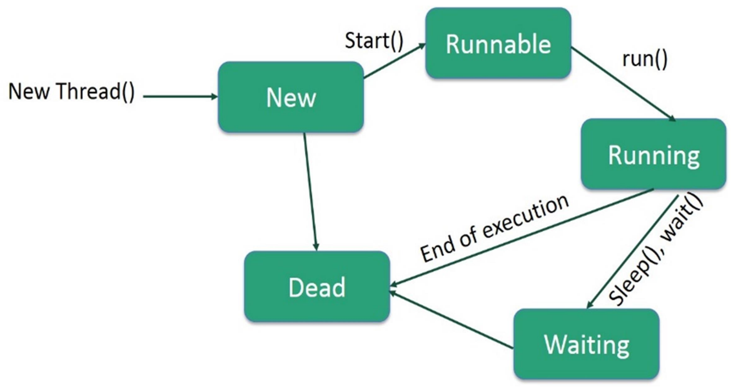
Step 2

Once Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

```
void start( );
```

Java- Lifecycle of Thread:

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



Following are the stages of the life cycle −

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.

- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Java- Use of *sleep(int)* method in threads:

Thread.sleep() method can be used to pause the execution of current thread for specified time in milliseconds. The argument value for milliseconds can't be negative, else it throws IllegalArgumentException. There is another overloaded method sleep(long millis, int nanos) that can be used to pause the execution of current thread for specified milliseconds and nanoseconds. The allowed nano second value is between 0 and 999999.

Java Thread Sleep important points

1. It always pause the current thread execution.
2. The actual time thread sleeps before waking up and start execution depends on system timers and schedulers. For a quiet system, the actual time for sleep is near to the specified sleep time but for a busy system it will be little bit more.
3. Thread sleep doesn't lose any monitors or locks current thread has acquired.
4. Any other thread can interrupt the current thread in sleep, in that case InterruptedException is thrown.

How Thread Sleep Works

Thread.sleep() interacts with the thread scheduler to put the current thread in wait state for specified period of time. Once the wait time is over, thread state is changed to runnable state and wait for the CPU for further execution. So the actual time that current thread sleep depends on the thread scheduler that is part of operating system.

Java- Thread synchronization:

When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unforeseen result due to concurrency issues. For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.

So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called **monitors**. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.

Java programming language provides a very handy way of creating threads and synchronizing their task by using **synchronized** blocks. You keep shared resources within this block.

11. Installation Steps / Performance Steps and Results –

Q1	Write java program where user will enter loginid and password as input. The password should be 8 digits containing one digit and one special symbol. If user enter valid password satisfying above criteria then show “Login Successful Message”. If user enter invalid Password then create InvalidPasswordException stating Please enter valid password of length 8 containing one digit and one special symbol.

Source code:

```
        import java.util.*;
class InvalidPasswordException extends Exception
{
        InvalidPasswordException(String s)
        {
                super(s);
        }
}
class PasswordCheckDemo
{
        public static boolean isPassCorrect(String s)
        {
                boolean digitFlag = false, symbolFlag = false, lengthFlag = false;

                char[] temp = s.toCharArray();
                if(s.length() >= 8)
                        lengthFlag = true;
                for(char c : temp)
                {
                        if(Character.isDigit(c))
                        {
                                digitFlag = true;
                                break;
                        }
                }
                for(char c : temp)
                {
                        if(c != 32 && (c < 48 || c > 57) && (c < 65 || c > 90 ) && ( c < 97 || c > 122))
                        {
                                symbolFlag = true;
                                break;
                        }
                }
                if(digitFlag && symbolFlag && lengthFlag)
                        return true;
```

```
		else
			return false;
	}
	public static void main(String [] args)
	{
		String name, password;
		Scanner sc = new Scanner(System.in);
		try
		{
			System.out.println("Enter User name");
			name = sc.next();
			System.out.println("Enter Password ");
			password = sc.next();
			if(isPassCorrect(password))
				System.out.println(" Login Successful ");
			else
				throw new InvalidPasswordException("Plese Enter correct password ");
		}
		catch (Exception e)
		{
			System.out.println(" Exception : "+e.getMessage());
		}
	}
}
```

Output:

```
	C:\SEM 3\THEORY>java PasswordCheckDemo
Enter User name
ayush
Enter Password
Ayush@0303
 Login Successful

C:\SEM 3\THEORY>java PasswordCheckDemo
Enter User name
Ayush
Enter Password
aysu
 Exception : Plese Enter correct password
```

Q2	Java Program to Create Account with 1000 Rs Minimum Balance, Deposit Amount, Withdraw Amount and Also Throws LessBalanceException. It has a Class Called LessBalanceException Which returns the Statement that Says WithDraw Amount(_Rs) is Not Valid. It has a Class Which Creates 2 Accounts, Both Account Deposite Money and One Account Tries to WithDraw more Money Which Generates a LessBalanceException Take Appropriate Action for the Same.

Source code:

```
class LessbalanceException extends Exception
{
    LessbalanceException(String s)
    {
        super(s);
    }
}
class Account
{
    int acc_no;
    int bal;
    Account(int an)
    {
        acc_no = an;
        bal = 1000;
        System.out.println("New Account Created With Account no. "+acc_no+" and balance =
"+bal);
    }
    void deposite(int amt)
    {
        bal = bal + amt;
        System.out.println("Account no. "+acc_no+" : Deposite of "+amt+" is successful! Current
Balance = "+bal);
        System.out.println();
    }
    void withdrwa(int amt)
    {
        try
        {
            if(bal < amt || (bal - amt <= 1000))
            {
                throw new LessbalanceException("Widraw  of "+amt+" is invalid");
            }
            bal = bal - amt;
```

```
                              System.out.println("Account no. "+acc_no+" Widrawl of amount : "+amt+" is
successful! Current Balance = "+bal);
                              System.out.println();

                    }         catch(Exception e)
                              {
                                        System.out.println("Exception : "+e.getMessage());
                              }
          }
          int getBal()
          {
                    return bal;
          }
}
class LessBalanceExceptionDemo
{
          public static void main(String[] args)
          {
                    Account a1 = new Account(1);
                    Account a2 = new Account(2);

                    a1.deposite(1000);
                    a1.getBal();
                    a1.withdrwa(500);

                    a2.deposite(1000);
                    a2.getBal();
                    a2.withdrwa(1000);
          }
}
```

Output:

```
          C:\SEM 3\THEORY>javac LessBalanceExceptionDemo.java

C:\SEM 3\THEORY>java LessBalanceExceptionDemo
New Account Created With Account no. 1 and balance = 1000
New Account Created With Account no. 2 and balance = 1000
Account no. 1 : Deposite of 1000 is successful! Current Balance = 2000

Account no. 1 Widrawl of amount : 500 is successful! Current Balance = 1500

Account no. 2 : Deposite of 1000 is successful! Current Balance = 2000
```

Exception : Widraw of 1000 is invalid

Q3	Create two threads such that one thread will print even number and another will print odd number in an ordered fashion.

Source code:

```
class Even extends Thread
{
        public void run()
        {
                for( int i=0;i<=10;i+=2)
                {
                        System.out.print(i+" ");
                        try
                        {
                                sleep(600);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }

                }
        }
}
class Odd extends Thread
{
        public void run()
        {
                for( int i=1;i<=10;i+=2)
                {

                        try
                        {
                                sleep(550);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                        System.out.print(i+" ");
                }
        }
}
class EvenorOdd
{
        public static void main(String [] args)
        {
                Even e1 = new Even();
                Odd o1 = new Odd();

                e1.start();
                o1.start();
        }
}
```

Output:

```
C:\SEM 3\THEORY>javac EvenorOdd.java

C:\SEM 3\THEORY>java EvenorOdd
0 1 2 3 4 5 6 7 8 9 10
```

12. Learning Outcomes Achieved

1. Students have understood exception handling mechanisms.
2. Students have understood how multithreading can improve performance of execution.

13. Conclusion:

1. **Applications of the studied technique in industry**
    a. Exception handling is an important practice in professional coding to improve robustness of applications.
2. **Engineering Relevance**
    a. Multithreading and exception handling can significantly improve performance metrics of any application.
3. **Skills Developed**
    a. Code robustness and error handling.

14. References:

[1] Stackoverflow.com. Stackoverflow.com is probably the most popular website in the programming world.
[2] Dzone.com.
[3] Java SE Technical Documentation. ...
[4] Java 2: The Complete Reference Book by Herbert Schildt
[5] The Java Language Specification Book by Bill Joy, Gilad Bracha, Guy L. Steele Jr., and James Gosling