6. EXCEPTION HANDLING

❖ Exception – Handling fundamentals

- **An exception is a problem that arises during the execution of a program**. An exception can occur for many different reasons, including the following:
  - A user has entered invalid data.
  - A file that needs to be opened cannot be found.
  - A network connection has been lost in the middle of communications, or the JVM has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.
- To understand how exception handling works in Java, you need to understand the three categories of exceptions:
  - **Checked exceptions:**
    - A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer.
    - For example, if a file is to be opened, but the file cannot be found, an exception occurs.
    - These exceptions cannot simply be ignored at the time of compilation.
  - **Runtime exceptions:**
    - A runtime exception is an exception that occurs that probably could have been avoided by the programmer.
    - As opposed to checked exceptions, runtime exceptions are ignored at the time of compliation.
  - **Errors:**
    - These are not exceptions at all, but problems that arise beyond the control of the user or the programmer.
    - Errors are typically ignored in your code because you can rarely do anything about an error.
    - For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

- Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.
  - Program statements that you want to check for exceptions are contained within a **try** block.
  - If an exception occurs within the **try** block, it is thrown.
  - Your code can catch this exception (using **catch**) and handle it in some normal manner.
  - System-generated exceptions are automatically thrown by the Java run-time system.To manually throw an exception, use the keyword **throw**.
  - Any exception that is thrown out of a method must be specified as such by a **throws** clause.
  - Any code that absolutely must be executed after a **try** block completes is put in a **finally** block.
- **Syntax:**

```
try
{
    // block of code to check for errors
}
catch (ExceptionType1 exOb)
{
    // exception handler for ExceptionType1
}
```

```
catch (ExceptionType2 exOb)
{
    // exception handler for ExceptionType2
}
// ...
finally
{
    // block of code to be executed after try block ends
}
```

❖ Exception types(Explain various exception techniques in java?...)

EXCEPTION TYPE	CAUSE OF EXCEPTION
Arithmetic Exception	Caused by math errors such as division by
Array Index Out of Bounds Exception	Caused by bad array indexes
Array Store Exception	Caused when a program tries to store the wrong type of data in an array
FileNotFoundException	Caused by an attempt to access a nonexistent file
IoException	Caused by general I/O failures, such as inability to read from a file
NullPointerException	Caused by referencing a null object
NumberFormatException	Caused when a conversion between strings and number fails
OutofMemoryException	Caused when there's not enough memory to allocate a new object
SecurityException	Caused when an applet tries to perform an action not allowed by the browser's security setting
StackOverFlowException	Caused when he system runs out of stack space
StringIndexOut of Bounds Exception	Caused when a program attempts to access a nonexistent character position in a string
ClassCastException	Thrown when attempting to cast a reference variable to a type that fails the IS-A test

Example:

1) ***int a=50/0;*** //ArithmeticException
2) ***String s="abc";***

   int i=Integer.parseInt(s); //NumberFormatException
3) ***String s=null;***

   System.out.println(s.length());//NullPointerException
4) ***int a[]=new int[5];***

   a[10]=50; //ArrayIndexOutOfBoundsException

❖ Using try and catch

➢ The try/catch statement encloses some code and is used to handle errors and exceptions that might occur in that code.

- With the help of try and catch we solve the run time error easy.
- The general syntax of the try/catch statement is as below:

```
try
{
        body-code
}
catch (exception-classname variable-name)
{
        handler-code
}
```

- **The try/catch statement has four parts.**
  - The **body-code** contains code that might throw the exception that we want to handle.
  - The **exception-classname** is the class name of the exception we want to handle.
  - The **variable-name** specifies a name for a variable that will hold the exception object if the exception occurs.
  - Finally, the **handler-code** contains the code to execute if the exception occurs. After the handler-code executes, execution of the thread continues after the try/catch statement.

- **<u>Advantage of using try & catch</u>:**
  - It allows the programmer to fix the error.
  - It prevents the programmer from automatically terminating.

- **Example:-**

```
class Trycatch
{
        public static void main(String[] args)
        {
                try
                {
                        int a,b,c;
                        a=12;
                        b=0;
                        c=a/b;
                        System.out.println(c);
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

❖ <u>Uncaught exceptions</u>

- This small program includes an expression that intentionally causes a divide-by-zero error:

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

```
        {
            int d = 0;
            int a = 42 / d;
        }
    }
```

- In above program error detects at the time of divide by zero so here it generates exception.
- This causes the execution of **Test** to stop.
- Any exception that is not caught by your program will ultimately be processed by the default handler provided by the Java run-time system.
- Here is the exception generated by default handler when this example is executed:
    - **Exception in thread "main" java.lang.ArithmeticException: / by zero at Test.main(Test.java:6)**
- Notice how the class name, **Test**; the method name, **main**; the filename, **Test.java**; and the line number, **6**, are all included in the simple stack trace.
- Also, notice that the type of exception thrown is a subclass of **Exception** called **ArithmeticException**, which more specifically describes what type of error happened.

Multiple catch clauses

- In some case, more than one exception could be raised by a single piece of code.
- In such situations two on more catch block can be specified each catching a different type of exception.
- The first catch block whose type matches is executed.
- After one catch statement executed, the others are bypassed & executed continues after try-catch block. At that time we use multiple catch
- **Example:-**

```
class Multicatch
{
        public static void main(String[] args)
        {
                try
                {
                         int a,b,c;
                        int d[]=new int[3];
                        a=12;
                        b=12;
                        c=a/b;
                        System.out.print(c);
                        d[4]=46;
                        System.out.print(d[4]);
                }
                catch (ArithmeticException e)
                {
                        System.out.print(e);
                }
                catch(ArrayIndexOutOfBoundsException e)
                {
                                System.out.print(e);
                }
        }
}
```

- When you use multiple **catch** statements, it is important to remember that exception subclasses must come before any of their superclasses.
- This is because a **catch** statement that uses a superclass will catch exceptions of that type plus any of its subclasses.
- Thus, a subclass would never be reached if it came after its superclass

Nested try statements

- In Java we can have nested **try** and **catch** blocks.
- It means that, a try block inside the another try block .
- If an inner try statement does not have a matching catch statement for a particular exception, the control is transferred to the next try statements catch handlers that are expected for a matching catch statement.
- This continues until one of the catch statements succeeds, or until all of the nested try statements are done in.
- If no one catch statements match, then the Java run-time system will handle the exception.

- **Syntax:-**

```
try
{
    try
    {
            // ...
    }
    catch (Exception1 e)
    {
            //statements to handle the exception
    }
}
catch (Exception2 e2)
{
    //statements to handle the exception
}
```

- **Example:-**

```
class NestedTry
{
        public static void main (String args [ ] )
        {
                try
                {
                                int a = Integer.parseInt(args [0]);
                                int b = Integer.parseInt(args [1]);
                                int c = 0;

                                try
                                {
                                        c = a / b;
                                        System.out.println(c);
                                }
                                catch (ArithmeticException e)
```

```
                    {
                            System.out.println("divide by zero");
                    }
            }
            catch (NumberFormatException e)
            {
                    System.out.println ("Incorrect argument type");
            }
        }
    }
```

❖ Throw, throws, finally

➢ Throw

- The throw keyword is used to explictily throw an exception.
- We can throw either checked or uncheked exception. The throw keyword is mainly used to throw custom exception
  - **Syntax:-**

    throw someThrowableObject;
- All the classes are children of the Throwable class, and all allow programs to differentiate among the various types of exceptions that can occur during the execution of a program.
- Used in program:

```
try
{
   // statements
   throw new UserDefinedException( );
   // statements
}
catch (UserDefinedException e)
{
   System.out.printIn ("User defined exception caught");
}
```

- The UserDefinedException is a class made specifically to handle an exception and it is encapsulating some specific kind of user defined behaviour.
- This exception is raised explicitly using the throw clause.
- At the same time the catch clause is there ready to catch the same exception object.

Example:-

```
class Throwclause
   {
      static void validate(int age)
       {
       if(age<18)
              throw new ArithmeticException("not valid");
      else
             System.out.println("welcome to vote");
     }
       public static void main(String args[])
       {
```

```
                        validate(13);
                        System.out.println("rest of the code...");
                }
          }
```

Output: Exception in thread main java.lang.ArithmeticException:not valid

➢ Throws

- The 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:-**

```
    void method_name() throws exception_class_name
    {
            ...
    }
```

- **Example:-**

```
class Throwsclause
{
    static void abc( ) throws IllegalAccessException
    {
                try
                {
                        throw new IllegalAccessException( ); // exception thrown
                }
                catch (IllegalAccessException e)// exception caught
                {
                        System.out.println ("catch block");
                        throw new IllegalAccessException( );
                        // exception thrown again
                }
    }
    public static void main (String args [ ])
    {
                try
                {
                        abc( );
                }
                catch (IllegalAccessException e)
                {
                        System.out.println ("Exception caught");
                }
    }
}
```

Diference between throw and throws

throw	throws
1)throw is used to explicitly throw an exception.	1) throws is used to declare an exception.
2)checked exception can not be propagated without throw.	2)checked exception can be propagated with throws.
3)throw is followed by an instance.	3) throws is followed by class.
4)throw is used within the method.	4) throws is used with the method signature.
5)You cannot throw multiple exception	5) You can declare multiple exception e.g. public void method() throws IOException, SQLException.

Finally

- Java support another statement known as finally statement that can be used to handle an exception that is not caught by any of the previous catch statements.
- Finally block can be used to handle any exception generated within a try block.
- It may be added immediately after the try block or after the last catch block shown as follow.
- When a finally block is defined, this is guaranteed to execute.
- As a result, we can use it to perform certain house-keeping operations such as closing file and releasing system resource.
- **Syntax:**

```
try
{
   // statements
}
catch (<exception> obj)
{
   // statements
}
finally
{
   //statements
}
```

- **Example:-**

```
class Finally
{
        public static void main(String[] args)
        {
                try
                {
                        int a,b,c;
                        a=2;
                        b=0;
                        c=a/b;
                        System.out.println(c);
                }
                catch (Exception e)
```

```
                    {
                            System.out.println(e);
                    }
                    finally
                    {
                            System.out.print("In the finally block");
                    }
            }
    }
```

❖ Creating your own exception sub classes

- You can create your own exception class by defining a subclass of Exception.
- The Exception class does not define any methods of its own. It inherits methods provided by Throwable.
- All exceptions have the methods defined by Throwable available to them.
- They are shown in the following list.
    - **Throwable fillInStackTrace( )**
        - Returns a Throwable object that contains a completed stack trace.
    - **Throwable getCause( )**
        - Returns the exception that underlies the current exception.
    - **String getLocalizedMessage( )**
        - Returns a localized description.
    - **String getMessage( )**
        - Returns a description of the exception.
    - **StackTraceElement[ ] getStackTrace( )**
        - Returns an array that contains the stack trace.
    - **Throwable initCause(Throwable causeExc)**
        - Associates causeExc with the invoking exception as a cause of the invoking exception.
    - **void printStackTrace( )**
        - Displays the stack trace.
    - **void printStackTrace(PrintStream stream**)
        - Sends the stack trace to the stream.
    - **void printStackTrace(PrintWriter stream)**
        - Sends the stack trace to the stream.
    - **void setStackTrace(StackTraceElement elements[ ])**
        - Sets the stack trace to the elements passed in elements.
    - **String toString( )**
        - Returns a String object containing a description of the exception.

- **Example:-**

```
    class MyException extends Exception
    {
            private int b;
            MyException(int a)
            {
                    b = a;
            }
            public String toString()
```

```
		{
			return "MyException[" + b + "]";
		}
	}
	public class Main
	{
		static void display(int a) throws MyException
		{
			System.out.println(a);
			if (a > 10)
				throw new MyException(a);
				System.out.println("Normal exit");
		}
		public static void main(String args[])
		{
			try
			{
				display(2);
				display(20);
			}
			catch (MyException e)
			{
				System.out.println("Caught " + e);
			}
		}
	}
```