# CHAPTER-10

# MANAGING ERRORS AND EXCEPTIONS

## Introduction:

Errors are mistakes that can make a program go wrong. An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash.

It is therefore important to detect and manage properly all the possible error conditions in the program so that the program will not terminate or crash during execution.

## Types of errors Errors :

- Compile-time errors
- Run-time errors

## Compile-time errors

- All syntax errors will be detected and displayed by the Java compiler and therefore these errors are known as compile-time errors.
- Whenever the compiler displays an error, it will not create the .class file. It is therefore necessary that we fix all the errors before we can successfully compile and run the program.
- The Java compiler does a nice job of telling us where the errors are in the program. :

**For example,** if we have missed the semicolon at the end of print statement the following message will be displayed in the screen

```
Error1.java :7: ';' expected
System.out.println ("Hello Java!")
^
1 error
```

## Run-Time Errors

Sometimes, a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow

## Common run-time errors are:

- Dividing an integer by zero
- Accessing an element that is out of the bounds of an array
- Trying to store a value into an array of an incompatible class or type
- Trying to cast an instance of a class to one of its subclasses
- Passing a parameter that is not in a valid range or value for a method
- Trying to illegally change the state of a thread
- Attempting to use a negative size for an array
- Using a null object reference as a legitimate object reference to access a method or a variable
- Converting invalid string to a number
- Accessing a character that is out of bounds of a string

# Exceptions

- An exception is a condition that is caused by a run-time error in the program.
- When the Java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it(i.e., informs us that an error has occurred).
- If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program

## Common Java Exceptions:

| Exception Type | Cause of Exception |
|---|---|
| ArithmeticException | Caused by math errors such as division by zero |
| ArrayIndexOutOfBoundsException | Caused by bad array indexes |
| ArrayStoreException | 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 |

| Exception Type | Cause of Exception |
|---|---|
| `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 the system runs out of stack space |
| `StringIndexOutOfBoundsException` | Caused when a program attempts to access a nonexistent character position in a string |

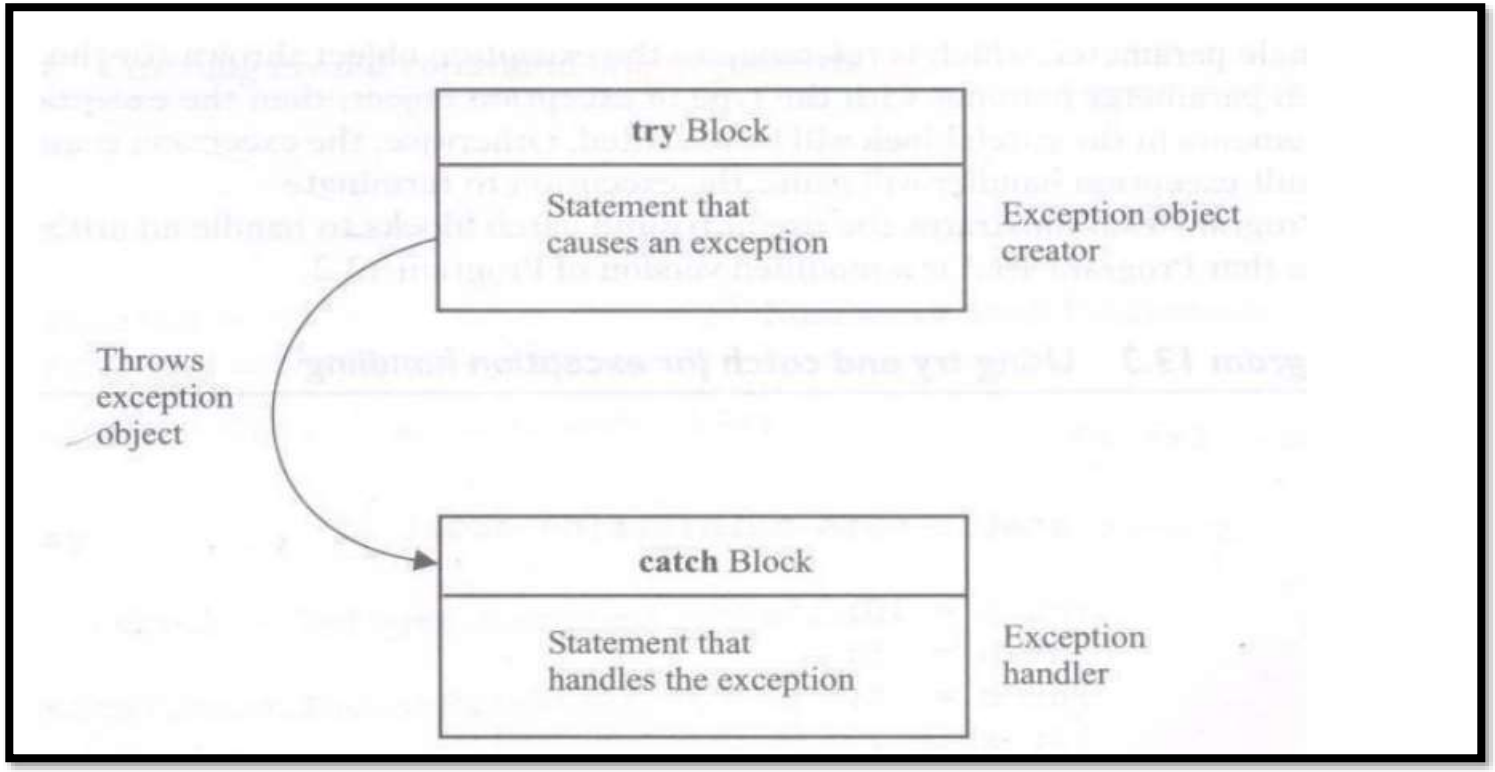
# Java provides five keywords that are used to handle the exception.

| Keyword | Description |
| --- | --- |
| try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

## Syntax of exception handling code

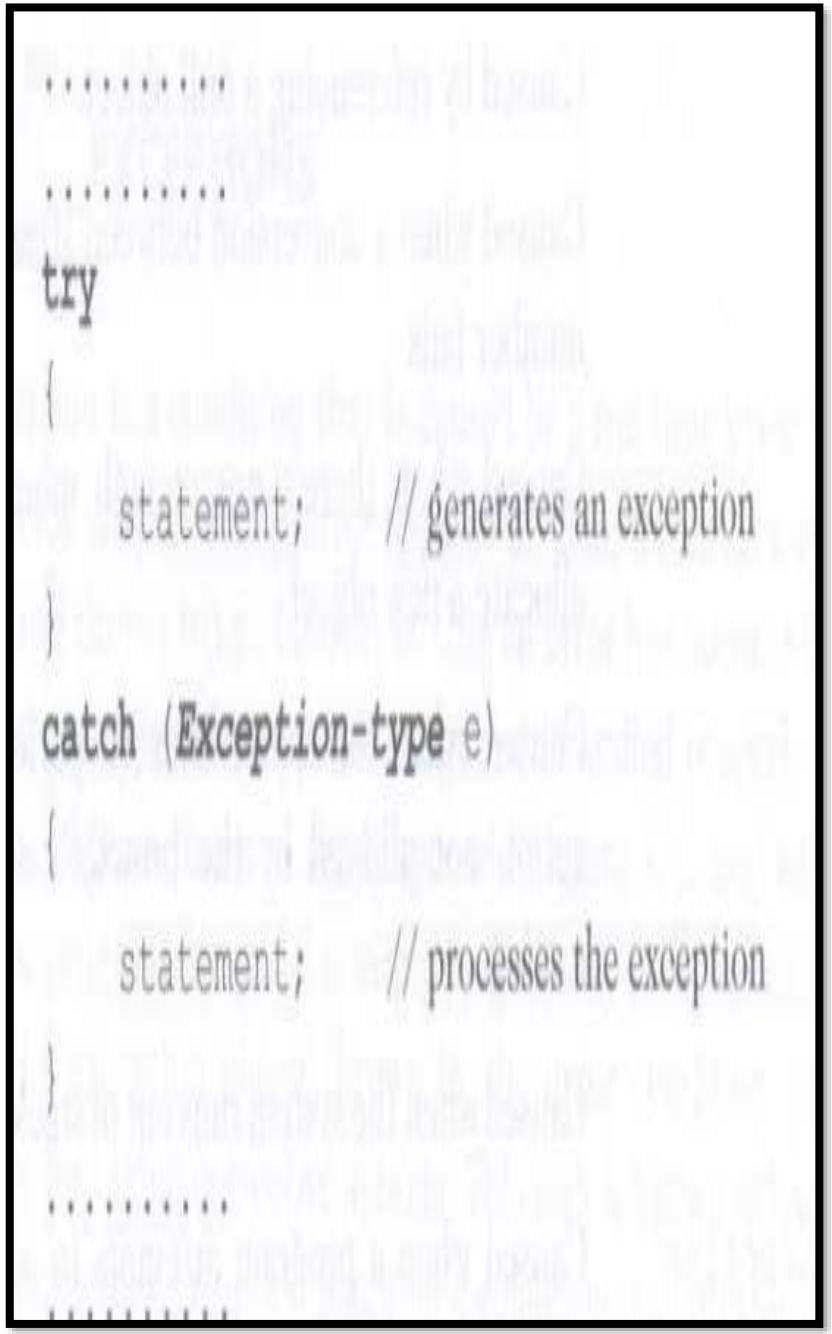
The basic concepts of exception handling are throwing an exception and catching it.

Java treats the multiple catch statements like cases in a switch statement.




- Java uses a keyword try to preface a block of code that is likely to cause an error condition and "throw" an exception.
- A catch block defined by the keyword catch "catches" the exception "thrown" by the try block and handles it appropriately
- The catch block is added immediately after the try block.

- The try block can have one or more statements that could generate an exception.
- If anyone statement generates an exception, the remaining statements in the block are skipped and execution jumps to the catch block that is placed next to the try block.
- The catch block too can have one or more statements that are necessary to process the exception. Remember that (every try statement should be followed by at least one catch statement; otherwise compilation error will occur. ).
- catch statement works like a method definition. The catch statement is passed a single parameter, which is reference to the exception object thrown (by the try block).
- If the catch parameter matches with the type of exception object, then the exception is caught and statements in the catch block will be executed. Otherwise, the exception is not caught and the default exception handler will cause the execution to terminate.



**Example:**

```
class  Error3
{
    public static void main(String args[ ])
    {
        int a = 10;
        int b =  5;
        int c =  5;
        int x, y ;

        try
        {
            x  =  a / (b-c);     // Exception here
        }

        catch (ArithmeticException e)
        {
            System.out.println("Division by zero");
        }

        y  =  a / (b+c);
        System.out.println("y  =  " + y);
    }
}
```

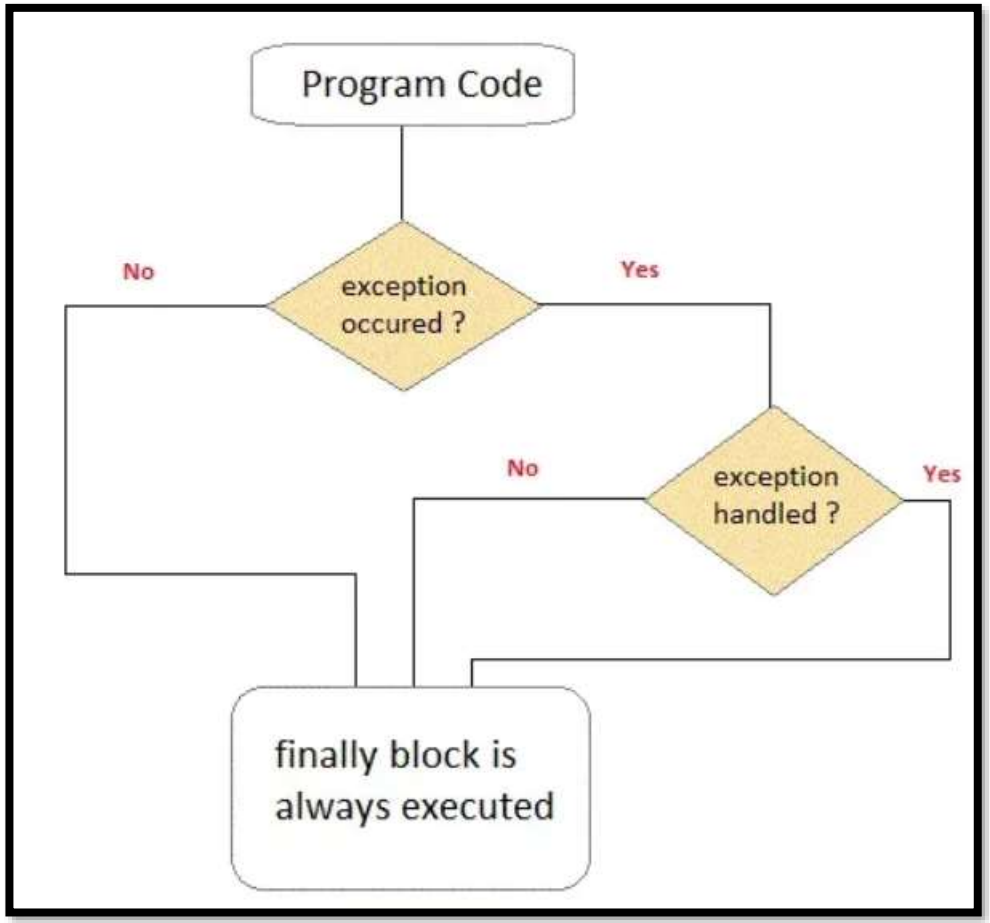
## Multiple catch statements

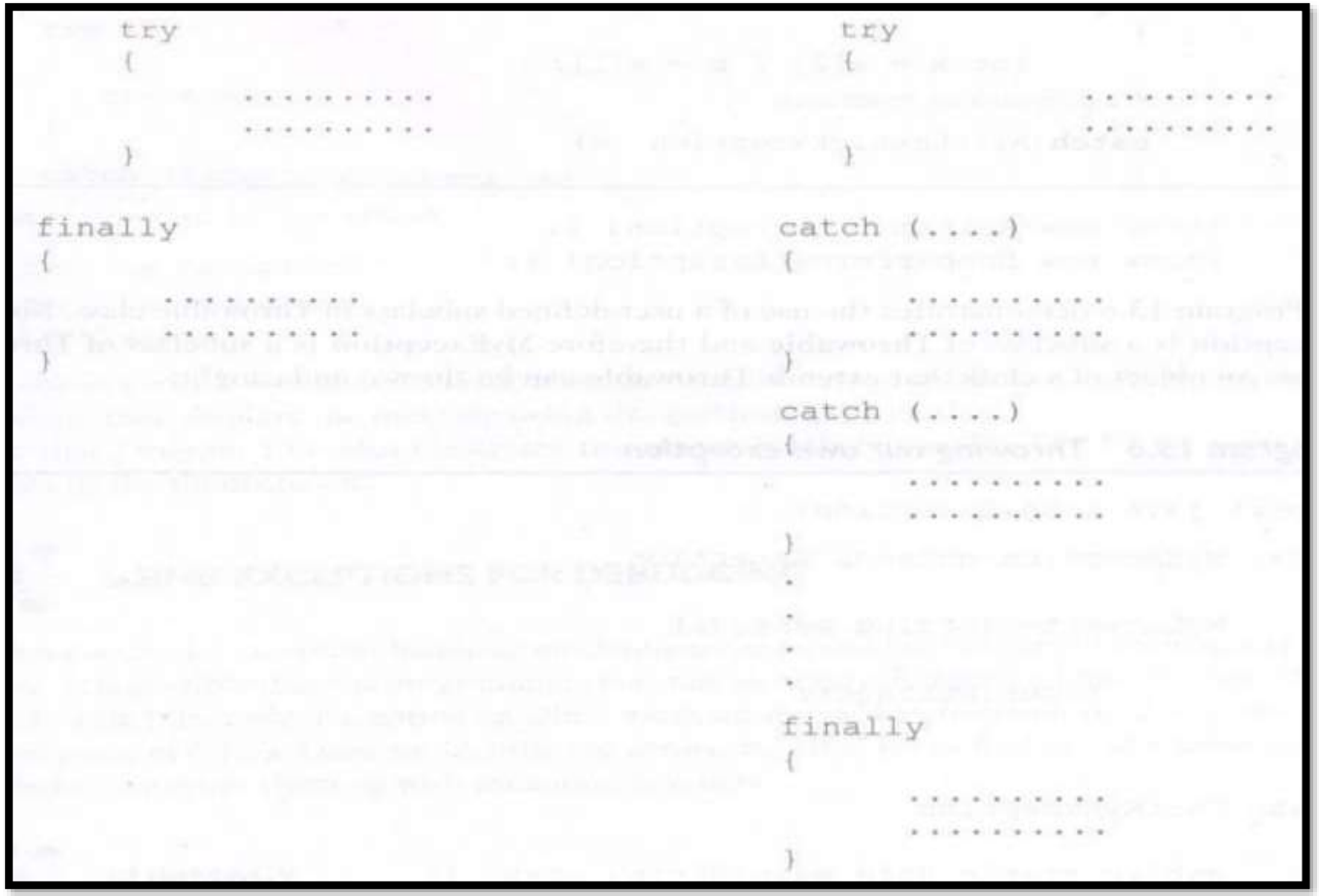
- It is possible to have more than one catch statement in the catch block.
- Java treats the multiple catch statements like cases in a switch statement

```
try
{
    statement ;                          // generates an exception
}
catch (Exception-Type-1  e)
{
    statement;                    // processes exception type 1
}
catch (Exception-Type-2  e)
{
    statement;                    // processes exception type 2
}
.
.
.
catch (Exception-Type-N  e)
{
    statement ;                   // processes exception type N
}
..........
..........
```

## Using finally Statement:

- **Java finally block** is a block used to execute important code such as closing the connection, etc.
- Java finally block is always **executed whether an exception is handled or not**. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.
- The finally block **follows the try-catch block**.



## Throwing our own exceptions

We can throw our own exceptions by using the **keyword throw** .

```
throw new Throwable_subclass;
```

## Examples:

throw new ArithmeticException( );

throw new NumberFormatException( );

- Exception is a subclass of Throwable and therefore MyException is a subclass of Throwable class.
- An object of a class that extends Throwable can be thrown and caught

**Example:**

```
import java.lang.Exception;
class MyException extends Exception
{
        MyException(String message)
        {
                super(message);
        }
}
class TestMyException
{
        public static void main(String args[ ])
        {
                int x = 5, y = 1000;
                try
                {
                    float z = (float) x / (float) y ;
                    if(z < 0.01)
                    {
                        throw new MyException("Number is too smal
                    }
                }
                catch (MyException    e)
                {
                    System.out.println("Caught my exception");
                    System.out.println(e.getMessage( ) );
                }
                finally
                {
                    System.out.println("I am always here");
                }
        }
 }
```