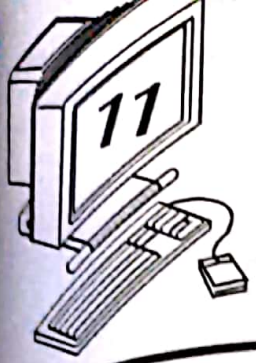

Exceptions Handling

11.1 Introduction

Errors occurring at runtime i.e. after clean compilation is called **exceptions.**

Exception handling allows us to manage runtime errors in a systematic manner.

Some examples of runtime errors or **exception** are :

- Division by zero
- array index out-of-range
- arithmetic overflow
- unexpected arguments
- file not found etc.

The purpose of the exception handling mechanism is to provide means to detect and report an **exceptional circumstance** so that appropriate action can be taken.

In this chapter, we will discuss some of the exception handling techniques of Java.

Note :

The **exception handling** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

11.2 Need for Exception Handling

The advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling.

Example 11.1

Consider the following scenario :

```
statement 1;
statement 2;
statement 3;        // exception occurs
statement 4;
statement 5;
statement 6;
statement 7;
statement 8;
```

Suppose there is 8 statements in a program and an exception occured at statement 3, rest of the code will not be executed i.e. statement 4 to 8 will not run. If we perform **exception handling**, rest of the code (i.e. statement 4 to 8) will be executed. That is why we use exception handling in java.

11.3 Exceptions and their Types

In java, all exceptions are represented by **classes**. The hierarchy of exception classes commence from **Throwable** class which is the base class for an entire family of exception classes, declared in **java.lang packages** as **java.lang.Throwable**. Exceptions are thrown if any kind of unusual condition occurs that can be caught. Sometimes it also happens that the exception could not be caught and the program may get terminated.

The hierarchy of java exception classes is shown in figure 11.1.

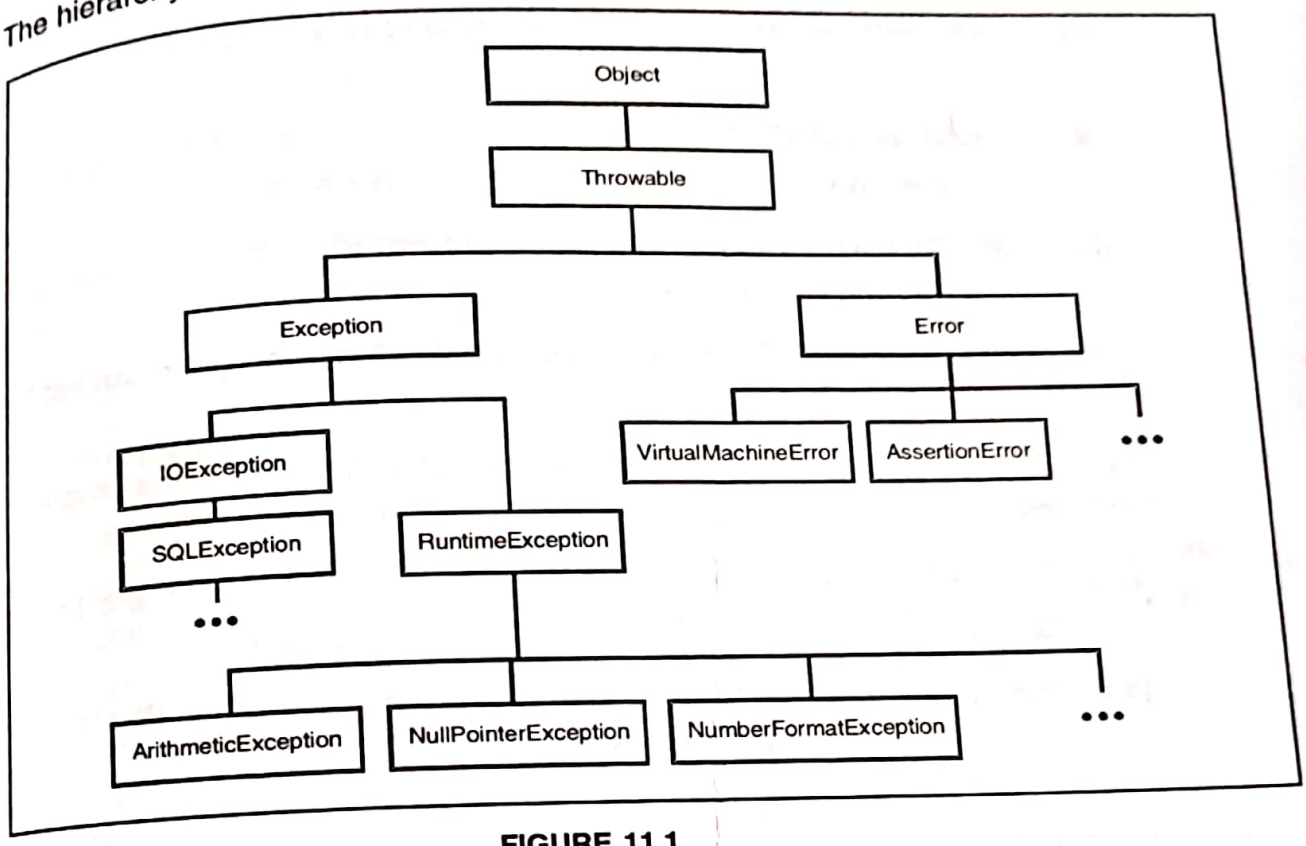

FIGURE 11.1

11.3.1 Types of Exception

There are three types of exceptions :

1. Checked Exception
2. Unchecked Exception
3. Error

11.3.1.1 Checked Exception

The classes that extend **Throwable** class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

11.3.1.2 Unchecked Exception

The classes that extend **RuntimeException** are known as unchecked exceptions e.g. ArithmeticException, NullPointerException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

The following are the common unchecked exceptions classes defined in java.lang.

(i) **ArithmeticException :** It is thrown, when a program attempts to perform divide by zero.

(ii) **ArrayIndexOutOfBoundsException :** It is thrown when a program attempts to access an index of an array that does not exist.

(iii) **NegativeArraySizeException :** It is thrown when a program attempt to access an array that has negative size.

(iv) **ClassNotFoundException :** It is thrown when a program can not find a class it depends at runtime.

(v) **StringIndexOutOfBoundsException :** It is thrown when a program attempts to access a character at a non-existent index in a String.

(vi) **NullPointerException :** It is thrown when JVM attempts to perform an operation on an Object that points to null or no data.

(vii) **NumerFormatException :** It is thrown when a program is attempting to convert a string to a numerical data type.

11.3.1.3 Error

Errors are typically ignored in your code because you can rarely do anything about an error. Errors are not exceptions at all, but problems that arise beyond the control of the user or the programmer.

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

1.4 Exception Handling

Java exception handling is managed via five keywords : **try, catch, throw, throws** and **finally.**

A block of statements in which an exception can occur must be prefixed by the keyword **try**. This block of statements is known as **try block.** When an exception is detected, then the runtime informs the application that an error has occurred. This process of informing is called **throwing** an exception.

A **catch block** defined by the keyword **catch,** catches the exception thrown in the **try block** and handles it appropriately.

The **catch** block that catches an exception must immediately follows the **try** block that throws the exception.

The **finally** block is used to handle an exception that is not caught by any of the previous catch statement. The **finally block** is added immediately after the try block or after the last catch block.

To manually throw an exception, use the keyword **throw**. In some cases, an exception that is thrown out of a method must be specified as such by a **throws** clause.