UNIT 2: EXCEPTION HANDLING

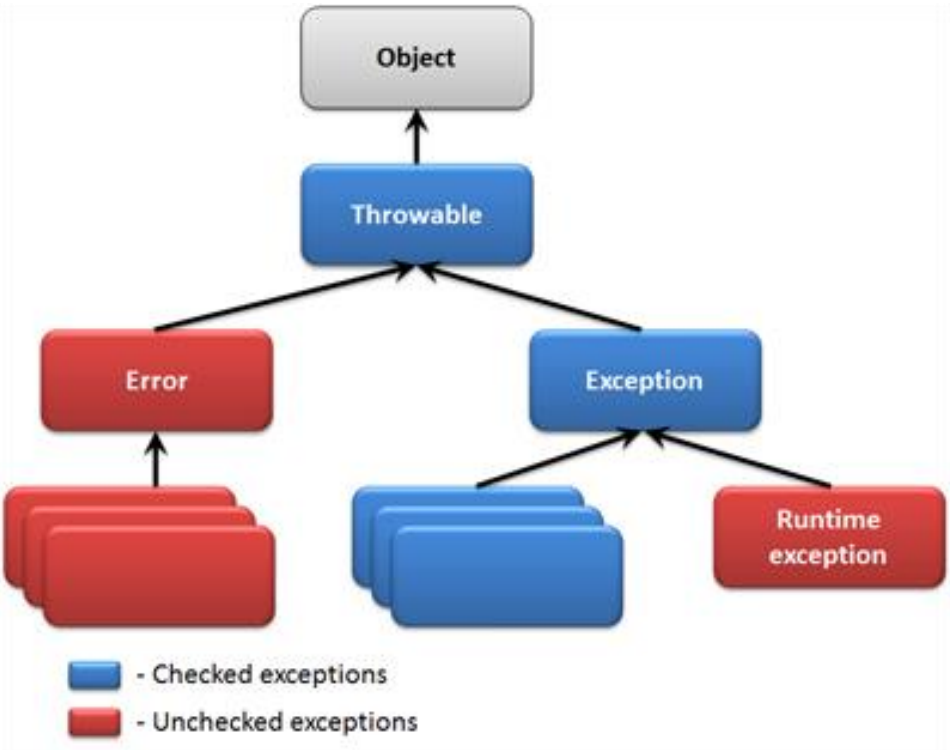
Unit Structure

2.0 Learning Objectives

After learning this unit, you will be able to:

- Explain the types of exception and uncaught exception
- Use try and catch blocks, multiple catch statements
- Illustrate methods defined by exception and throwable
- Discuss defined exceptions
- Describe throws/ throw keyword
- State finally keyword and nested try statements

2.1 Introduction



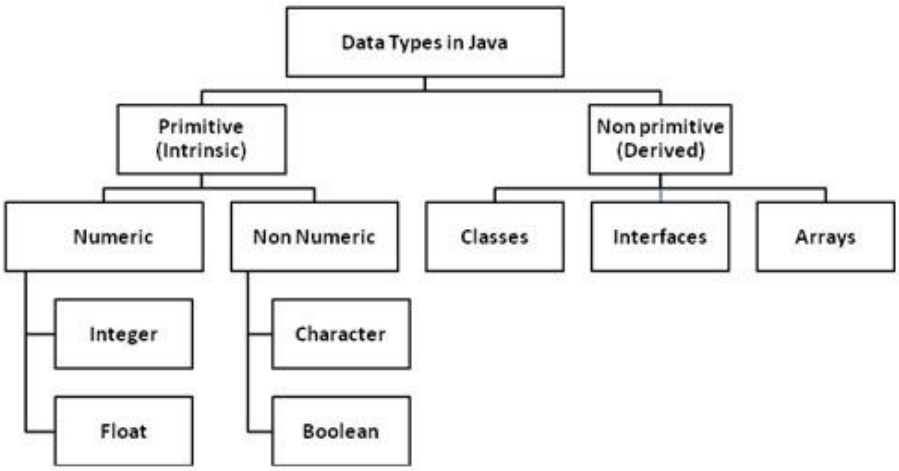
A runtime error that arises during the execution of a program is called an exception. Various languages don't support handling exceptions and thus, the errors are to be checked for and taken care of manually through error codes etc. This approach is quite cumbersome and troublesome.

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.

Such exceptions can often be caused due to programmer or user errors and sometimes due to other physical resources failing in some manner.

2.2 Types of Exceptions



Java provides several predefined Exception classes in the package java.lang. In order to understand the concept of exception handling, you have to understand the following categories of exceptions:

1. Checked exceptions
2. Unchecked exceptions
3. Errors

The checked exceptions must be caught or re-thrown. The unchecked exceptions do not have to be caught.

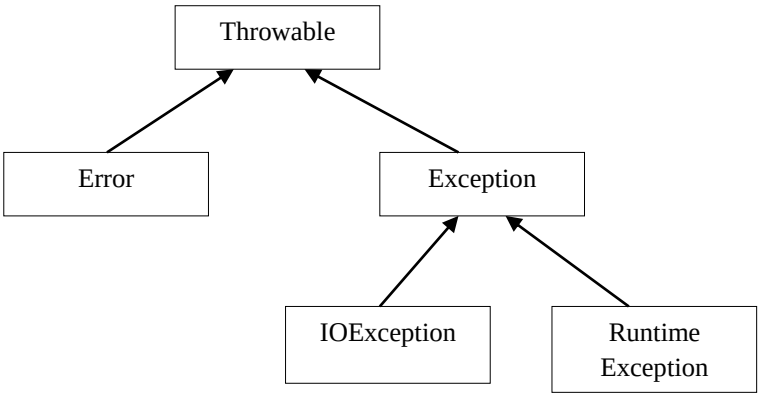
1. **Checked exceptions:** This is basically a user error that the programmer cannot foresee. For example, if a file is to be opened but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
2. **Unchecked or Runtime exceptions:** This exception could have probably been avoided by the programmer. Runtime errors are ignored during compilation as opposed to checked exceptions.
3. **Errors:** These are not exceptions at all but problems that arise beyond the control of the user or the programmer. Usually since the programmer can rarely do anything about an error it is ignored. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception Hierarchy:

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Normally errors cannot be trapped from the Java program. Such conditions arise in the case of severe failures which java programs do not handle. Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of Memory. Normally programs cannot recover from errors.

The Exception class has two main subclasses: IOException class and Runtime Exception Class.



Common Exceptions:

In java it is possible to define two categories of Exceptions and Errors.

- JVM Exceptions: These errors are either exclusively or logically thrown by the JVM. Examples: NullPointerException, ArrayIndexOutOfBounds Exception, ClassCastException
- Programmatic exceptions. These exceptions are thrown explicitly by the application or the API programmers Examples: IllegalArgumentException, IllegalStateException

Check your progress 1

1. Name the subclasses of exception class.

.....

.....

.....

.....

.....

.....

.....

2.3 Uncaught Exception

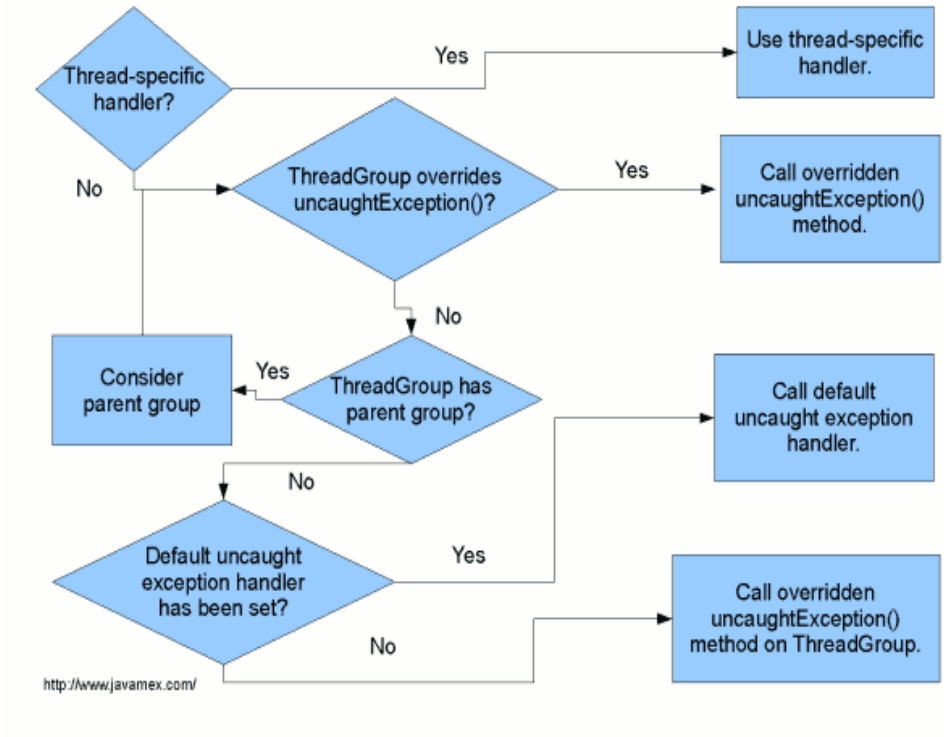
When the exceptions are not caught in a try/catch block, then what you often see in practice is Java prints the exception stack trace and then terminate your program. "Java actually handles uncaught exceptions according to the thread in which they occur. When an uncaught exception occurs in a particular thread, Java looks for what is called an uncaught exception handler, actually an implementation of the interface UncaughtExceptionHandler". The latter interface has a method handleException (), which the implementer overrides to take appropriate action, such as printing the stack trace to the console.

The specific procedure is as follows. When an uncaught exception occurs, the JVM does the following:

- It calls a special private method, dispatchUncaughtException(), on the Thread class in which the exception occurs
- It then terminates the thread in which the exception occurred

The dispatch Uncaught Exception method, in turn, calls the thread's get Uncaught Exception Handler () method to find out the appropriate uncaught exception handler to use. Normally, this will actually be the thread's parent Thread Group, whose handle Exception () method by default will print the stack trace.

Thus, the full process used to determine which uncaught exception handler is called is shown in Figure below:



Process by which Java decides on which uncaught exception handler to call for a given thread

Check your progress 2

1. Explain an uncaught exception.
2. Write a program for an uncaught exception handler.

.....

.....

.....

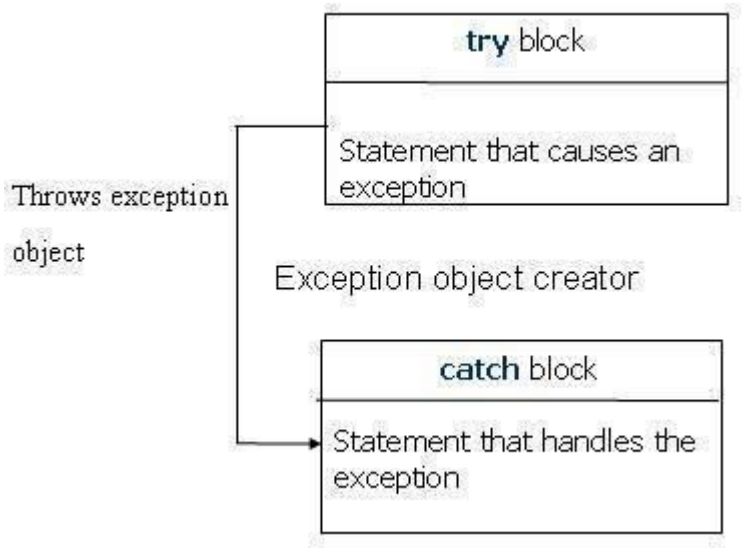
.....

.....

.....

.....

2.4 Using Try and Catch Block



When an exception arises, an object representing that exception is created and thrown in the method that caused the error. This mechanism in Java creates an object which describes an exception condition.

When an exception condition arises, an object representing that exception is created and thrown in the method which caused the error. When an error occurs within a method, the method creates an object and hands it off to the runtime system.

This exception object is called an exception object which contains information about the error along with its type and the state of the program when the error occurred. This method of creating an exception object and handing it to the runtime system is called throwing an exception.

Java Exception handling is managed using five keywords-

a. try

b. catch

c. throw

d. throws

e. finally

The general form of the Exception Handling is

```
try
{
//do something that might cause an exception
}
catch (ExceptionType variable)
{
//handle the exception
}
finally
{
//always execute these statements
}
```

The program statements which you want to monitor for exceptions are written within try block. If an exception occurs within the try block (when exception is thrown), it has to be handled in some manner in catch block.

Any code which exactly has to be executed, after exiting from try clock is put in finally block. The catch block is only executed if a particular exception occurs. Whereas, the finally block is always executed, irrespective of exception occurs or not. A brief description of these keywords is given below:

Try:

The code that could generate errors is put in try block. The statements are executed unless an exception occurs. If an exception is thrown, java breaks out of the try block by skipping the rest of the statements and searches for a respective matching catch statement.

If an exception does not occur, java executes all the statements within that block.

Catch:

A catch statement is included immediately following the try block; it specifies the exception type that has to be caught.

Finally:

It creates a block of code following the try catch block. It will execute whether or not an exception is thrown. Each try statement should have at least one catch or finally clause.

Check your progress 3

1. List the keywords that manage Java exception handling.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

2.5 Using Multiple Catch Statements

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
{
  //Protected code
}catch(ExceptionType1 e1)
{
  //Catch block
}catch(ExceptionType2 e2)
{
  //Catch block
}catch(ExceptionType3 e3)
{
  //Catch block
}
```

The previous statements demonstrate three catch blocks but you can have any number of them after a single try. The exception is thrown to the first catch block in the list if it occurs in the protected code. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement.

This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Catching Exceptions:

By using a combination of the try and catch keywords a method catches an exception. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code and the syntax for using try/catch looks like the following:

```
try
{
  //Protected code
}catch(ExceptionName e1)
{
  //Catch block
}
```

A catch statement comprises of declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or a block) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Check your progress 4

1. Write the syntax for multiple catch blocks.
2. Explain the execution of multiple catch statements.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

2.6 Using Methods Defined by Exception and Throwable

Following is the list of important methods available in the Throwable class:

SN	Methods with Description
1	"public String get Message() Returns a detailed message about the exception that has occurred. This message is initialised in the Throwable constructor.
2	Public Throwable get Cause() Returns the cause of the exception as represented by a Throwable object.
3	public String to String() Returns the name of the class concatenated with the result of get Message()
4	public void print Stack Trace() Prints the result of to String() along with the stack trace to System. err, the error output stream.
5	Public Stack Trace Element [] get Stack Trace() Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack and the last element in the array represents the method at the bottom of the call stack.
6	Public Throwable fill In Stack Trace() Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace".

Check your progress 5

1. Write a note on public Throwable get Cause().
2. Write a program that throws an exception.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

2.7 User Defined Exceptions

"You can create your own exceptions in Java. Keep the following points in mind while writing your own exception classes:

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the Run time Exception class."

We can define our own Exception class as below:

```
class My Exception extends Exception{
}
```

You just need to extend the Exception class to create your own Exception class. These are considered to be checked exceptions. The following Insufficient Funds Exception class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

Example:

```
// File Name InsufficientFundsException.java

import java.io.*

public class InsufficientFundsException extends Exception

{

 private double amount

public InsufficientFundsException(double amount)

{

this.amount = amount

}

public double getAmount()

{

return amount

}

}
```

To demonstrate using our user-defined exception, the following CheckingAccount class contains a withdraw () method that throws an InsufficientFundsException.

```
// File Name CheckingAccount.java

import java.io.*;

public class ChkAccount

{

private double balance;

private intAcNumber;
```

```
public ChkAccount(intAcNumber)
{
this.AcNumber = AcNumber;
}
public void deposit(double amount)
{
balance += amount;
}
public void withdraw(double amount) throwsInSufficientFundsException
{
if(amount <= balance)
{
balance -= amount;
}
else
{
double needs = amount - balance
throw new InSufficientFundsException(needs)
}
}
public double getBalance()
{
return balance;
}
public intgetAcNumber()
{
return AcNumber;
```

```
}
}
```

The following BankDemo program demonstrates invoking the deposit () and withdraw () methods of CheckingAccount.

```
// File Name BankDemo.java
public class BankDemo
{
 public static void main(String [] args)
{
ChkAccount c = new ChkAccount(101);
System.out.println ("Depositing Rs500...");
c.deposit(500.00);
try
{
System.out.println("\nWithdrawing Rs100...")
c.withdraw(100.00);
System.out.println("\nWithdrawing Rs600...")
c.withdraw(600.00)
}catch(InsufficientFundsException e)
{
System.out.println("Sorry but you are short Rs".
                        + e.getBalance());
e.printStackTrace();
}
}
}
classInSufficientFundsException extends Exception
```

```
{
InSufficientFundsException( double needs)
{
System.out.println("Insufficient fund"+ needs);
}
}
```

Check your progress 6

1. What points should be kept in mind when writing your own exception classes?

.....

.....

.....

.....

.....

.....

.....

.....

2.8 Using Throws/throw Keywords

Before catching an exception it is must to be thrown first. This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception. A single argument is required by the throw statement i.e. a throwable object. Throwable objects are instances of any subclass of the Throwable class.

throw new VeryFastException()

The reference should be of type Throwable or one of its subclasses.

For instance the example below shows how to throw an exception. Here we are trying to divide a number by zero so we have thrown an exception here as

"throw new MyException ("can't be divided by zero")"

```
classMyException extends Exception {
public MyException(String msg){
super(msg);
}
}

public class Test {


static void  divide(intfirst,int second) throws MyException{
  if(second==0)
  throw new MyException("can't be divided by zero") }

 public static void main(String[] args) {
  try {
divide(4,0);
  }
catch (MyExceptionexc) {
exc.printStackTrace();
  }
  }//main
}//Test
```

Output

```
C:\Computer\vinod\Exception>javac Test.java


C:\Computer\vinod\Exception>java Test
MyException: can't be divided by zero
at Test.divide (Test.java:10)
at Test.main(Test.java:15)
```

The method must declare by using the throws keyword if it does not handle a checked exception. The throws keyword appears at the end of a method's signature.

By using the throw keyword you can throw an either newly instantiated exception or an exception you just caught. The difference in throws and throw keywords should be understood.

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a Remote Exception and an Insufficient Funds Exception.

```
import java.io.*;

public class className

{

public void withdraw(double amount) throws
RemoteException,InsufficientFundsException

  {

    // Method implementation

  }

  //Remainder of class definition

}
```

Difference between throw and throws keywords

We use the throw keyword in order to force an exception. The throw keyword (note the singular form) is used to force an exception. The throw keyword also passes a custom message to your exception handling module. For instance in the above example we have used

Throw new MyException ("can't be divided by zero")

Whereas, we use the throws keyword when we already know that a particular exception will be thrown or when we pass a possible exception. Point to note here is that the Java compiler very well knows about the exceptions thrown by some methods so it insists us to handle them.

We can also use throws clause on the surrounding method instead of try and catch exception handler. For instance in the above given program we have used the following clause which will pass the error up to the next level

staticint divide(intfirst,int second) throws MyException{

Check your progress 7

1. Where is throw keyword declared in a method?
2. Write a program using throw keyword.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

2.9 Using Finally Keyword

"The finally keyword is used to create a block of code which follows a try block. A finally block of code always executes, whether or not an exception has occurred."

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
  //Protected code
}catch(ExceptionType1 e1)
{
  //Catch block
}catch(ExceptionType2 e2)
{
  //Catch block
}catch(ExceptionType3 e3)
{
  //Catch block
}finally
{
  //The finally block always executes.
}
```

Note the following:

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.

- The try block cannot be present without either catch clause or finally clause.

Any code cannot be present in between the try, catch, finally blocks.

Check your progress 8

1. Write the syntax of finally keyword.
2. Write some points for finally keyword.

.....

.....

.....

.....

.....

.....

.....

.....

.....

2.10 Nested Try Statements

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

The syntax of nested try-catch blocks is given below:

```
try {
try {
 // ...
}
catch (Exception1 e)
{
 //statements to handle the exception
}
}
catch (Exception 2 e2)
{
//statements to handle the exception
}
```

Example

```
import java.io.*;
public class NestedTryDemo{
  public static void main (String args[])throws IOException  {
    int number=5,result=0;
try{
FileInputStream fis=null;
fis = new FileInputStream (new File (args[0]))
try{
result=number/0;
System.out.println("The result is"+res);
}
catch(ArithmeticException e){
System.out.println("divided by Zero");
}
}
catch (FileNotFoundException e){
System.out.println("File not found!")
}
catch(ArrayIndexOutOfBoundsException e){
```

```
System.out.println("Array index is Out of bound!Argument required");
}
catch(Exception e){
System.out.println("Error.."+e)
}
}
}
```

Output

C:\Computer\>javac NestedTry.java

C:\Computer\>java NestedTry

Array index is Out of bound! Argument required

In this given example, we have implemented nested try-catch blocks concept where an inner try block is kept with in an outer try block, that's catch handler will handle the arithmetic exception. But before that an Array Index Out Of Bounds Exception will be raised, if a file name is not passed as an argument while running the program.

Check your progress 9

1. Write the syntax of nested try catch block.
2. Explain the execution of nested try statements.

.....

.....

.....

.....

.....

.....

.....

.....

.....

2.11 Let Us Sum Up

Deals with An exception is a problem that arises during the execution of a program. It is a runtime error. In some of the languages, which do not support exception handling, errors must be checked and handled manually typically through the use of error codes and so on. This approach is quite cumbersome and troublesome.

Then there is understanding relating to the predefined Exception, Java provides several predefined Exception classes in the package java.lang. To understand how exception handling works in Java, you need to understand these categories of exceptions:

4. Checked exceptions
5. Unchecked exceptions
6. Errors

"When the exceptions are not caught in a try/catch block, then what you often see in practice is that Java prints the exception stack trace and then terminates your program. Java actually handles uncaught exceptions according to the thread in which they occur. When an uncaught exception occurs in a particular thread, Java looks for what is called an uncaught exception handler, actually an implementation of the interface Uncaught Exception Handler. The latter interface has a method handle Exception (), which the implementer overrides to take appropriate action, such as printing the stack trace to the console."

We have understood that Java Exception handling is managed using five keywords- a. try, b. catch, c. throw, d. throws and finally e. finally.

Catching Exceptions: "A method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code"

User defined Exceptions: While creating your own exceptions in Java, the following points are to be kept in mind.

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.

- If you want to write a runtime exception, you need to extend the Runtime Exception class.

Using Throws/throw Keywords: **“**This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception. Know after that we understood. The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.”

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

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

2.12 Suggested Answer for Check Your Progress

Check your progress 1

Answers: See Section 2.2

Check your progress 2

Answers: See Section 2.3

Check your progress 3

Answers: See Section 2.4

Check your progress 4

Answers: See Section 2.5

Check your progress 5

Answers: See Section 2.6

Check your progress 6

Answers: See Section 2.7

Check your progress 7

Answers: See Section 2.8

Check your progress 8

Answers: See Section 2.9

Check your progress 10

Answers: See Section 2.9

2.13 Glossary

1. **Uncaught exceptions -** When the exceptions are not caught in a try/catch block, then what you often see in practice is Java prints the exception stack trace and then terminates your program.
2. **Catching Exceptions -** A method catches an exception using a combination of the try and catch keywords.
3. **Throw an exception -** This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception.

2.14 Assignment

Write a program which accepts two integers and an operator from the user. Perform the operation and fire the following user defined exception for following situations and handle the exceptions with proper error messages:

a. If the numbers are not of integer datatype, fire an exception

b. If the result is negative, fire an exception

c. Handle all the possible exceptions

2.15 Activities

1. Write a note on catching exceptions.
2. Explain try and catch statements

2.16 Case Study

Create a class Number having the following features:

Attributes

int	first number
int	second number
result	double stores the result of arithmetic operations performed on a and b

Member functions

Number(x, y)	constructor to initialize the values of a and b
add()	stores the sum of a and b in result
sub()	stores difference of a and b in result
mul()	stores product in result
div()	stores a divided by b in result

Test to see if b is 0 and throw an appropriate exception since division by zero is undefined.

Display a menu to the user to perform the above four arithmetic operations.

2.17 Further Reading

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, MaryCampione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition.