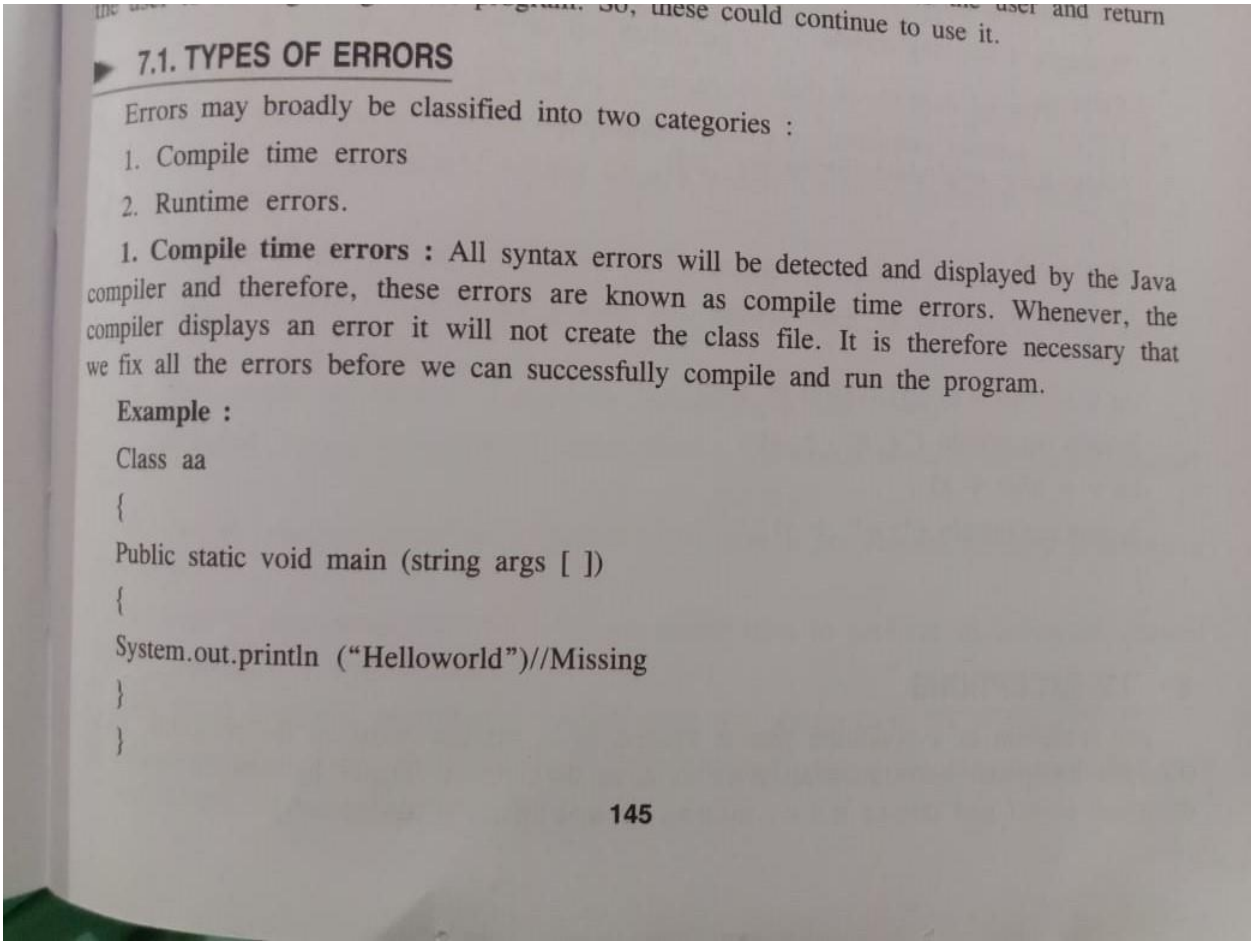
UNIT 7

Exception Handling

Error

Error is an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed.

Types of Errors



these could continue to use it.

7.1. TYPES OF ERRORS

Errors may broadly be classified into two categories :

1. Compile time errors
2. Runtime errors.

1. 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.

Example :

```
Class aa
{
Public static void main (string args [ ])
{
System.out.println ("Helloworld")//Missing
}
}
```

145

Most of compile time errors are due to typing mistakes.

(*i*) Missing semicolons.

(*ii*) Missing brackets in classes and methods.

(*iii*) Missing double quotes in strings.

(*iv*) Use of undeclared variables.

(*v*) Misspelling of identifiers and keywords.

2. Runtime errors : 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.

Most common runtime errors are as the following :

(*i*) Dividing an integer by zero.

(*ii*) Accessing an element that is out of the bounds of an array.

(*iii*) Trying to store a value into an array of an incompatible class or type.

(*iv*) Converting invalid string to a number.

(*v*) Attempting to use a negative size for an array.

(*vi*) Accessing a character that is out of bounds of a string.

Example : Runtime errors

```
Class aa
{
Public static void main (String args [ ])
{
int a = 10 ;
int b = 5 ;
int c = 5 ;
int x = a/(b - c) ;//Division by zero
System.out.println ("x =" + x) ;
int y = a/(b + c) ;
System.out.println ("y=" + y) ;
}
}
```

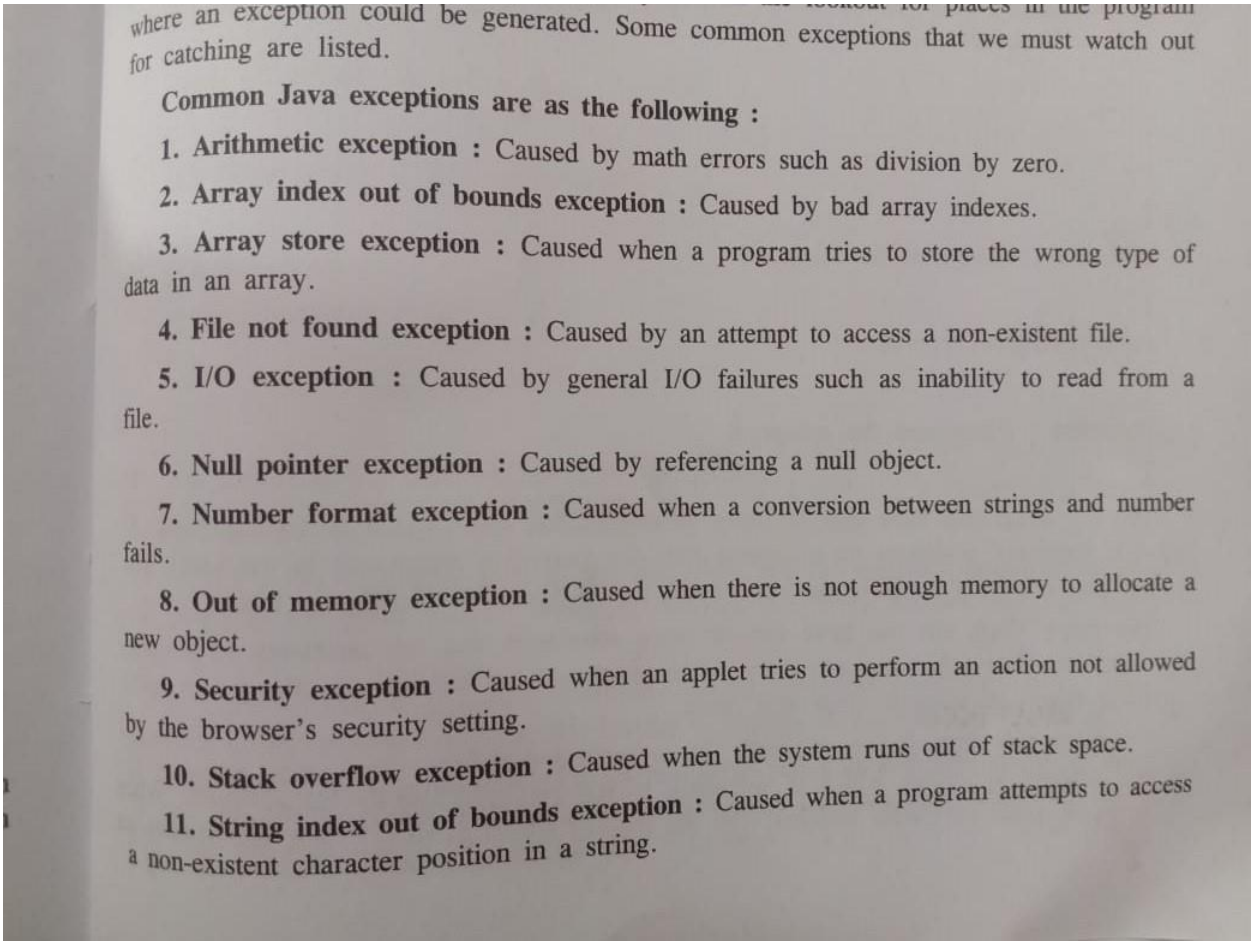
7.2. EXCEPTIONS

An exception is a condition that is caused by a runtime error in the Java interpreter encounters an error such as

Exception

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time that disrupts the normal flow of the program's instructions.

An exception is a condition that is caused by a runtime 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.



where an exception could be generated. Some common exceptions that we must watch out for catching are listed.

Common Java exceptions are as the following :

1. **Arithmetic exception :** Caused by math errors such as division by zero.
2. **Array index out of bounds exception :** Caused by bad array indexes.
3. **Array store exception :** Caused when a program tries to store the wrong type of data in an array.
4. **File not found exception :** Caused by an attempt to access a non-existent file.
5. **I/O exception :** Caused by general I/O failures such as inability to read from a file.
6. **Null pointer exception :** Caused by referencing a null object.
7. **Number format exception :** Caused when a conversion between strings and number fails.
8. **Out of memory exception :** Caused when there is not enough memory to allocate a new object.
9. **Security exception :** Caused when an applet tries to perform an action not allowed by the browser's security setting.
10. **Stack overflow exception :** Caused when the system runs out of stack space.
11. **String index out of bounds exception :** Caused when a program attempts to access a non-existent character position in a string.

Exception Handling

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

Advantage of Exception Handling

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

Java Exception Keywords/ Methods to use Exception handling

There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
Try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
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 important 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 doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

The Try block

Try block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement of try block, the rest of the block code will not execute. So, it is recommended not to keeping the code in try block that will not throw an exception.

Java try block must be followed by either catch or finally block.

The Catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

The catch block must be used after the try block only. You can use multiple catch block with a single try block.

Syntax of Java try-catch

```
1. Try
2. {
3. //code that may throw an exception
4. }
5. catch(Exception_class_Name ref)
6. {
7. }
```

Java try-catch block Example

```
1. public class TryCatchExample2
2. {
3.
4.     public static void main(String[] args)
5. {
6.         try
7.         {
8.         int data=50/0; //may throw exception
9.         }
10.                 //handling the exception
11.             catch(ArithmeticException e)
12.             {
13.                 System.out.println(e);
14.             }
15.             System.out.println("rest of the code");
16.         }
17.
18.}
```

Output:

```
java.lang.ArithmeticException: / by zero
rest of the code
```

The 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.

Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

A **finally block** contains all the crucial statements that must be executed whether exception occurs or not. The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.

Syntax of finally block

```
try
{
    //Statements that may cause an exception
}
catch
{
   //Handling exception
}
Finally
 {
   //Statements to be executed
}
```

A Simple Example of finally block

.

```
class Example
   {
   public static void main(String args[])
    {
      Try
    {
        int num=121/0;
        System.out.println(num);
      }
```

```
      catch(ArithmeticException e)
      {
         System.out.println("Number should not be divided by zero");
      }
      /* Finally block will always execute
       * even if there is no exception in try block
       */
      Finally
      {
        System.out.println("This is finally block");
      }
      System.out.println("Out of try-catch-finally");
   }
}
```

Output:

```
Number should not be divided by zero
This is finally block
Out of try-catch-finally
```

Few Important points regarding finally block

1. A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.

2. Finally block is optional, as we have seen in previous tutorials that a try-catch block is sufficient for exception handling, however if you place a finally block then it will always run after the execution of try block.

3. 3. In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.

4. An exception in the finally block, behaves exactly like any other exception.

5. The statements present in the **finally block** execute even if the try block contains control transfer statements like return, break or continue.

The Throw statement

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 used to throw an exception explicitly. Only object of Throwable class or its sub classes can be thrown. Program execution stops on encountering throw statement, and the closest catch statement is checked for matching type of exception.

Java Throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
1. public class TestThrow1
2. {
3.   static void validate(int age)
4. {
5.     if(age<18)
6.      throw new ArithmeticException("not valid");
7.     else
8.      System.out.println("welcome to vote");
9.   }
10.         public static void main(String args[])
11.      {
12.           validate(13);
13.           System.out.println("rest of the code...");
```

```
14.         }
15.       }
```

Output:

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

The Throws statement

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

```
1. return_type method_name() throws exception_class_name
2. {
3. //method code
4. }
```

Java throws keyword example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
1. import java.io.IOException;
```

```
2. class Testthrows1
3. {
4.   void m()throws IOException
5. {
6.     throw new IOException("device error");//checked exception
7.   }
8.   void n()throws IOException
9. {
10.          m();
11.        }
12.        void p()
13.       {
14.          Try
        {
15.          n();
16.         }
17.       catch(Exception e)
18.       {
19.       System.out.println("exception handled");
20.       }
21.         }
22.         public static void main(String args[])
23.       {
24.          Testthrows1 obj=new Testthrows1();
25.          obj.p();
26.          System.out.println("normal flow...");
27.         }
28.       }
```

Output:

```
exception handled
normal flow...
```

Difference between throw and throws

throw	throws
Throw keyword is used to throw an exception explicitly.	Throws keyword is used to declare an exception possible during its execution.
Throw keyword is followed by an instance of Throwable class or one of its sub-classes.	Throws keyword is followed by one or more Exception class names separated by commas.
Throw keyword is declared inside a method body.	Throws keyword is used with method signature (method declaration).
We cannot throw multiple exceptions using throw keyword.	We can declare multiple exceptions (separated by commas) using throws keyword.

Importance of exception handling in practical implementation of live projects

Example 1

```
19.     public class TryCatchExample2
20.     {
21.
22.        public static void main(String[] args)
23.     {
24.           try
25.           {
26.           int data=50/0; //may throw exception
27.           }
28.              //handling the exception
29.           catch(ArithmeticException e)
30.           {
31.              System.out.println(e);
32.           }
33.           System.out.println("rest of the code");
34.        }
35.
36.      }
```

Output:

```
java.lang.ArithmeticException: / by zero
rest of the code
```

Example 2 :

```
Class AB
{
public static void main (string args [ ])
try
{
int a[ ] = new int [10] ;
// Array has only 10 elements
a[11] = 9 ;
}
catch (Array Index out of Bound Exception)
System.out.println (''Array Index out of Bound'')
}
}
```

Output :

Array Index out of Bound Exception

Let us Revise

- An exception is a condition that is caused by a runtime erro
- Two types of errors :
  (*i*) Compile time errors
  (*ii*) Runtime errors.
- All syntax errors will be detected and displayed by the Java are known as compile time errors.
- A program may compile successfully creating the class file may produce the wrong result due to wrong logic.
- The purpose of exception handling mechanism is to provi exceptional circumstance so that right action can be taken.
- A checked exception is an exception that is typically a use foreseen by the programmer.
- A runtime exception is an exception that occurs that prob programmer.

problems that arise b