PROGRAMMING IN JAVA

UNIT-3

Exception:

an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling:

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

The core advantage of exception handling is **to maintain the normal flow of the application**.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

1. Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2. Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3. Error

Error is irrecoverable. Some examples of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords:

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

1. 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.

Syntax:

```
try{
//statements that may cause an exception
}
```

2. 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.

Syntax:

```
try
{
 //statements that may cause an exception
}
 catch (exception(type) e(object))
 {
 //error handling code
}
```

3. Finally:

The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

Syntax:

```
try {
//Statements that may cause an exception
}
catch {
//Handling exception
}
```

```
finally {

//Statements to be executed

}
```

4. Throw:

The "throw" keyword is used to throw an exception.

Syntax:

```
throw ThrowableInstance
```

5. 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:

```
type method_name(parameter_list) throws exception_list

{

 //definition of method

}
```

Java Exception Handling Example

Let's see an example of Java Exception Handling in which we are using a try-catch statement to handle the exception.

JavaExceptionExample.java

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Arithmetic Exception:

The arithmetic exception is a type of unusual outcome or unchecked error of the code, which is thrown or raised whenever a wrong mathematical or arithmetic operation appears in the code during run time. A runtime issue, also called an exception, appears whenever the denominator of a fraction is 0, and the JVM is not able to find out the result; hence the program execution is terminated, and an exception is raised. Note that at the point where the exception has been raised, the program terminates. However, the code earlier than that is executed, and the appropriate result is displayed.

Arithmetic Exception Structure

The arithmetic exception base class is java.lang.ArithmeticException, which is the child class of java.lang.RuntimeException, which in turn is the child class of java.lang.Exception.

Arithmetic Exception Constructor

1. **ArithmeticException():** It is used to define an arithmetic exception with zero parameters passed.
2. **ArithmeticException(String s):** It is used to define an arithmetic exception with one parameter passed.

Arithmetic Exception Occurrence:

The following are two situations where the arithmetic exception may occur.

1. When we perform a division where 0 is used as a divisor, i.e., 0 comes as the denominator.
2. A long decimal number that is non-terminating by Big Decimal.

Java instanceof operator

The java **'instanceof' operator** is used to test whether an object is an instance of a specified type (class or sub - class or visual interface).

'instanceof' in Java is also known as the type comparison operator because it compares the instances with type. It returns true or false. If we apply the 'instanceof' operator with a variable value, the value returned is false.

Example of Java instanceof Operator

Let's look at a simple example of instanceof operator where it checks the current class.

student.java

```
{
  public static void main( String args[ ] )
  {
    // declaring an object 's' of the student class
    student s = new student( ) ;
    // checking whether s is an instance of the student class using instanceof operator
```

```
        Boolean str = s instanceof student;
        // printing the string value
        System.out.println( str ) ;
  }
}
```

Output: True

Throwing Exception:

In Java, exceptions allow us to write good quality codes where the errors are checked at the compile time instead of runtime and we can create custom exceptions making the code recovery and debugging easier.

Java throw keyword

The Java throw keyword is used to throw an exception explicitly.

We specify the **exception** object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.

We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception. We will discuss custom exceptions later in this section.

We can also define our own set of conditions and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException if we divide a number by another number. Here, we just need to set the condition and throw exception using throw keyword.

The syntax of the Java throw keyword is given below.

throw Instance i.e.,

```
throw new exception_class("error message");
```

Let's see the example of throw IOException.

```
throw new IOException("sorry device error");
```

User-defined Exception:

An exception is an issue (run time error) that occurred during the execution of a program. When an exception occurred the program gets terminated abruptly and, the code past the line that generated the exception never gets executed.

Java provides us the facility to create our own exceptions which are basically derived classes of Exception. Creating our own Exception is known as a custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user needs. In simple words, we can say that a User-Defined Exception or custom exception is creating your own exception class and throwing that exception using the ‘throw’ keyword.

For example, MyException in the below code extends the Exception class.

Uses of user defined exceptions:

Java exceptions cover almost all the general types of exceptions that may occur in the programming. However, we sometimes need to create custom exceptions.

Following is a few of the reasons to use custom exceptions:

- To catch and provide specific treatment to a subset of existing Java exceptions.
- Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.

In order to create a custom exception, we need to extend the Exception class that belongs to **java.lang package.**

Example: We pass the string to the constructor of the superclass- Exception which is obtained using the “getMessage()” function on the object created.

```
// A Class that represents use-defined exception

class MyException extends Exception {
   public MyException(String s)
   {
      // Call constructor of parent Exception
      super(s);
   }
}

// A Class that uses above MyException
public class Main {
   // Driver Program
   public static void main(String args[])
   {
      try {
         // Throw an object of user defined exception
         throw new MyException("GeeksGeeks");
      }
      catch (MyException ex) {
         System.out.println("Caught");

         // Print the message from MyException object
         System.out.println(ex.getMessage());
      }
   }
}
```

Java Applet:

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

How to run an Applet:

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Applet Life Cycle:

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application. It basically has five core methods namely init(), start(), stop(), paint() and destroy().These methods are invoked by the browser to execute.

Along with the browser, the applet also works on the client side, thus having less processing time.

Methods of Applet Life Cycle:

There are five methods of an applet life cycle, and they are:

- **init():** The init() method is the first method to run that initializes the applet. It can be invoked only once at the time of initialization. The web browser creates the initialized objects, i.e., the web browser (after checking the security settings) runs the init() method within the applet.
- **start():** The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time the browser is loaded or refreshed, the start() method is invoked. It is also invoked whenever the applet is maximized, restored, or moving from one tab to another in the browser. It is in an inactive state until the init() method is invoked.
- **stop():** The stop() method stops the execution of the applet. The stop () method is invoked whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked. When we go back to that page, the start() method is invoked again.
- **destroy():** The destroy() method destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed. It removes the applet object from memory and is executed only once. We cannot start the applet once it is destroyed.
- **paint():** The paint() method belongs to the Graphics class in Java. It is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the start() method and when the browser or applet windows are resized.

Sequence of method execution when an applet is executed:

1. init()
2. start()
3. paint()

Sequence of method execution when an applet is executed:

4. stop()
5. destroy()

Applet Life Cycle Working:

- The Java plug-in software is responsible for managing the life cycle of an applet.
- An applet is a Java application executed in any web browser and works on the client-side. It doesn't have the main() method because it runs in the browser. It is thus created to be placed on an HTML page.
- The init(), start(), stop() and destroy() methods belongs to the **applet.Applet** class.
- The paint() method belongs to the **awt.Component** class.
- In Java, if we want to make a class an Applet class, we need to extend the **Applet**
- Whenever we create an applet, we are creating the instance of the existing Applet class. And thus, we can use all the methods of that class.

Applet Viewer:

It is a java application that allows you to view applets. It's similar to a mini-browser that will enable you to see how an applet might appear in a browser. It recognizes the APPLET tag and uses it during the creation process. The APPLET tag should be written in the source code file, with comments around it.

- Write HTML APPLET tag in comments in the source file.
- Compile the applet source code using javac.
- Use applet viewer ClassName.class to view the applet.

```
// Java program to run the applet
// using the applet viewer

import java.awt.*;
import java.applet.*;
public class GfgApplet extends Applet
{
     String msg="";
     public void init()
    {
         msg="Hello Geeks";
     }

     public void start()
     {
         msg=msg+",Welcome to GeeksForGeeks";
     }

     public void paint(Graphics g)
     {
          g.drawString(msg,20,20);
     }

}
/*
<applet code="GfgApplet" width=300 height=100>
</applet>
*/
```

To use the applet viewer utility to run the applet, type the following at the command prompt:

```
c:\>javac GfgApplet.java
c:\>appletviewer GfgApplet.java
```

Graphics in Java:

Graphics is an abstract class provided by Java AWT which is used to draw or paint on the components. It consists of various fields which hold information like components to be painted, font, color, XOR mode, etc., and methods that allow drawing various shapes on the GUI components. Graphics is an abstract class and thus cannot be initialized directly. Objects of its child classes can be obtained in the following two ways.

Graphics APIs

Swing, AWT (Abstract Window Toolkit), and Graphics run on top of the JVM (Java Virtual Machine), along with other Java libraries. This core trio is responsible for creating windows, user interface components, and graphics and painting them on a computer screen in such a manner that the user applications do not get a hint of window system APIs of the underlying native platform.

Abstract Window Toolkit (AWT)

Java’s first full-fledged GUI library was used to create GUI components for user interface programming. AWT closely works with the native libraries of the underlying platform to create and display graphical components. The library is quite heavy due to coercing with the native API library and inflexible due to compromising Java’s philosophy of least native API influence.

Swing

Swing is lightweight because the GUI components in Swing do not correspond to the native components as they do it in AWT. They are drawn using Graphics from scratch; therefore, an interesting aspect of the Swing GUI is that they can be further extended and customized in several interesting ways. The customized look and feel we find in Swing is this simple example of its flexibility.

Java Color:

Java programming language allows us to create different types of applications like windows application or web application. The user interface is an important factor while developing an application. The GUI of the Java application can be made interactive using different colors available in Java programming.

Java Color Constants

The color constants in Java are values that cannot be changed and can be used with different Java programs.

The following table shows the color constants available in the Java programming. The all-capital version depicts a constant value. But lowercase version also works fine.

The basic colors of color system are red, green, and blue. Java provides the Color class constructor with different RGB color codes as arguments. Many developer tools are available that helps in picking up the correct RGB value.

The following table shows some color code combinations using different RGB values.

Color	**RGB value**
Black	0-0-0
Very light red	255-102-102
Light red	255-51-51
Red	255-0-0
Dark red	204-0-0
Very dark red	153-0-0
Very light blue	51-204-255
Light blue	51-153-255
Blue	0-0-255
Dark blue	0-0-204
Very dark blue	0-0-153
Very light green	102-255-102
Light green	0-255-51
Green	0-204-0
Dark green	0-153-0
Very dark green	0-102-0
White	255-255-255

Java Font:

In Java, Font is a class that belongs to the java.awt package. It implements the Serializable interface. FontUIResource is the direct known subclass of the Java Font class.

It represents the font that are used to render the text. In Java, there are two technical terms that are used to represent font are characters and Glyphs.

Types of Fonts in Java

There are two types of fonts in Java:

1. Physical Fonts
2. Logical Fonts

1. Physical Fonts

Physical fonts are actual Java font library. It contains tables that maps character sequence to glyph sequences by using the font technology such as TrueType Fonts (TTF) and PostScript Type 1 Font. Note that all implementation of Java must support TTF. Using other font technologies is implementation dependent. Physical font includes the name such as Helvetica, Palatino, HonMincho, other font names.

The property of the physical font is that it uses the limited set of writing systems such as Latin characters or only Japanese and Basic Latin characters. It may vary as to configuration changes. If any application requires a specific font, user can bundle and instantiate that font by using the createFont() method of the Java Font class.

2. Logical Fonts

Java defines five logical font families that are Serif, SansSerif, Monospaced, Dialog, and DialogInput. It must be supported by the JRE. Note that JRE maps the logical font names to physical font because these are not the actual font libraries. Usually, mapping implementation is locale dependent. Each logical font name map to several physical fonts in order to cover a large range of characters.

For example, AWT components such as Label and TextField uses logical fonts only.

Font Faces and Names

A font may have many faces such as heavy, regular, medium, oblique, gothic, etc. All font faces have the similar typograph design.

A Font object have three different names that are:

Logical font name: It is the name that is used to construct the font.

Font face name: It is the name of particular font face. For example, Helvetica Bold.

Family name: It is the name of the font family. It determines the typograph design among several faces.

The Java Font class represents an instance of a font face from a collection of font faces that are present in the system resources of the host system. Example of font faces are Arial Bold, Courier Bold Italic, etc. A font face (each differing in size, style, transform and font feature) may associate with several Font objects.