**Exception**

**1**. **Introduction**

**Definition**: Exception handling is a mechanism to handle errors and unexpected situations that occur during the execution of a program.

**Importance**: Helps in graceful handling of errors, prevents program crashes, and enhances reliability.

**2. Basics of Exceptions**

**Types of Exceptions**: Checked and Unchecked.

**Checked Exceptions**: Must be handled explicitly using try-catch or declared in the method signature using throws keyword.

**Unchecked Exceptions**: Not required to be handled explicitly.

**3. Common Exceptions**

**NullPointerException**: Occurs when trying to access methods or fields of an object which is null.

**ArrayIndexOutOfBoundsException**: Occurs when trying to access an array element with an invalid index.

**ArithmeticException**: Occurs when arithmetic operations like division by zero are performed.

Example:

public class ExceptionDemo {  
  
 public static void main(String[] args) {  
 int i=8;  
 int j;  
  
 j=i/0;  
 }  
}

O/P

Exception in thread "main" java.lang.ArithmeticException: / by zero

at ExceptionsExampple.ExceptionDemo.main(ExceptionDemo.java:9)

**4. Handling Exceptions**

try-catch Block: Used to handle exceptions where code that may throw an exception is placed inside the try block and exception handling code inside the catch block.

Example:

public class HandlingException {  
 public static void main(String[] args) {  
 try {  
 // Code that may throw an exception  
 int result = 5/0; // ArithmeticException  
 } catch (ArithmeticException e) {  
 // Handling the exception  
 // System.out.println("ArithmeticException"+e.getMessage());  
 System.*out*.println("Error: Division by zero");  
 }  
 }  
}

**5. The finally Block**

Used to execute code regardless of whether an exception is thrown or not.

Commonly used for resource cleanup (closing files, releasing database connections).

Example:

public class HandlingException {  
 public static void main(String[] args) {  
 try {  
 // Code that may throw an exception  
 int result = 5/0; // ArithmeticException  
 } catch (ArithmeticException e) {  
 // Handling the exception  
 // System.out.println("ArithmeticException"+e.getMessage());  
 System.*out*.println("Error: Division by zero");  
 }  
 finally {  
 System.*out*.println("Finally block always execute");  
 }  
 }  
}

**Multiple catch Blocks: Allows handling different types of exceptions separately.**

Example:

public class MultipleCatchBlocks {  
 public static void main(String[] args) {  
 try {  
 int[] numbers = {1, 2, 3};  
 int result = numbers[5] / 0; // ArithmeticException  
 } catch (ArrayIndexOutOfBoundsException e) {  
 System.*out*.println("Invalid index: " + e.getMessage());  
 } catch (ArithmeticException e) {  
 System.*out*.println("Error: Division by zero");  
 }  
 }  
  
}

O/P

Invalid index: Index 5 out of bounds for length 3

**Explanation**

Inside the try block, an attempt is made to access the element at index 5 of the numbers array, which is beyond the bounds of the array. This results in an **ArrayIndexOutOfBoundsException**.

Additionally, there is an arithmetic operation where division by zero is attempted, resulting in an **ArithmeticException**.

Since the **ArrayIndexOutOfBoundsException** is thrown first, and it's the first catch block defined, it will be executed.

Therefore, the output will be:

Invalid index: Index 5 out of bounds for length 3

**Note : Best practice always use Main class of exception i.e Catch( Exception e) {} in the last.**

public class Main {

public static void main(String[] args) {

try {

int[] numbers = {1, 2, 3};

int result = numbers[5] / 0; // ArithmeticException

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Invalid index: " + e.getMessage());

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero");

}

catch (Exception e) {

System.out.println("Error "+e.getMessage());

}

}

}

In Java, exceptions are organized into a hierarchy, with the root of the hierarchy being the Throwable class. This hierarchy is divided into two main branches: checked exceptions and unchecked exceptions. Here's an overview of the hierarchy:

**Throwable**

* The root class for all exceptions in Java. It has two main subclasses: **Exception and Error**.

**Exception**

* Represents exceptional conditions that a well-written application should anticipate and handle.
* Divided into two main categories: checked exceptions and unchecked exceptions.

**Checked Exceptions:**

* Subclasses of Exception that must be either caught or declared to be thrown by the calling method.
* **Examples**: IOException, SQLException.

**Unchecked Exceptions:**

* Subclasses of **RuntimeException** and those that extend it.
* Typically represent programming errors or other unexpected conditions.
* Not required to be caught or declared by the calling method.
* Examples: NullPointerException, ArrayIndexOutOfBoundsException.

**Error**

* Represents serious problems that are not expected to be caught or handled by the application.
* Typically indicate problems with the runtime environment rather than the application itself.
* **Examples**: OutOfMemoryError, StackOverflowError.
* Here's a simplified visualization of the exception hierarchy:

Throwable

|

/---------------\

Exception Error

|

/ \

RuntimeException Checked Exceptions

|

Unchecked Exceptions

**Exception with throw keyword**

**What is throw keyword**

The throw keyword in Java is used to explicitly throw an exception. It is typically used when a condition occurs that disrupts the normal flow of execution in a program and requires special handling.

Here's how the throw keyword works:

**Create an Exception Object**: First, you create an object of the specific exception class that you want to throw. This object represents the occurrence of an exceptional condition.

**Throw the Exception**: Then, you use the throw keyword followed by the exception object to throw the exception.

Here's the basic syntax of using the throw keyword:

**throw new ExceptionType("Exception message");**

Example 1

public class ExceptionThrowKeyword {  
  
 public static void main(String[] args) {  
 int i=20,j;  
 try{  
 j=18/i;  
 if(j==0){  
 throw new ArithmeticException("I dont want to print Zero");  
 }  
 }  
 catch (ArithmeticException e) {  
 j=18/1;  
 System.*out*.println("Thats the default Output" +e);  
 }  
 System.*out*.println(j);  
 }  
}

try Block:

try: The code inside the try block is the segment where exceptions may occur.

j = 18 / i;: Attempts to perform division 18 / i. If i is 0, this operation will result in an ArithmeticException.

if (j == 0): Checks if the result of the division is 0.

throw new ArithmeticException("I dont want to print Zero");: If the result of the division is 0, it throws a new ArithmeticException with the message "I dont want to print Zero".

Example 2

public class ExceptionThrowKeyword1 {  
 public static void main(String[] args) {  
 try {  
 // Manually throw an exception  
 throw new IllegalArgumentException("Invalid argument supplied");  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Caught IllegalArgumentException: " + e.getMessage());  
 }  
 }  
 }

In this program:

Inside the try block, we manually throw an IllegalArgumentException using the throw keyword.

We provide a message "Invalid argument supplied" to the constructor of the IllegalArgumentException.

The catch block catches the thrown IllegalArgumentException and prints the message associated with it using getMessage() method.

When you run this program, it will output:

**Caught IllegalArgumentException: Invalid argument supplied**

**Custom Exception**

package ExceptionsExampple;  
  
class vikashException extends Exception{  
  
 public vikashException(String string){  
 super(string);  
 }  
}  
  
public class CustomException {  
  
 public static void main(String[] args) {  
 int i=20,j;  
 try{  
 j=18/i;  
 if(j==0){  
 throw new vikashException("I dont want to print Zero");  
 }  
 }  
 catch (vikashException e) {  
 j=18/1;  
 System.*out*.println("Thats the default Output"+e);  
 }  
 System.*out*.println(j);  
 }  
}

**Exception With throws Keyword**

Java, the throws keyword is used in the method declaration to specify that the method might throw one or more types of exceptions. It indicates that the method does not handle the exceptions itself but instead expects the caller of the method to handle them.

Here's how the throws keyword works:

**Method Declaration**: You add the throws keyword followed by a list of exception classes to the method signature. This indicates that the method might throw exceptions of these types during its execution.

**Caller's Responsibility**: When a method is declared with throws, the caller of that method must either handle the exceptions using a try-catch block or propagate them further by declaring them with throws in its own method signature.

package ExceptionsExampple;  
  
class A{  
  
 public void show() throws ClassNotFoundException {  
  
 Class.*forName*("calc");  
 }  
}  
  
public class ExceptionWithThrows {  
  
 static{  
 System.*out*.println("class loder");  
 }  
  
 public static void main(String[] args) {  
 A obj =new A();  
 try {  
 obj.show();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}