# **Exception Handling in Java**

Exception:

- Exception is an unwanted condition that occurs during the execution of the program and disrupts the normal flow of instructions.
- It represents an unexpected event that the program may encounter.
- When an exceptional situation arises, Java creates an object called Exception object representing the specific exception type and description of the exception. These exceptions can be caught and handled to prevent the program from terminating abruptly.

Exception Handling:

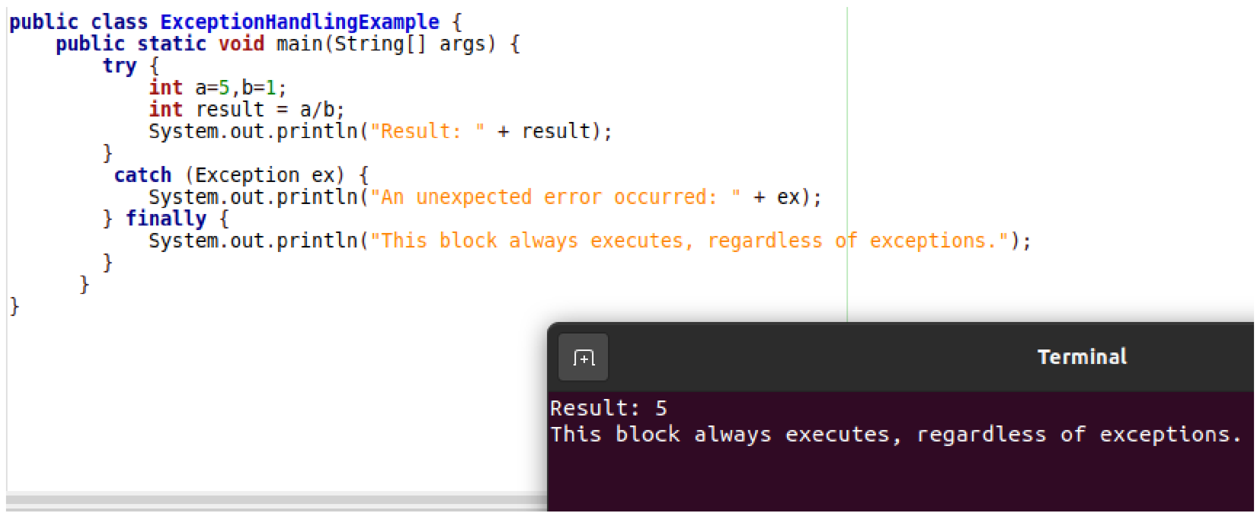
- Exception handling is the process of managing and responding to exceptions that occur while executing the program.
- It allows the program to handle errors, log meaningful error messages, and take appropriate actions to recover from exceptional situations.

- Exception handling is implemented using different ways.
- General Syntax of try-catch block:

```
try {
   // Code that may throw an exception
} catch (ExceptionType1 ex1) {
   // Code to handle ExceptionType1
} catch (ExceptionType2 ex2) {
   // Code to handle ExceptionType2
} finally {
   // Code that always executes (optional)
```

}

Example in Java: Let's consider a simple example where we try to divide two numbers and handle exceptions:

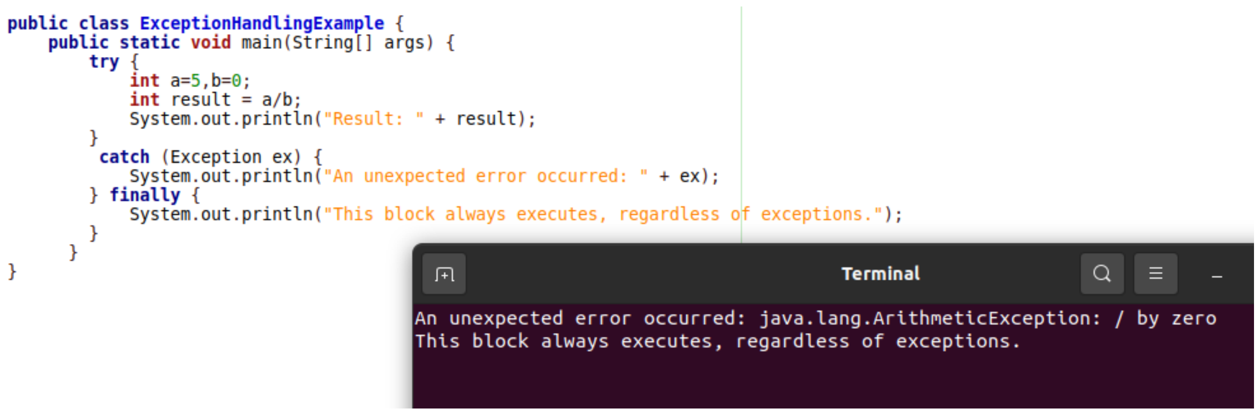


```java
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            int a=5,b=1;
            int result = a/b;
            System.out.println("Result: " + result);
        }
         catch (Exception ex) {
            System.out.println("An unexpected error occurred: " + ex);
        } finally {
            System.out.println("This block always executes, regardless of exceptions.");
        }
      }
}
```

```
Result: 5
This block always executes, regardless of exceptions.
```

The try block: It contains the code that may throw an exception.

The catch block: If an exception occurs within the try block, Java looks for an appropriate catch block to handle that exception. In this example, If any unexpected exception occurs in the program, the catch block with the Exception parameter will catch it.
The finally block: This block is optional and is used to execute code that should be run whether an exception occurred or not. It is typically used to perform cleanup tasks or resource release operations. The finally block will execute no matter what.

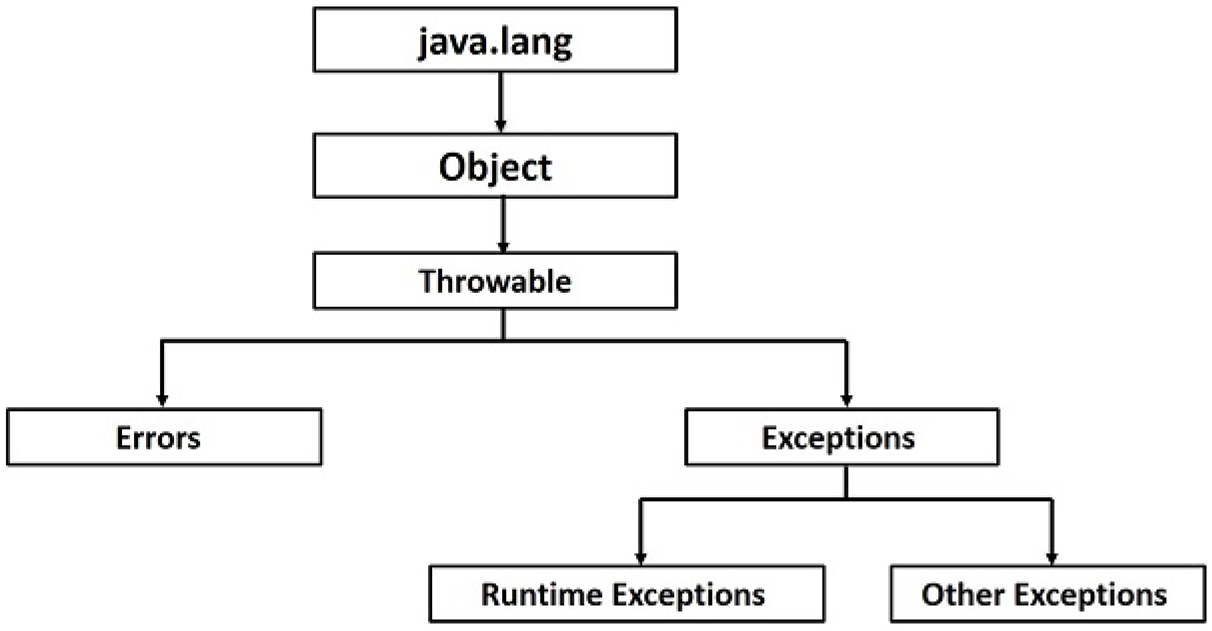


```java
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            int a=5,b=0;
            int result = a/b;
            System.out.println("Result: " + result);
        }
         catch (Exception ex) {
            System.out.println("An unexpected error occurred: " + ex);
        } finally {
            System.out.println("This block always executes, regardless of exceptions.");
        }
      }
}
```

```
An unexpected error occurred: java.lang.ArithmeticException: / by zero
This block always executes, regardless of exceptions.
```

In the first case, the division is performed successfully. In the second case, an ArithmeticException is thrown when dividing by zero. So, we can handle exceptions and keep your program running smoothly even when errors occur.

Exception Hierarchy:



**java. lang.Throwable:**

- All the exceptions are derived from the java.lang.Throwable class.
- Throwable is at the top of Exception hierarchy.
- The throwable class has two subclasses that partition exception into Exception and Errors.

**Error:**

- Error are the problems that are beyond the control of the application and are not caught or handled by regular application code. It typically lead to the termination of the application.
- It is impossible to recover.  Errors are of unchecked type i.e. it happens at run-time.
- It usually indicates critical failures in the Java Virtual Machine (JVM) or the underlying hardware or environment.
    - Example: ‘OutOfMemoryError’, ‘NoClassDefFoundError’ etc.

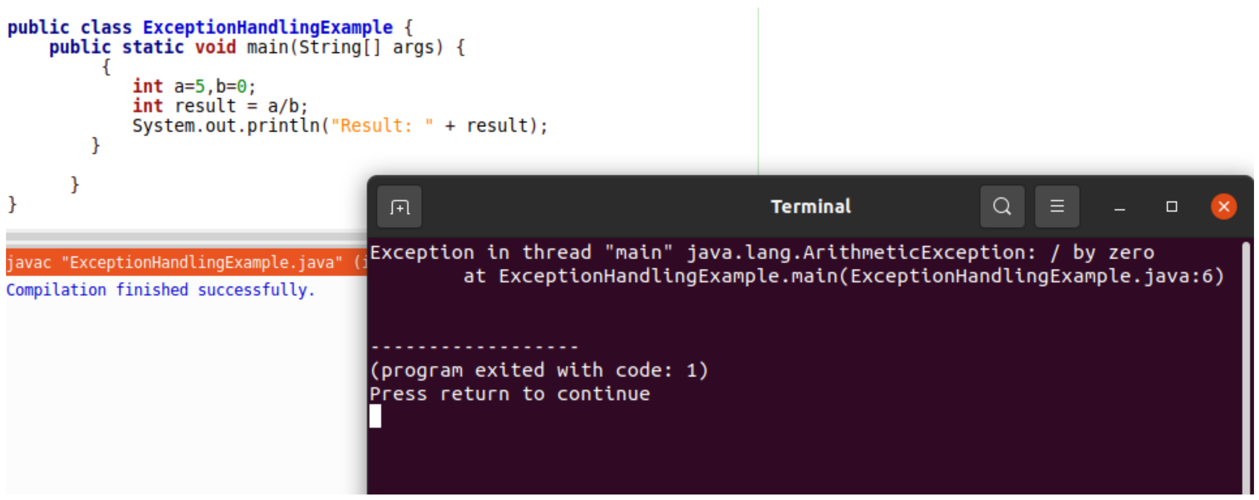
**Exception:**

- It indicates the exceptional conditions that can be caught and handled by the application.
- It can happen at compile time and run time.
- It can be recovered by handling them.
- It is of two types: Checked Exception, Unchecked Exception
    - Checked Exception:
      These are exceptions that must be either caught and handled during compile time.
      These are exceptions that need to be explicitly caught or declared in the method signature.
      If no exception handling code is provided, then compiler signals a compilation error.
      Examples: ‘IOException’, ‘SQLException’ etc.

- Unchecked Exception:
  These are exceptions that do not need to be explicitly caught or declared in the method signature.so it's up to the developer to handle them properly.
  These are the exceptions that are not checked at compile time.

  Examples:‘ArithmeticException’, ArrayIndexOutOfBoundsException’, etc.

Example 2:
Without exception handling at compile time(unchecked exception)

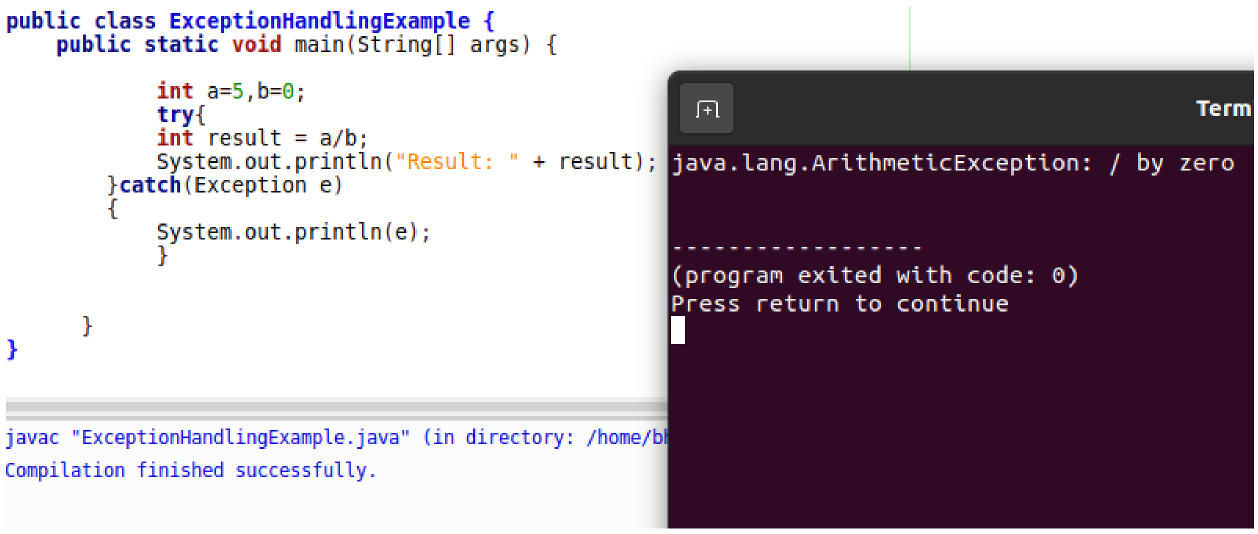


```
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        {
            int a=5,b=0;
            int result = a/b;
            System.out.println("Result: " + result);
        }

    }
}
```

```
javac "ExceptionHandlingExample.java" (i
Compilation finished successfully.
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionHandlingExample.main(ExceptionHandlingExample.java:6)


------------------
(program exited with code: 1)
Press return to continue
```

Here, there is no identification of exception during compile time and caught during compile time. The exception is identified only during run time. So, it is unchecked exception.

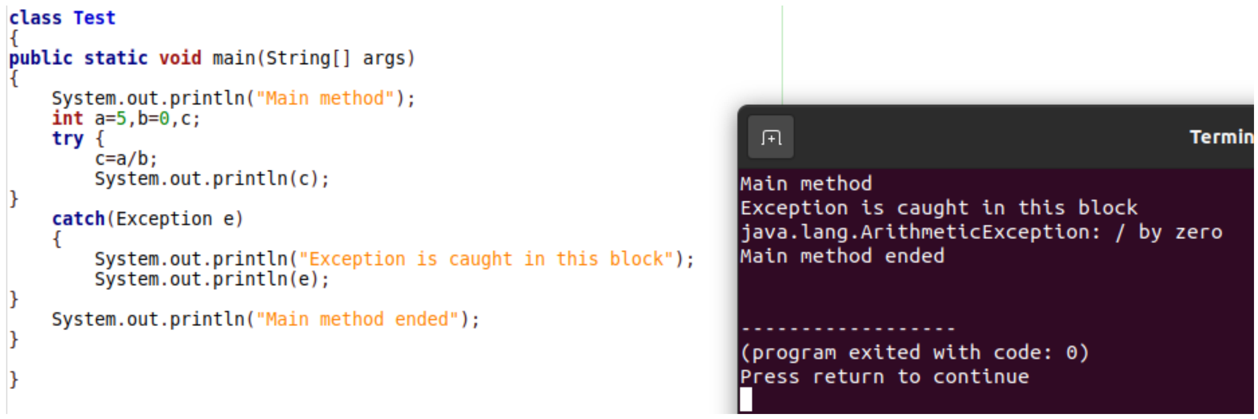


```
public class ExceptionHandlingExample {
    public static void main(String[] args) {

            int a=5,b=0;
            try{
            int result = a/b;
            System.out.println("Result: " + result);
        }catch(Exception e)
        {
            System.out.println(e);
            }


      }
}
```

```
javac "ExceptionHandlingExample.java" (in directory: /home/bh
Compilation finished successfully.
```

```
java.lang.ArithmeticException: / by zero


------------------
(program exited with code: 0)
Press return to continue
```

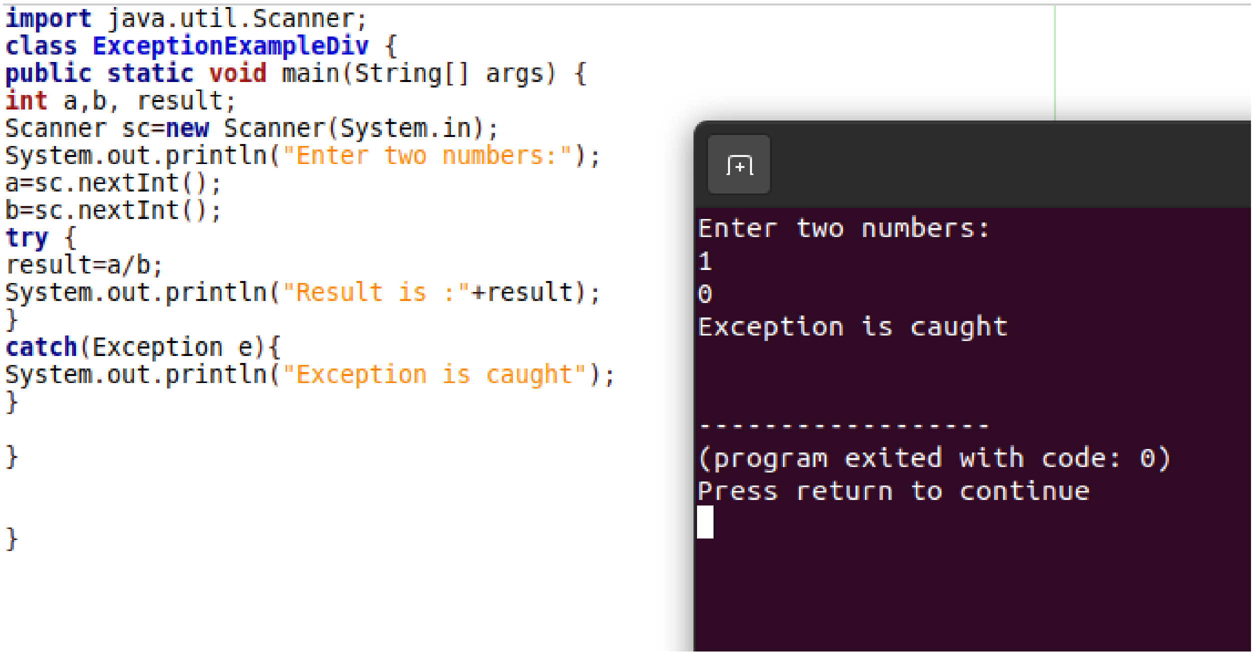
Example 3:



```
class Test
{
public static void main(String[] args)
{
    System.out.println("Main method");
    int a=5,b=0,c;
    try {
        c=a/b;
        System.out.println(c);
}
    catch(Exception e)
    {
        System.out.println("Exception is caught in this block");
        System.out.println(e);
}
    System.out.println("Main method ended");
}

}
```

```
Main method
Exception is caught in this block
java.lang.ArithmeticException: / by zero
Main method ended


------------------
(program exited with code: 0)
Press return to continue
```

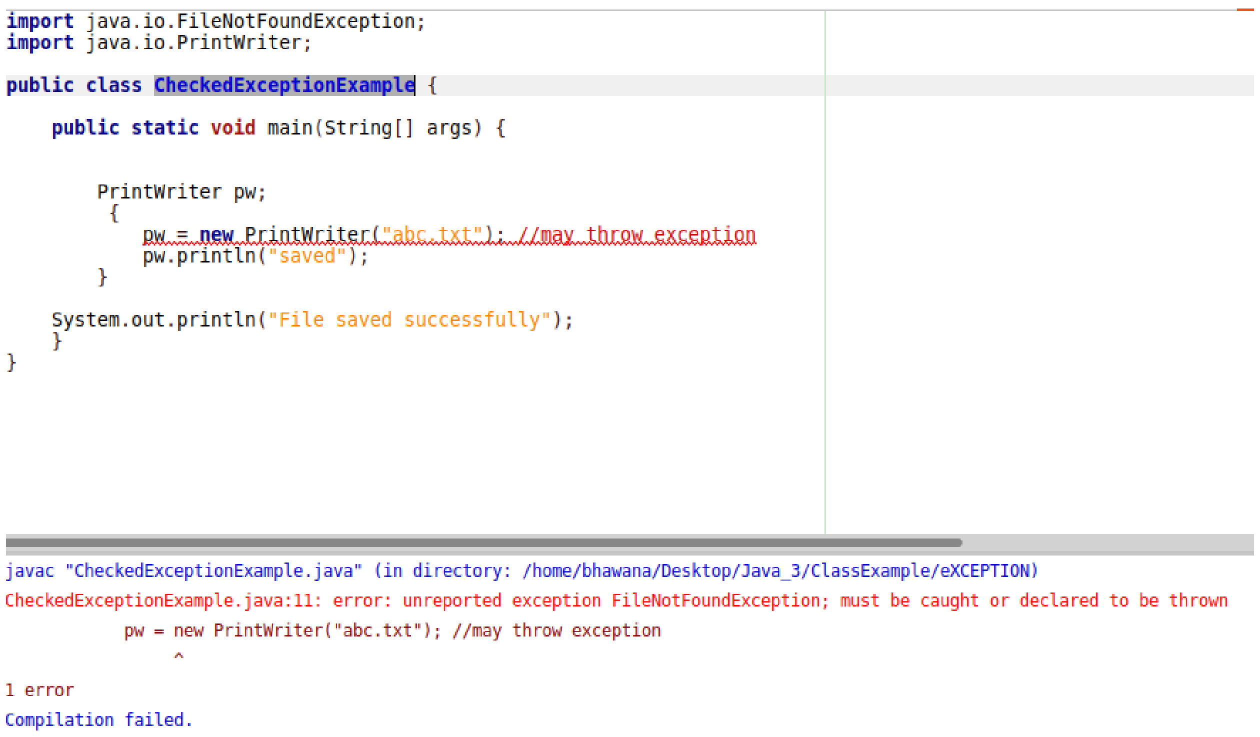
Example 4:



```
import java.util.Scanner;
class ExceptionExampleDiv {
public static void main(String[] args) {
int a,b, result;
Scanner sc=new Scanner(System.in);
System.out.println("Enter two numbers:");
a=sc.nextInt();
b=sc.nextInt();
try {
result=a/b;
System.out.println("Result is :"+result);
}
catch(Exception e){
System.out.println("Exception is caught");
}

}


}
```

```
Enter two numbers:
1
0
Exception is caught


------------------
(program exited with code: 0)
Press return to continue
```

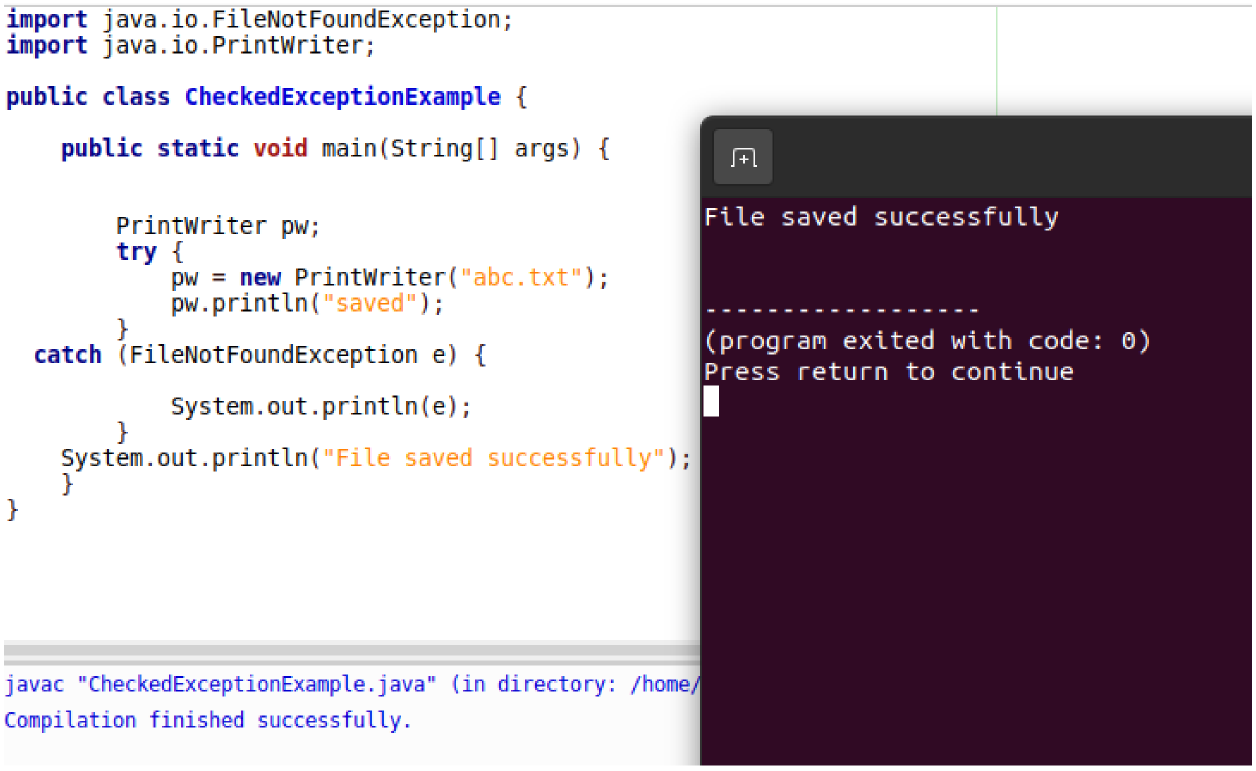
Example 5:(Checked exception)



```
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class CheckedExceptionExample {

    public static void main(String[] args) {


        PrintWriter pw;
         {
            pw = new PrintWriter("abc.txt"); //may throw exception
            pw.println("saved");
        }

    System.out.println("File saved successfully");
    }
}
```

```
javac "CheckedExceptionExample.java" (in directory: /home/bhawana/Desktop/Java_3/ClassExample/eXCEPTION)
CheckedExceptionExample.java:11: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
            pw = new PrintWriter("abc.txt"); //may throw exception
                 ^
1 error
Compilation failed.
```

## For solving:



```
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class CheckedExceptionExample {

    public static void main(String[] args) {


        PrintWriter pw;
        try {
            pw = new PrintWriter("abc.txt");
            pw.println("saved");
        }
  catch (FileNotFoundException e) {

            System.out.println(e);
        }
    System.out.println("File saved successfully");
    }
}
```

```
javac "CheckedExceptionExample.java" (in directory: /home/
Compilation finished successfully.
```

```
File saved successfully


------------------
(program exited with code: 0)
Press return to continue
```

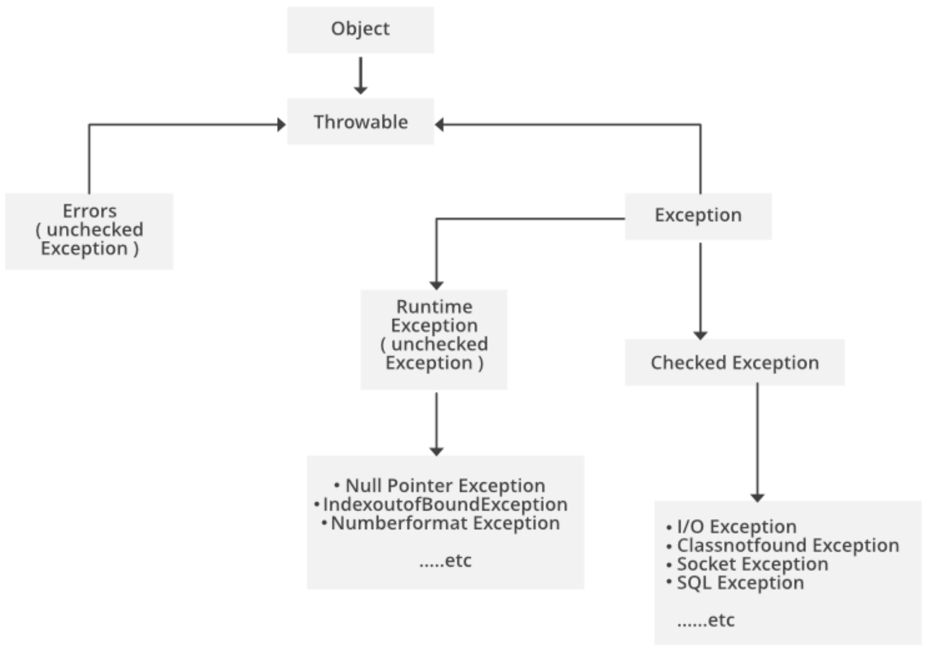
Saved location:

**Java Exception Keywords:**

| **Keyword** | **Description** |
|---|---|
| try | This keyword is used to specify a block and this block must be followed by either catch or finally. That is, we can't use try block alone. |

| | |
|---|---|
| catch | This keyword must be preceded by a try block to handle the exception and can be followed by a final block later. |
| finally | This keyword is used to execute the program, whether an exception is handled or not. |
| throw | This keyword is used to throw an exception. |
| throws | This keyword is used to declare exceptions. |

Exception Classes:

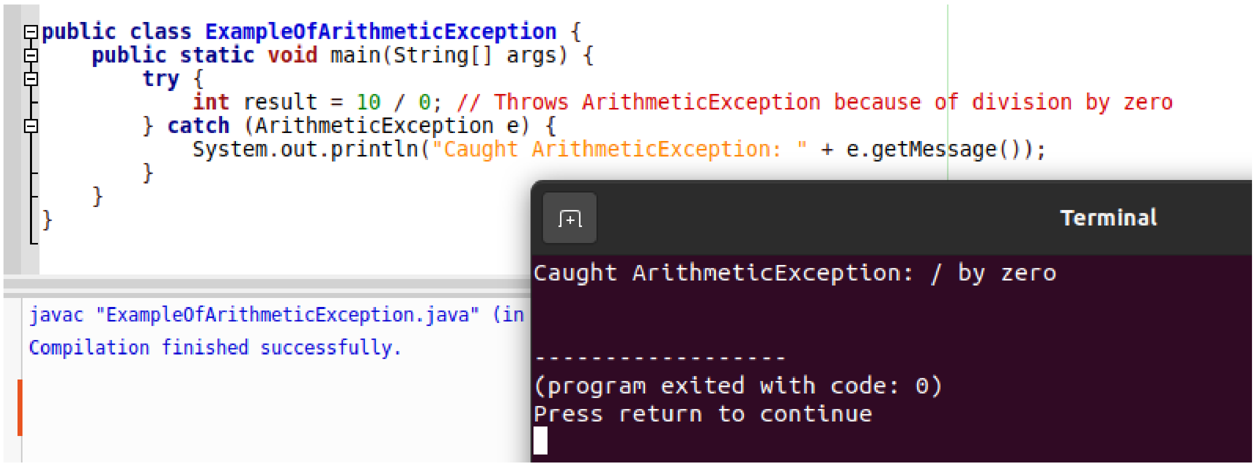


Different  Scenarios of Exception:

1. ArithmeticException

   It occurs when there is exceptional condition in  arithmetic operations such as division or modulo. Example: Division by zero or modulo operation with a zero divisor.

   int a=10/0;
   Program:



```
public class ExampleOfArithmeticException {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // Throws ArithmeticException because of division by zero
        } catch (ArithmeticException e) {
            System.out.println("Caught ArithmeticException: " + e.getMessage());
        }
    }
}
```

```
javac "ExampleOfArithmeticException.java" (in
Compilation finished successfully.
```

```
Caught ArithmeticException: / by zero


------------------
(program exited with code: 0)
Press return to continue
```

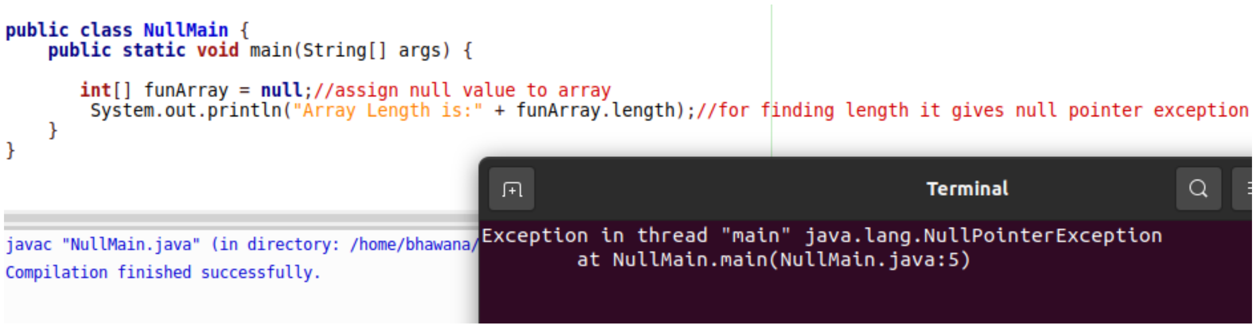
2. NullPointerException

   It occurs when you try to access an object reference that is null. For eg. if we try to call a method or access a field on a variable that is not initialized as:

   String name=null;

   System.out.println(name.length());

   **Program Example:**



```
public class NullMain {
    public static void main(String[] args) {

        int[] funArray = null;//assign null value to array
         System.out.println("Array Length is:" + funArray.length);//for finding length it gives null pointer exception
    }
}
```

```
javac "NullMain.java" (in directory: /home/bhawana/
Compilation finished successfully.
```

```
Exception in thread "main" java.lang.NullPointerException
        at NullMain.main(NullMain.java:5)
```

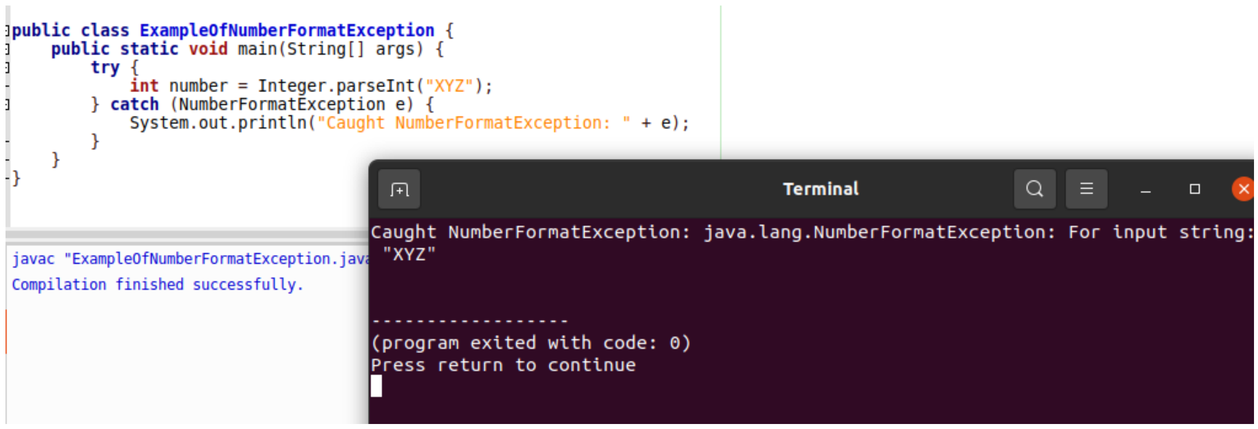
3. NumberFormatException

   It occurs when you try to convert a string to  a numeric type, but the string donot have a valid numeric format.
   Eg. parsing a non-numeric string using Integer.parseInt() or Double.parseDouble().

   String name="abc";

   int i=Integer.parseInt(name);

   Program Example:



```
public class ExampleOfNumberFormatException {
    public static void main(String[] args) {
        try {
            int number = Integer.parseInt("XYZ");
        } catch (NumberFormatException e) {
            System.out.println("Caught NumberFormatException: " + e);
        }
    }
}
```

```
javac "ExampleOfNumberFormatException.java
Compilation finished successfully.
```

```
Caught NumberFormatException: java.lang.NumberFormatException: For input string:
 "XYZ"

------------------
(program exited with code: 0)
Press return to continue
```

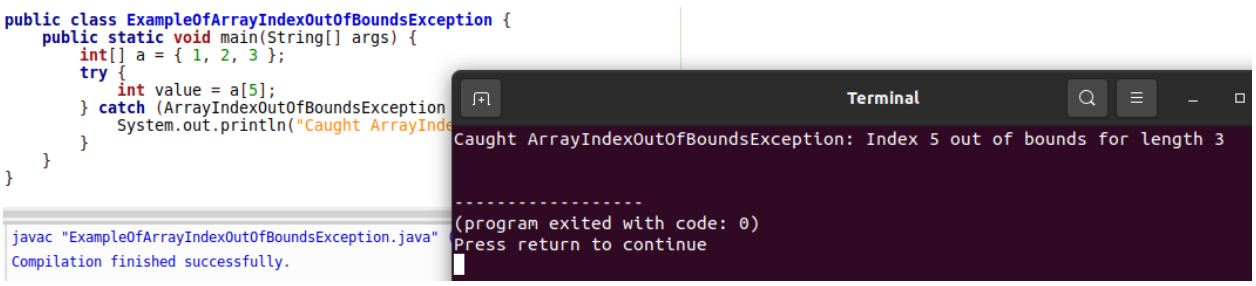
4. ArrayIndexOutOfBoundsException

   It occurs when we try to access an invalid index of an array. Eg. Accessing an element at an index that is outside the bounds of the array as:

   int a[]=new int[5];

   a[7]=12;

   Program Example:



```
public class ExampleOfArrayIndexOutOfBoundsException {
    public static void main(String[] args) {
        int[] a = { 1, 2, 3 };
        try {
            int value = a[5];
        } catch (ArrayIndexOutOfBoundsException
            System.out.println("Caught ArrayInde
        }
    }
}
```

```
javac "ExampleOfArrayIndexOutOfBoundsException.java" (
Compilation finished successfully.
```

```
Caught ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3


------------------
(program exited with code: 0)
Press return to continue
```