Exception Handling in Java

What is an exception?

- Exception is an event that occurs during the execution of a program that disrupts the normal flow of control in a program.

- **Exception handling** in java is one of the most effective mechanisms to handle the runtime errors so that normal flow of the application can be maintained.

- Exception Handling is a mechanism to handle runtime errors.

What is an error?

- Errors are problems that mainly occur due to the lack of system resources.
- It is mostly caused by the environment in which the application is running.
- It cannot be caught or handled.
- Example: OutOfMemoryError, StackOverflowError etc.
- **Exception can be recovered by using the try-catch block. An error cannot be recovered.**

Types of Exceptions

- **Checked**: are the exceptions that are checked at compile time.
- If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

- **Unchecked** are the exceptions that are not checked at compiled time.
- In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception.

Java built-in exceptions

- ArithmeticException
- ArrayIndexOutOfBounds
- ClassNotFoundException
- FileNotFoundException
- NullPointerException
- NumberFormatException
- StringIndexOutOfBoundsException

A scenario of checked exception

```
import java.io.*;
public class Exception_HandlingClass2 {
public static void main(String[] args) {
   FileInputStream  f =new FileInputStream("hello.txt");
}
}
```

*** compiler will show a **warning message**.

*** Use **try-catch block** or **throws** keyword.

```
import java.io.*;
public class Exception_HandlingClass2 {
    public static void main(String[] args) {
        try{
           FileInputStream  f =new   FileInputStream("hello.txt");
        }
        catch(FileNotFoundException e){
            System.out.println(e);
        }
    }
}
```

*** A **FileNotFoundException** will be thrown if **hello.txt** file is not found.

Some Scenarios (Unchecked Exception)

1. `int a=50/0;` //ArithmeticException

2.
```
String s=null;
System.out.println(s.length());
```
//NullPointerException

3.
```
String s="abc";
int i=Integer.parseInt(s);
```
//NumberFormatException

4.
```
int a[]=new int[5];
a[10]=50;
```
//ArrayIndexOutOfBoundsException

Java Exception Handling Keywords

Keywords used in java for exception handling:

- Try
- Catch
- Finally
- Throw

Exception Handling Terms

- try – used to enclose a segment of code that may produce an exception
- catch – placed directly after the try block to handle one or more exception types
- finally – execute important code such as closing connection, stream etc.
- throw – to generate an exception or to describe an instance of an exception

Try-Catch-Finally

Syntax of java try-catch:

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

OR

Syntax of try-finally block:

```
try{
//code that may throw exception
}finally{}
```

Problem without exception handling

```
public class ExceptionExample {


    public static void main(String[] args) {
        int data = 50 / 0;
System.out.println("rest the code ....");
    }
}
```

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

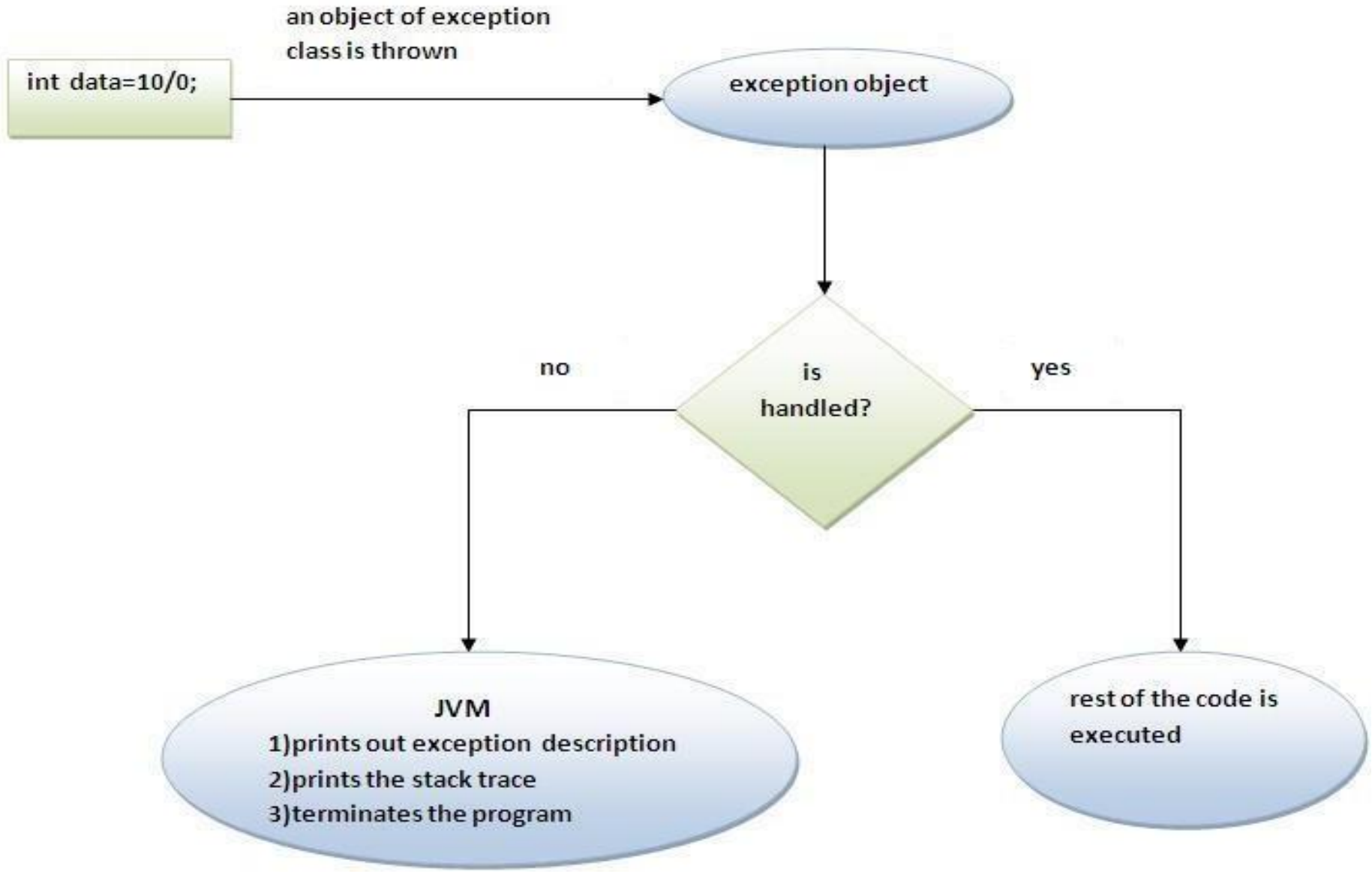
Solution by exception handling Try-Catch

```
public class ExceptionExample {


    public static void main(String[] args) {
        try {
            int data = 50 / 0;
        }
        catch (ArithmeticException e) {
            System.out.println(e);
        }
        System.out.println("rest the code ....");
    }
}
```

Output:
java.lang.ArithmeticException: / by zero
rest the code ....

Internal working of java try-catch block



Catch multiple exceptions

- A try block can be followed by one or more catch blocks.
- Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
- At a time only one exception occurs and at a time only one catch block is executed
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception

```
public class MultipleCatch {
    public static void main(String[] args) {
        try {
            int a[] = new int[5];
            a[5] = 30 / 0;        }
catch (ArithmeticException e) {
            System.out.println("task1 is completed");        }
 catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("task 2 completed");        }
catch (Exception e) {
            System.out.println("common task completed");        }
        System.out.println("rest of the code...");
    }}
```

Output:
task1 is completed
rest of the code...

Compilation error (Not hierarchy maintained!)

```
public class MultipleCatch {
    public static void main(String[] args) {
        try {
            int a[] = new int[5];
            a[5] = 30 / 0;         }
catch (Exception e) {
            System.out.println("common task completed");         }
catch (ArithmeticException e) {
            System.out.println("task1 is completed");         }
catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("task 2 completed");         }
System.out.println("rest of the code...");
    }}
```

Java 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 must be followed by try or catch block.

Why use java finally

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

exception doesn't occur.

```
public class FinallyException {
    public static void main(String[] args) {
        try {
            int data = 25 / 5;
            System.out.println(data);          }
catch (NullPointerException e) {
            System.out.println(e);        }
 finally {
   System.out.println("finally block is always executed");
     }
        System.out.println("rest of the code...");     }
}
```

Output:
5
finally block is always executed
rest of the code...

exception occurs and not handled.

```
public class FinallyException {
    public static void main(String[] args) {
        try {
            int data = 25 / 0;
            System.out.println(data);        }
catch (NullPointerException e) {
            System.out.println(e);        }
finally {
     System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```

Output:
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero

exception occurs and handled.

```
public class FinallyException {
    public static void main(String[] args) {
        try {
            int data = 25 / 0;
            System.out.println(data);
        }
catch (ArithmeticException e) {
            System.out.println(e);
        }
finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```

Output:
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...

Throw

- The Java throw keyword is used to explicitly throw an exception.
- The throw keyword is mainly used to throw custom exception.

Example

```
public class ThrowException {
  static void validate(int age){
     if(age<18)
      throw new ArithmeticException("not valid");
     else
      System.out.println("welcome to vote");
   }
   public static void main(String args[]){
      validate(13);
      System.out.println("rest of the code...");
  }
}
```

Output:

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

Custom Exception

- To create a custom exception, we have to extend the Exception class.
- we also have to provide a constructor that takes a String as the error message and called the parent class constructor.

```
class CustomException extends Exception{
    CustomException(String message){
        super(message);
    }
}
```

- To throw the Custom Excetion we have to use throw keyword.

```
throw new CustomException("Exception message");
```