Exception Handling

Exception handling in java is a powerful mechanism or technique that allows us to handle runtime errors in a program so that the normal flow of the program can be maintained. All the exceptions occur only at runtime. A syntax error occurs at compile time.

What is Exception in Java?

An **exception in java** is an object representing an error or an abnormal condition that occurs
at runtime execution and interrupts (disrupts) the normal execution flow of the program.

```
import java.util.Scanner;
public class Test
{
public static void main(String[] args)
{
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter two integer numbers");

// Read two integer numbers.
  int num1 = sc.nextInt();
  int num2 = sc.nextInt();
  System.out.println(num1 + "/" + num2 + " = " + (num1/num2));
  }
}
```

Why does Exception occur in Program?

1. Opening a non-existing file in your program.
2. Reading a file from a disk, but the file does not exist there.
3. Writing data to a disk but the disk is full or unformatted.
4. When the program asks for user input and the user enters invalid data.
5. When a user attempts to divide an integer value by zero, an exception occurs.
6. When a data stream is in an invalid format, etc.

Exception Handling in Java

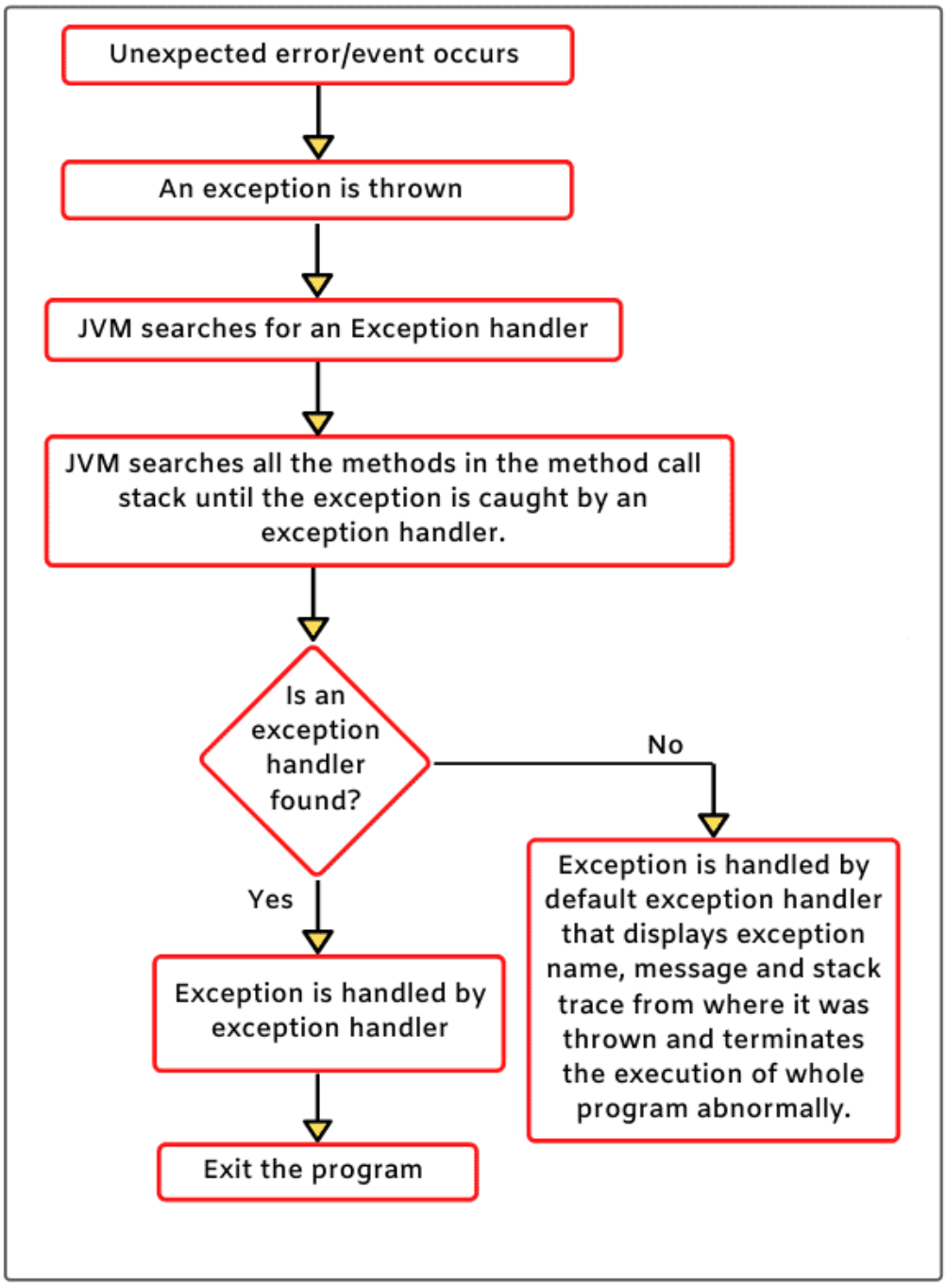
The mechanism of handling unexpected errors in a java program is called exception handling. It is a
 powerful mechanism to handle runtime errors, ClassNotFoundException, FileNotFoundException, IOException, etc. so that the normal execution flow of the program can be maintained.,

What is Exception Handler in Java?

The code that catches the exception thrown by JVM is called **exception handler in Java**. It is responsible for receiving information about the exception/error.

How does Exception handling mechanism work?

The main
purpose of using exception handling mechanism in a java program is to
handle unexpected errors and maintain the normal flow of the program.



1. When an exception occurs inside a method in java program, the method in which exception has occurred, creates an exception object (i.e., an object of exception class) internally with the help of JVM and hands it over to the java runtime system (JVM). This process is called **throwing an exception in java**.The exception object contains information about the exception such as the name of exception, and Stack Trace/Location.

---

2. When a java method throws an exception, JVM searches for a method in the method call stack that can handle that exception. A method call stack is an ordered

list of methods. This process continues until the exception is caught by an exception handler.

The method which handles thrown exception is called exception handler. It is used to catch the exception that is thrown by JVM. This process is called **catching an exception**.

---

3. If an exception handler is found, the control of execution is transferred to exception handler, and statements specified inside exception handler are executed.

4. If JVM does not find an appropriate exception handler, exception is caught by the default exception handler provided by JVM.

Default exception handler is a part of the run-time system that displays exception information on the console such as exception name, message, and a full stack trace from where it was thrown.

The full stack track describes the sequence of steps that is responsible for throwing an error.

5.After printing exception information on the console, the default exception handler terminates the execution of the whole program abnormally.Thus, an exception handler mechanism works to manage unexpected situations in a program at runtime. The exception may be user-defined (custom exception) or predefined. Most of the exceptions are predefined and must be handled in the program to avoid runtime errors.

You can keep in mind the working of exception handling mechanism by the below points.

- Get the exception (Detects problem)
- Throw exception (Inform that an error has occurred)
- Catch the exception (Receive error information)
- Handle exception (Take appropriate and corrective actions)

In Java program, we can handle such unexpected errors by using try and catch block.

```
package exceptionHandling;
public class ExceptionWithoutError
{
public static void main(String[] args)
{
  System.out.println("One");
  System.out.println("Two");
  System.out.println("Three");
  System.out.println("Four");
```

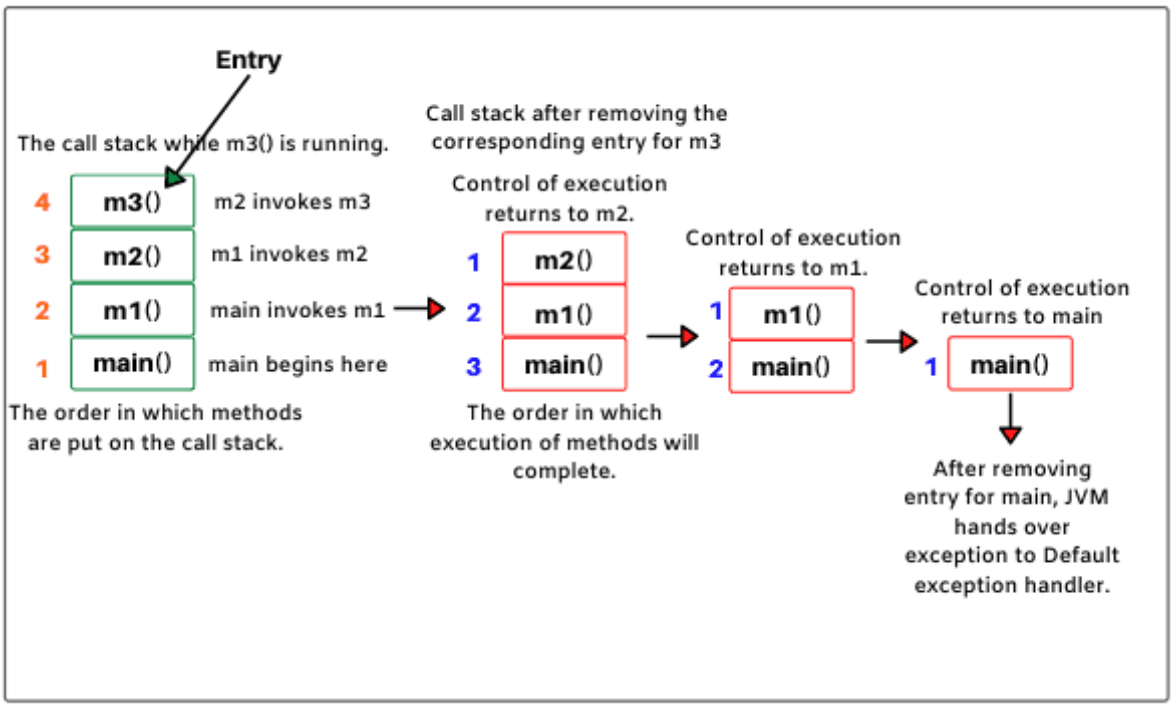
```
 }
}
```

```
package exceptionHandling;
public class ExceptionDivideByZero
{
public static void main(String[] args)
{
 System.out.println("One");
 System.out.println("Two");

 int a = 1/0; // Exceptional case.

 System.out.println("Three");
 System.out.println("Four");
 }
}
```

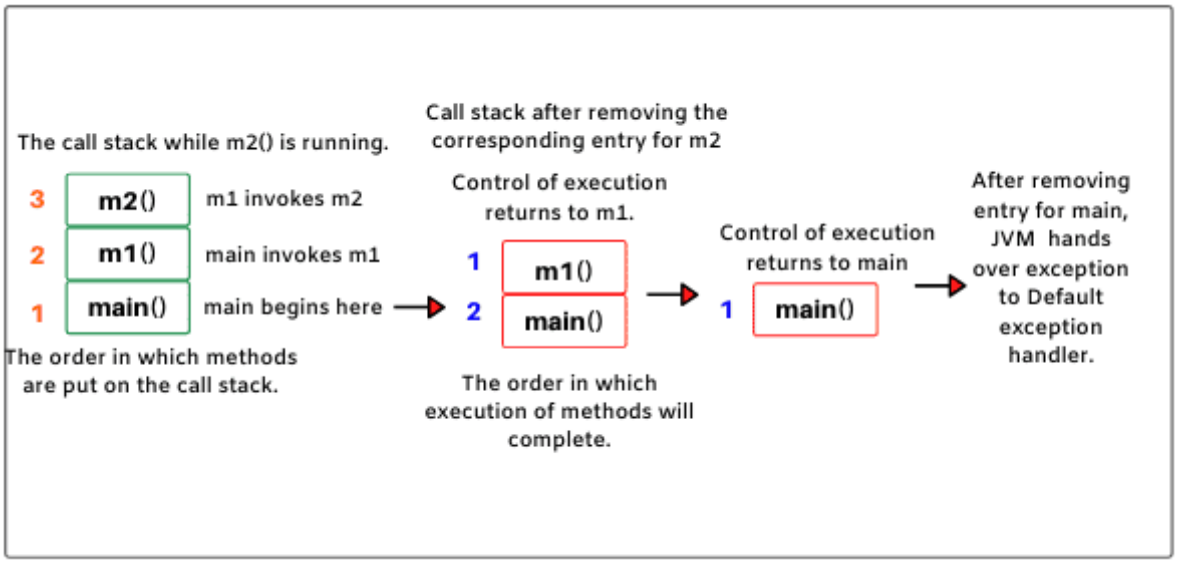
```
package exceptionHandling;
public class ExceptionHandlingwithTryCatch {
public static void main(String[] args)
{
  System.out.println("One");
  System.out.println("Two");
// Declare a try-catch block.
  try // Error handling code starts here.
  {
   System.out.println("Before divide");
   int a = 1/0; // Exceptional case (Exception has occurred).
   System.out.println("After divide");
  }
  catch(ArithmeticException e) // Exception handled. Here, catch block is an exception
 handler.
  {
    System.out.println("A number cannot be divided by zero."); // User-friendly message
  }
  System.out.println("Three");
  System.out.println("Four");
 }
}
```

```
package exceptionHandling;
public class DefaultExceptionHandlingEx1 {
public static void main(String[] args)
{
  m1(); // main() method calling m1().
}
public static void m1()
{
  m2(); // m1() method calling m2().
}
```

```
public static void m2()
{
  m3(); // m2() method calling m3().
}
public static void m3()
{
  System.out.println(1/0); // Exceptional case. A number cannot be divided by zero.
 }
}
```




```
package exceptionHandling;
public class DefaultExceptionHandlingEx2
{
public static void main(String[] args)
{
  m1(); // main() method calling m1().
}
public static void m1()
{
  m2(); // m1() method calling m2().
  System.out.println(1/0); // Exceptional case. A number cannot be divided by zero.
}
public static void m2()
{
  System.out.println("Hello");
 }
}
```


The call stack while m2() is running.
3
m2()
m1 invokes m2
2
m1()
main invokes m1
1
main()
main begins here
The order in which methods are put on the call stack.
Call stack after removing the corresponding entry for m2
Control of execution returns to m1.
1
m1()
2
main()
The order in which execution of methods will complete.
Control of execution returns to main
1
main()
After removing entry for main, JVM hands over exception to Default exception handler.