Object Oriented Programming with Java (Subject Code: BCS-403)

Unit 2

Lecture 14

Lecture 14

- Control Flow in Exceptions
- JVM Reaction to Exceptions
- Use of try, catch and finally

Flow control in try catch finally in Java

1. **Control flow in try-catch clause OR try-catch-finally clause**
    1. **Case 1:** Exception occurs in try block and handled in catch block
    2. **Case 2:** Exception occurs in try-block is not handled in catch block
    3. **Case 3:** Exception doesn't occur in try-block
2. **try-finally clause**
    1. **Case 1:** Exception occurs in try block
    2. **Case 2:** Exception doesn't occur in try-block

Exception occurs in try block and handled in catch block

If a statement in try block raised an exception, then the rest of the try block doesn't execute and control passes to the **corresponding** catch block.

After executing the catch block, the control will be transferred to finally block(if present) and then the rest program will be executed.

Exception occurred in try-block is not handled in catch block

In this case, the default handling mechanism is followed.

If finally block is present, it will be executed followed by the default handling mechanism.

Exception doesn't occur in try-block:

In this case catch block never runs as they are only meant to be run when an exception occurs. finally block(if present) will be executed followed by rest of the program.

```
class A
{
    public static void main (String[] args)
    {
    try
    {
      String str = "123";
       int num = Integer.parseInt(str);
       System.out.println("try block fully executed");
     }
    catch(NumberFormatException ex)
    {
     System.out.println("catch block executed...");
     }
     finally
    {
        System.out.println("finally block executed");
    }
   System.out.println("Outside try-catch-finally clause");
    }
}
```

Control flow in try-finally

In this case, no matter whether an exception occurs in try-block or not finally will always be executed. But control flow will depend on whether an exception has occurred in the try block or not.

1. Exception raised: If an exception has occurred in the try block then the control flow will be finally block followed by the default exception handling mechanism.

```
class A
{
    public static void main (String[] args)
    {
       int[] arr = new int[4];
        try
        {
            int i = arr[4];
            System.out.println("Inside try block");
        }
        finally
        {
            System.out.println("finally block executed");
        }
        // rest program will not execute
        System.out.println("Outside try-finally clause");
    }
}
```

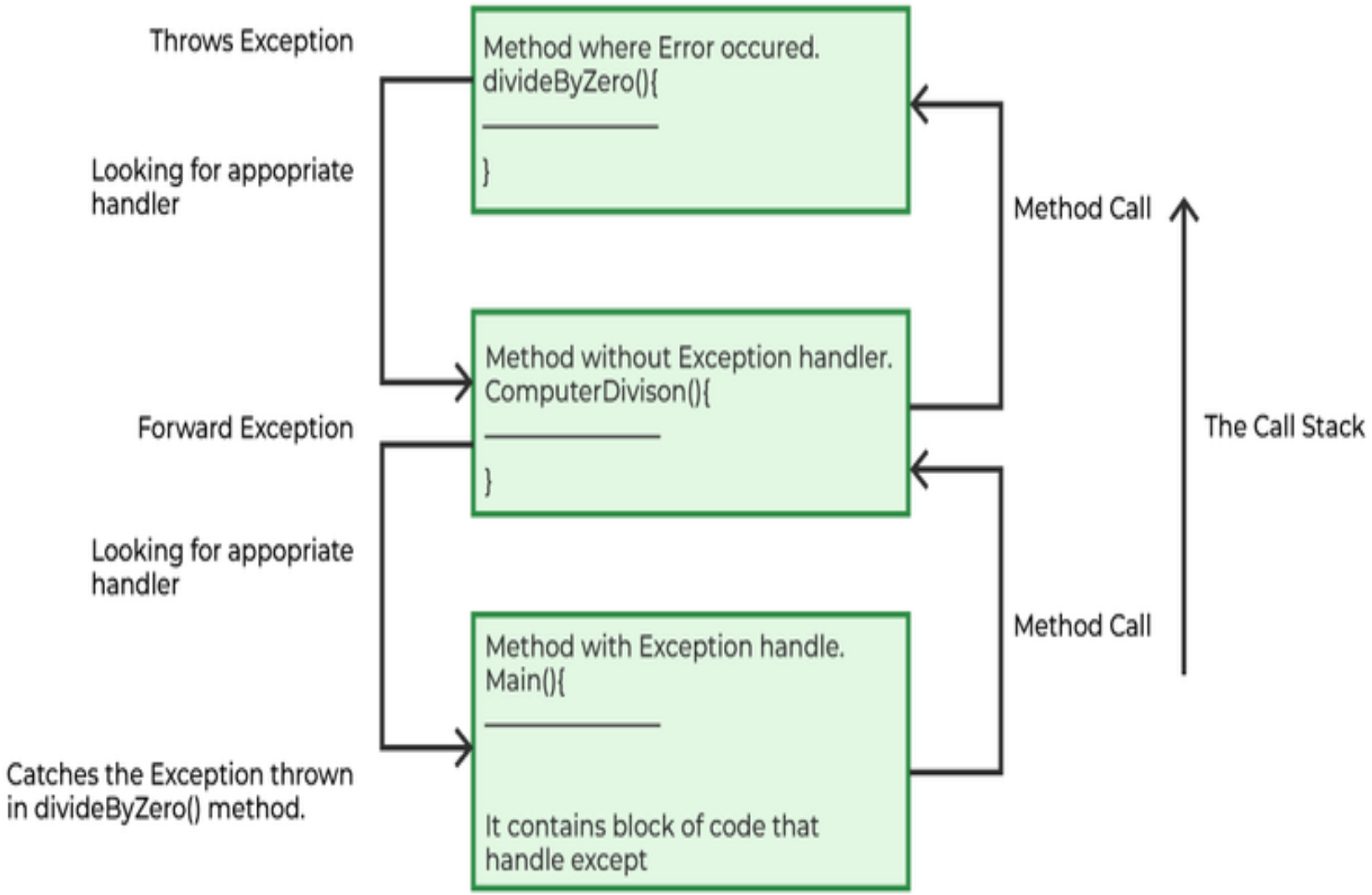
Exception not raised:

If an exception does not occur in the try block then the control flow will be finally block followed by the rest of the program

Default Exception Handling

- Whenever inside a method, if an exception has occurred, the method creates an Object known as an Exception Object and hands it off to the run-time system(JVM).
- The exception object contains the name and description of the exception and the current state of the program where the exception has occurred.
- Creating the Exception Object and handling it in the run-time system is called throwing an Exception.

There might be a list of the methods that had been called to get to the method where an exception occurred. This ordered list of methods is called Call Stack.



The Call Stack and searching the call stack for exception handler.

- The run-time system searches the call stack to find the method that contains a block of code that can handle the occurred exception. The block of the code is called an Exception handler.
- The run-time system starts searching from the method in which the exception occurred and proceeds through the call stack in the reverse order in which methods were called.

.

- If it finds an appropriate handler, then it passes the occurred exception to it. An appropriate handler means the type of exception object thrown matches the type of exception object it can handle.

.

- If the run-time system searches all the methods on the call stack and couldn't have found the appropriate handler, then the run-time system handover the Exception Object to the **default exception handler**, which is part of the run-time system.
- This handler prints the exception information in the following format and terminates the program **abnormally**

Exception in thread "xxx" Name of Exception : Description

... . // Call Stack

Java Exception Keywords

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

```
class example {
   public static void main (String args[]) {
      int num1 = 15, num2 = 0, result = 0;
      try{
          result = num1/num2;
          System.out.println("The result is" +result);
      }
      catch (ArithmeticException e) {
        System.out.println ("Can't be divided by Zero " + e);
      }
   }
}
```

Java Multi-catch block

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.

```
public class MultipleCatchBlock1 {
  public static void main(String[] args) {
          try{
            int a[]=new int[5];
            a[5]=30/0;
           }
           catch(ArithmeticException e)
             {
              System.out.println("Arithmetic Exception occurs");
             }
           catch(ArrayIndexOutOfBoundsException e)
             {
              System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
           catch(Exception e)
             {
              System.out.println("Parent Exception occurs");
             }
           System.out.println("rest of the code");
    }
}
```

```
public class MyMain {
public static void main(String[] args) {
int a[]=new int[4];
try
{
a[0]=12/0;
System.out.println(a[6]);
}
catch(ArithmeticException e)
{
System.out.println("Aritmetic Exception"+e.getMessage());
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Index out of bound"+e.getMessage());
}
catch(Exception e)
{
System.out.println(e.getMessage());
}
finally
{
System.out.println("Finally block executed");
}
System.out.println("outside Finally block executed");
}
}
```