Exception Handling

SMITA SANKHE
Assistant Professor
Department of Computer Engineering

Outline

- Exception as objects
- Exception hierarchy
- try block
- catch block
- finally block
- throw, throws keywords
- User defined exceptions

What is an exception?

- An Exception is an **unwanted event that interrupts the normal flow of the program**
- When an exception occurs program execution gets terminated and we get a system generated error message.
- The good thing about exceptions is that they can be handled in Java.
- **Exception handling** means providing a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user.

Why an exception occurs?

- There can be several reasons that can cause a program to throw exception.
- For example:
  - Opening a non-existing file in your program
  - Network connection problem
  - Bad input data provided by user etc.

Exception Handling

- If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user.

- **An exception generated by the system is given below**

```
Exception in thread "main" java.lang.ArithmeticException: / by zero at ExceptionDemo.main(ExceptionDe
ExceptionDemo : The class name
main : The method name
ExceptionDemo.java : The filename
java:5 : Line number
```

- Thus, **exception handling** allows to handle such exceptions by printing user friendly warning message so that programmer can correct it

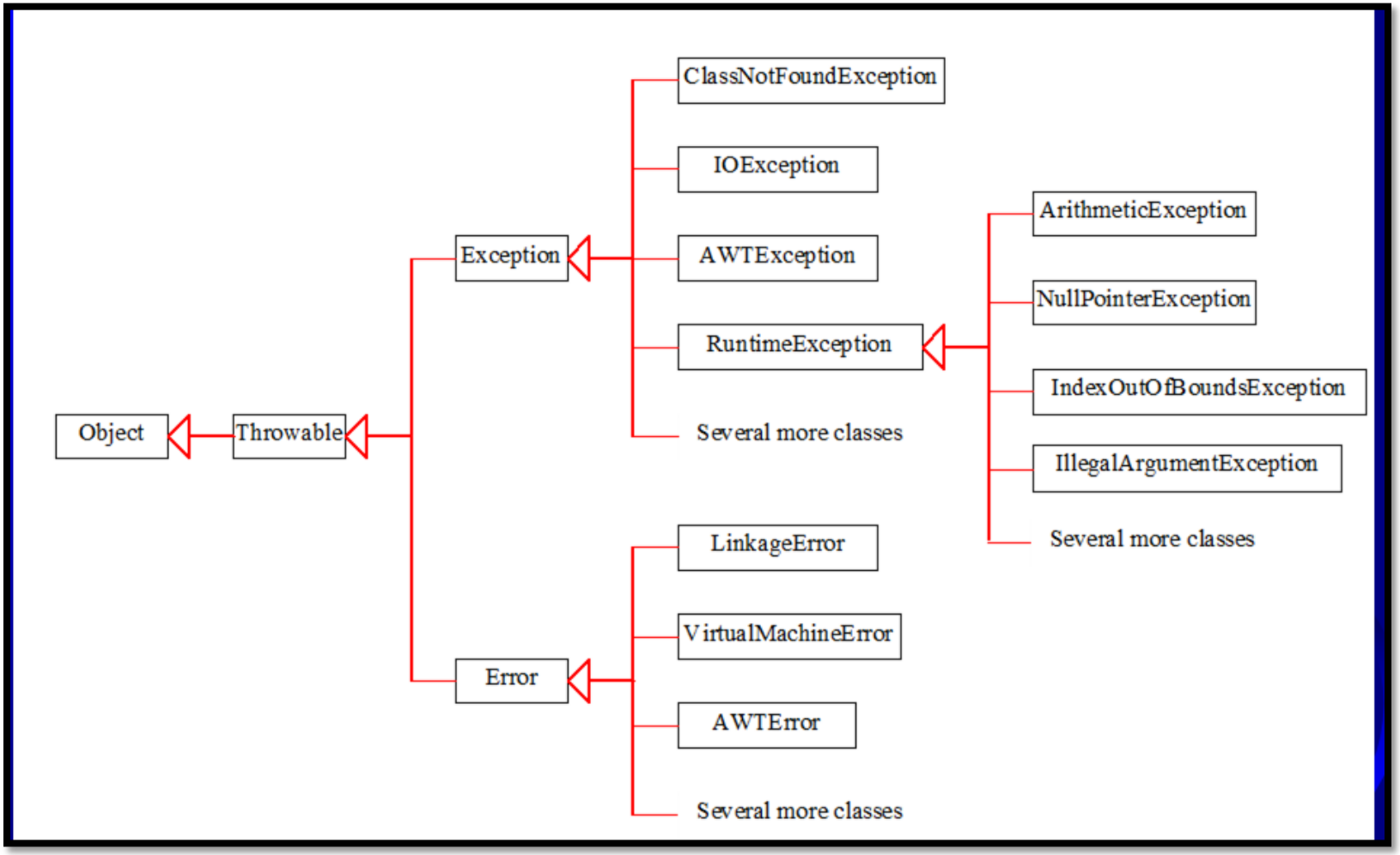
Advantage: Exception handling

- Ensures that the flow of the program doesn't break when an exception occurs

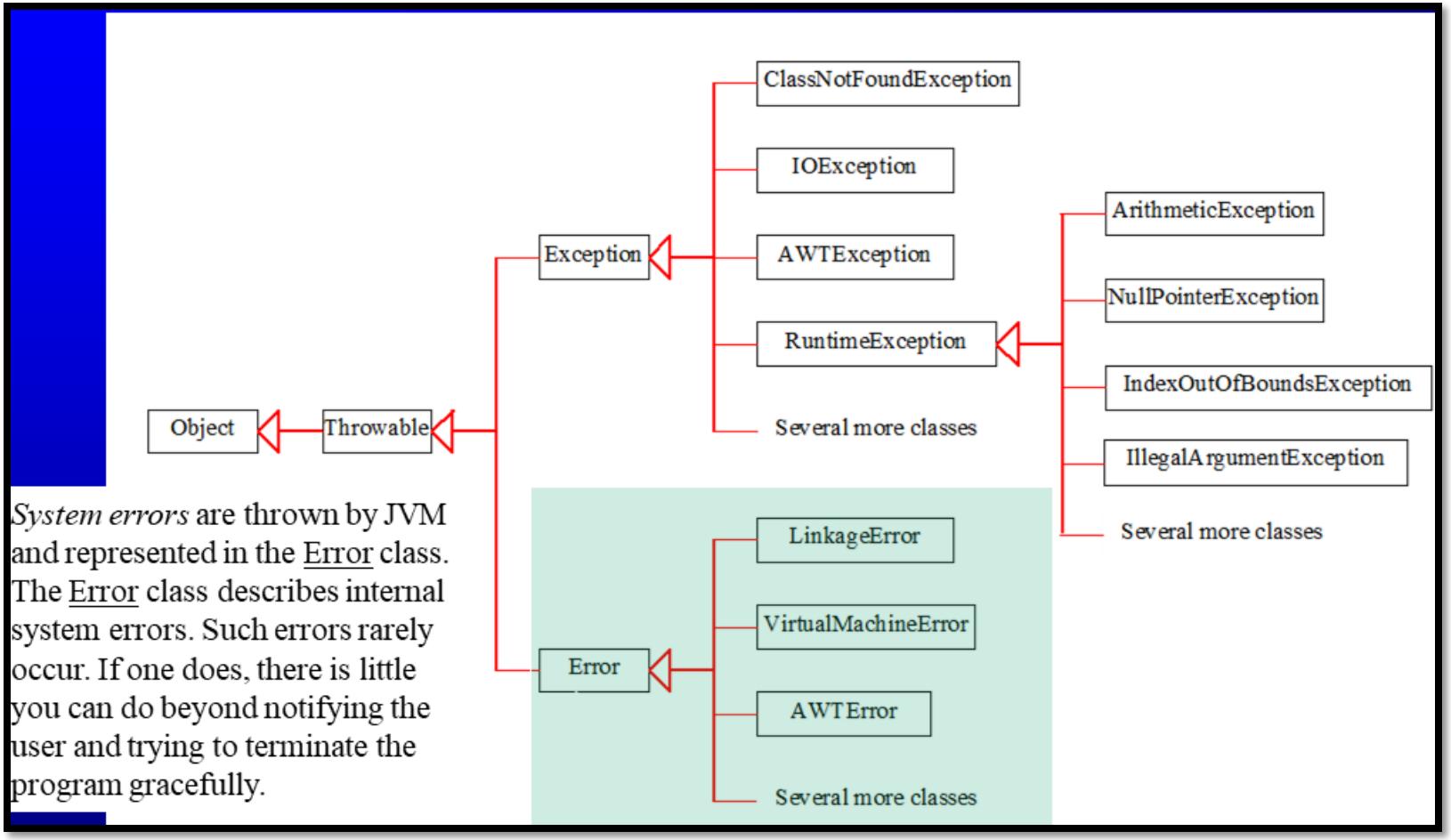
- For example:
  - If a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.
  - By handling exceptions, we make sure that all the statements execute and the flow of program doesn't break.

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Exception Hierarchy

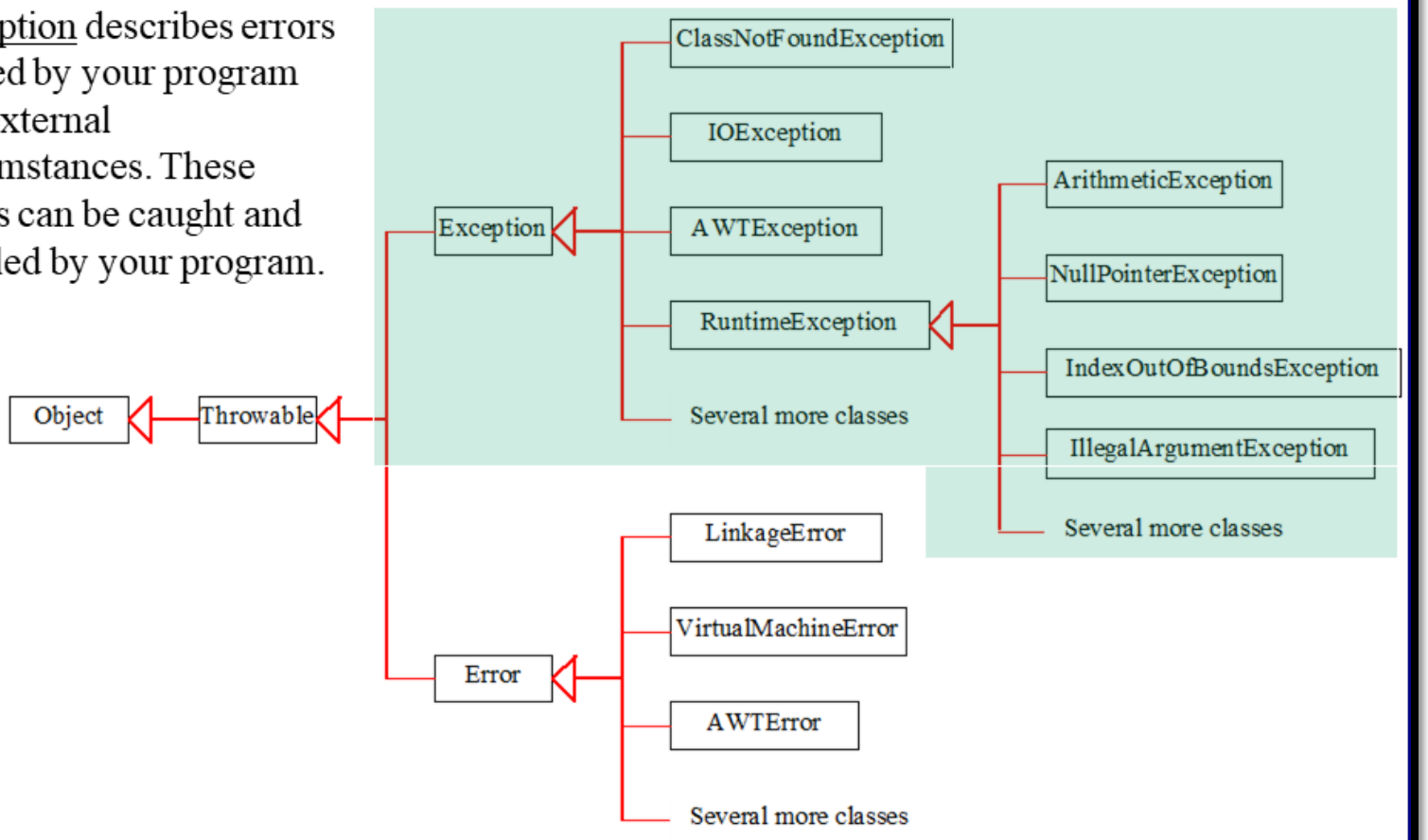


Exception Hierarchy: System Errors

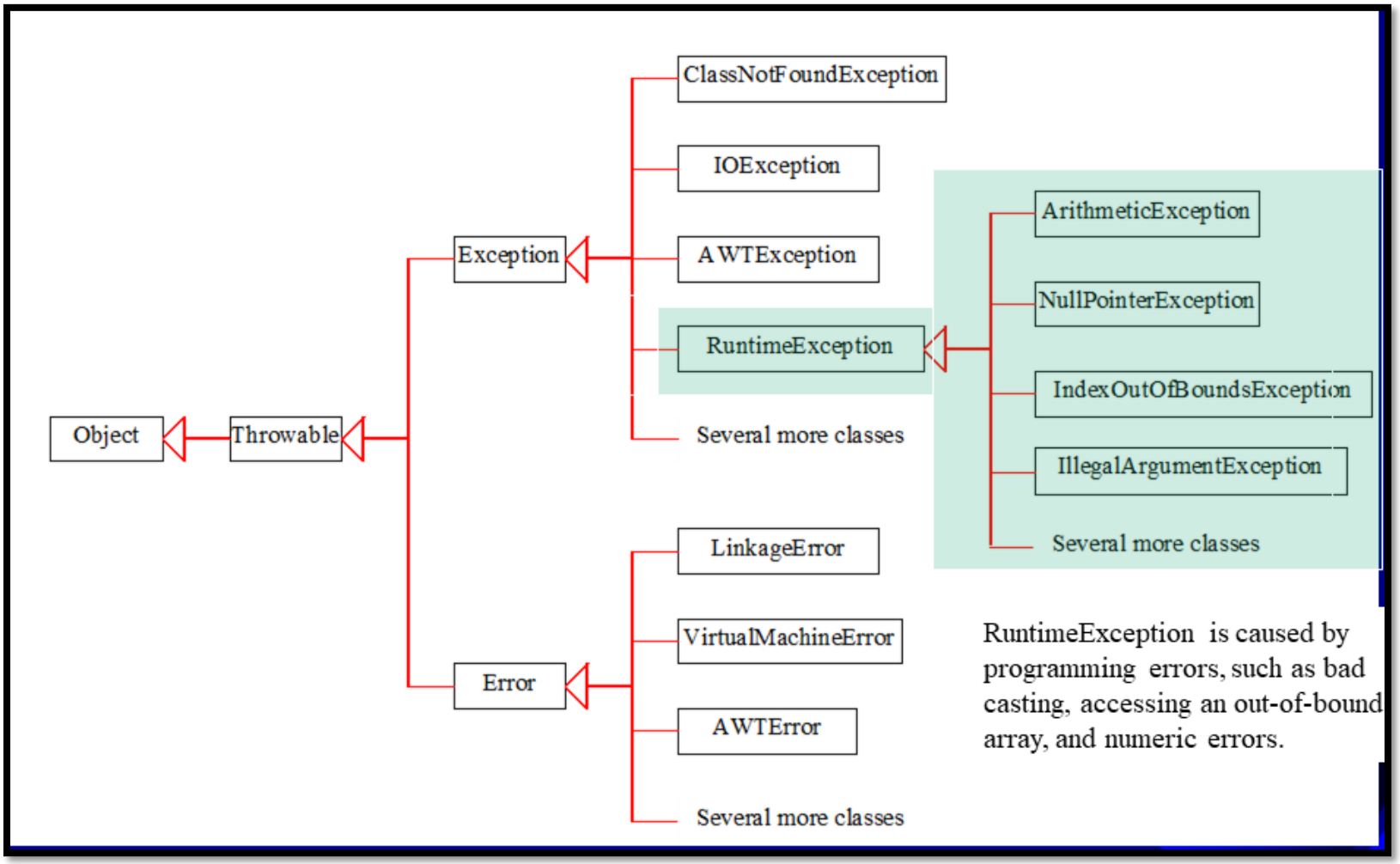


Exception Hierarchy: Exceptions

Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.



Exception Hierarchy: Runtime Exceptions



Types of Exceptions

1. **Checked Exceptions:**

- All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not.
- If these exceptions are not handled/declared in the program, you will get compilation error.
- Example:
    - SQLException
    - IOException
    - ClassNotFoundException etc.

Types of Exceptions

2. **Unchecked Exceptions:**

- These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit.
- **Error, Runtime Exceptions and their subclasses** are known as Unchecked Exceptions
- Example –
  - ArithmeticException
  - NullPointerException
  - ArrayIndexOutOfBoundsException etc

Try Catch in Java

Try block

- The try block contains set of statements where an exception can occur.
- A try block is always followed by a catch block, which handles the exception that occurs in associated try block.
- A try block must be followed by catch blocks or finally block or both.
- can't use try block alone
- **Syntax of try block**

```
try{
        //statements that may cause an exception
    }
```

Try Catch in Java (contd.)

Catch block

- A catch block is where you handle the exceptions, this block must follow the try block.
- A single try block ***can have several catch blocks*** associated with it.
- You can catch different exceptions in different catch blocks.
- When an exception occurs in try block, the corresponding catch block that handles that particular exception executes.
- For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.
- can't use catch block alone

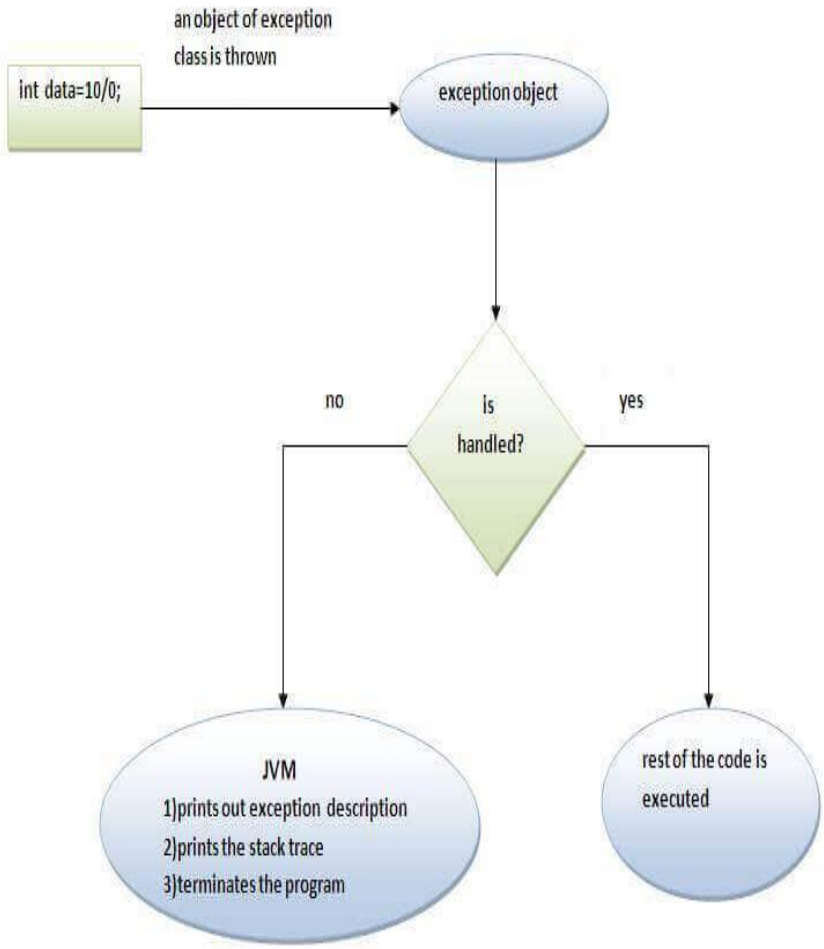
Syntax for try catch in java

```
try
{
        //statements that may cause an exception
 }

catch (exception(type) e(object))
 {
        //error handling code
 }
```

Example:

```
1  public class JavaExceptionExample
2  {
3      public static void main(String args[]){
4      try{
5          //code that may raise exception
6          int data=100/0;
7          }
8        catch(ArithmeticException e)
9          {     System.out.println(e);
10         }
11       //rest code of the program
12       System.out.println("rest of the code...");
13     }
14 }
```




```
java.lang.ArithmeticException: / by zero
rest of the code...
```

Common Scenarios of Java Exceptions

1. **A scenario where ArithmeticException occurs**

- occurs when an attempt is made to divide two numbers and the number in the denominator is zero

- **Ex:**

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

1. Arithmetic Exception Example

```
1  public class MyClass {
2      public static void main(String args[]) {
3       try{
4           int data = 50/0;
5           System.out.println("rest of code");
6       }
7      catch(Exception e)
8      {
9         System.out.println(e);
10     }
11
12   }
13 }
```

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

1. Arithmetic Exception Example(custom message)

```
1  public class MyClass {
2      public static void main(String args[]) {
3       try{
4           int data = 50/0;  //may throw exception
5           System.out.println("rest of code");
6       }
7       //handling exception
8      catch(Exception e)
9      {
10        System.out.println("Cannot be divided by zero");
11     }
12   }
13  }
14
```

```
Cannot be divided by zero
```

Common Scenarios of Java Exceptions

2. **A scenario where NullPointerException occurs**

- NullPointerException is thrown when program attempts to use an object reference that has the null value
- Invoking a method from a null object.
- Accessing or modifying a null object's field.
- Taking the length of null, as if it were an array.
- Accessing or modifying the slots of null object, as if it were an array.

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

2. NullPointerException Example

```java
import java.io.*;

public class GFG
{
    public static void main (String[] args)
    {
        // Initializing String variable with null value
        String ptr = null;

        // Checking if ptr.equals null or works fine.
        try
        {
            // This line of code throws NullPointerException
            // because ptr is null
            if (ptr.equals("gfg"))
                System.out.print("Same");
            else
                System.out.print("Not Same");
        }
        catch(NullPointerException e)
        {
            System.out.print("NullPointerException Caught");
        }
    }
}
```

```
NullPointerException Caught
```

Common Scenarios of Java Exceptions

3. A scenario where NumberFormatException occurs

Since NumberFormatException occurs due to the inappropriate format of string for the corresponding argument of the method which is throwing the exception, there can be various ways of it. A few of them are mentioned as follows-

- The input string provided might be null-
  Example- Integer.parseInt(null);
- The input string might be empty-
  Example- Integer.parseInt("");
- The input string might be having trailing space-
  Example- Integer.parseInt("123 ");
- The input string might be having a leading space-
  Example- Integer.parseInt(" 123");
- The input string may be alphanumeric-
  Example- Long.parseLong("b2");

NumberFormatException Example

```
public class Example {


    public static void main(String[] args) {
        int a = Integer.parseInt(null); //throws Exception as
    }


}
```

```
Exception in thread "main" java.lang.NumberFormatException: null
        at java.base/java.lang.Integer.parseInt(Unknown Source)
        at java.base/java.lang.Integer.parseInt(Unknown Source)
        at crux19aug.Crux19Aug2018.src.assignments.Example.main(Example.java:14)
```

NumberFormatException Example

```
1  public class NumberFormatExceptionExample {
2
3      private static final String inputString = "123.33";
4
5      public static void main(String[] args) {
6          try {
7                  int a = Integer.parseInt(inputString);
8          }
9
10         catch(NumberFormatException ex){
11             System.err.println("Invalid string in argumment");
12             //request for well-formatted string
13         }
14      }
15 }
```

```
Invalid string in argumment
```

Common Scenarios of Java Exceptions

4. **A scenario where ArrayIndexOutOfBoundsException occurs**

- The ArrayIndexOutOfBoundsException occurs whenever we are trying to access any item of an array at an index which is not present in the array.
- In other words, the index may be negative or exceed the size of an array.

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

ArrayIndexOutOfBound Exception Example

```
class Main {
    public static void main(String[] args) {
        //array of subjects. There are 5 elements.
        String[] subjects = {"Maths","Science","French","Sanskrit", "English"};

        //for loop iterates from 0 to 5 (length of array)
        for(int i=0;i<=subjects.length;i++) {
            //when 'i' reaches 5, it becomes invalid index and exception will be thrown
            System.out.print(subjects[i] + " ");
        }
    }
```

Output:

```
Maths Science French Sanskrit English Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5

        at Main.main(Main.java:9)
```

ArrayIndexOutOfBound Exception Example Solution

```
class Main {
   public static void main(String[] args) {
         //array of subjects. There are 5 elements.
         String[] subjects = {"Maths","Science","French","Sanskrit", "English"};

         System.out.println("")
        //define enhanced for loop to iterate over array
        for(String strval:subjects) {
            //iterates only through valid indices
            System.out.print(strval + " ");
        }
    }
}
```

Output:

```
Maths Science  French Sanskrit English
```

Multiple catch blocks in Java

- A single try block can have any number of catch blocks.
- A generic catch block can handle all the exceptions.

Whether it is ArrayIndexOutOfBoundsException or ArithmeticException or NullPointerException or any other type of exception, this handles all of them.

- If no exception occurs in try block then the catch blocks are completely ignored.
- Corresponding catch blocks execute for that specific type of exception:
  - catch(ArithmeticException e) is a catch block that can handle ArithmeticException
  - catch(NullPointerException e) is a catch block that can handle NullPointerException

Example: 1

```
1  public class MultipleCatchBlock1{
2    public static void main(String[] args){
3      try{
4          int a[]=new int[7];
5           a[5]=30/0;
6          }
7      catch(ArrayIndexOutOfBoundsException e)
8      {
9      System.out.println("ArrayIndexOutOfBounds Exception occurs");
10     }
11     catch(ArithmeticException e)
12     {
13     System.out.println("Arithmetic Exception occurs");
14     }
15     catch(Exception e)
16     {
17     System.out.println("Parent Exception occurs");
18     }
19 System.out.println("rest of the code");
20 }
21 }
```

```
Arithmetic Exception occurs
rest of the code
```

Example: 2 (more specific to general)

```
1  public class MultipleCatchBlock1{
2    public static void main(String[] args){
3      try{
4          String s=null;
5          System.out.println(s.length());
6          }
7      catch(ArrayIndexOutOfBoundsException e)
8      {
9      System.out.println("ArrayIndexOutOfBounds Exception occurs");
10     }
11     catch(ArithmeticException e)
12     {
13     System.out.println("Arithmetic Exception occurs");
14     }
15     catch(Exception e)
16     {
17     System.out.println("Parent Exception occurs");
18     }
19 System.out.println("rest of the code");
20 }
21 }
```

```
Parent Exception occurs
rest of the code
```

Example:3 (general to more specific)

```
1  public class MultipleCatchBlock1{
2    public static void main(String[] args){
3      try{
4          int a[]=new int[5];
5          a[5]=30/0;
6          }
7      catch(Exception e)
8      {
9      System.out.println("Parent Exception occurs");
10     }
11     catch(ArrayIndexOutOfBoundsException e)
12     {
13     System.out.println("ArrayIndexOutOfBounds Exception occurs");
14     }
15     catch(ArithmeticException e)
16     {
17     System.out.println("Arithmetic Exception occurs");
18     }
19 System.out.println("rest of the code");
20 }
21 }
```

```
/MultipleCatchBlock1.java:11: error: exception ArrayIndexOutOfBoundsException has already been caught
    catch(ArrayIndexOutOfBoundsException e)
    ^
/MultipleCatchBlock1.java:15: error: exception ArithmeticException has already been caught
    catch(ArithmeticException e)
    ^
2 errors
```

Nested try catch block in Java

- When a try catch block is present in another try block then it is called the nested try catch block.
- Each time a try block does not have a catch handler for a particular exception, then the catch blocks of parent try block are inspected for that exception, if match is found that that catch block executes.
- If neither catch block nor parent catch block handles exception then the system generated message would be shown for the exception, similar to what we see when we don’t handle exception.

Syntax: Nested try catch block

```
Main try block
try {   statement 1;
        //try-catch block inside another try block
        try {
          statement 3;
              //try-catch block inside nested try block
               try {
                 statement 5;
                    }
               catch(Exception e2) {  //Exception Message
               }
          }
          catch(Exception e1) {//Exception Message
          }


    }
//Catch of Main(parent) try block
catch(Exception e3) { //Exception Message
}
```

Example:

```
1  public class Excep6{
2   public static void main(String args[]){
3   try{
4       try{
5          System.out.println("going to divide");
6          int b=39/0;
7          }
8       catch(ArithmeticException e)
9       {
10         System.out.println(e);
11      }
12      try{
13         int a[]=new int[5];
14         a[5]=4;
15        }
16      catch(ArrayIndexOutOfBoundsException e)
17      {
18         System.out.println(e);
19      }
20    System.out.println("other statement");
21  }
22  catch(Exception e)
23  {
24      System.out.println("handeled");
25  }
26  System.out.println("normal flow..");
27  }
28  }
```

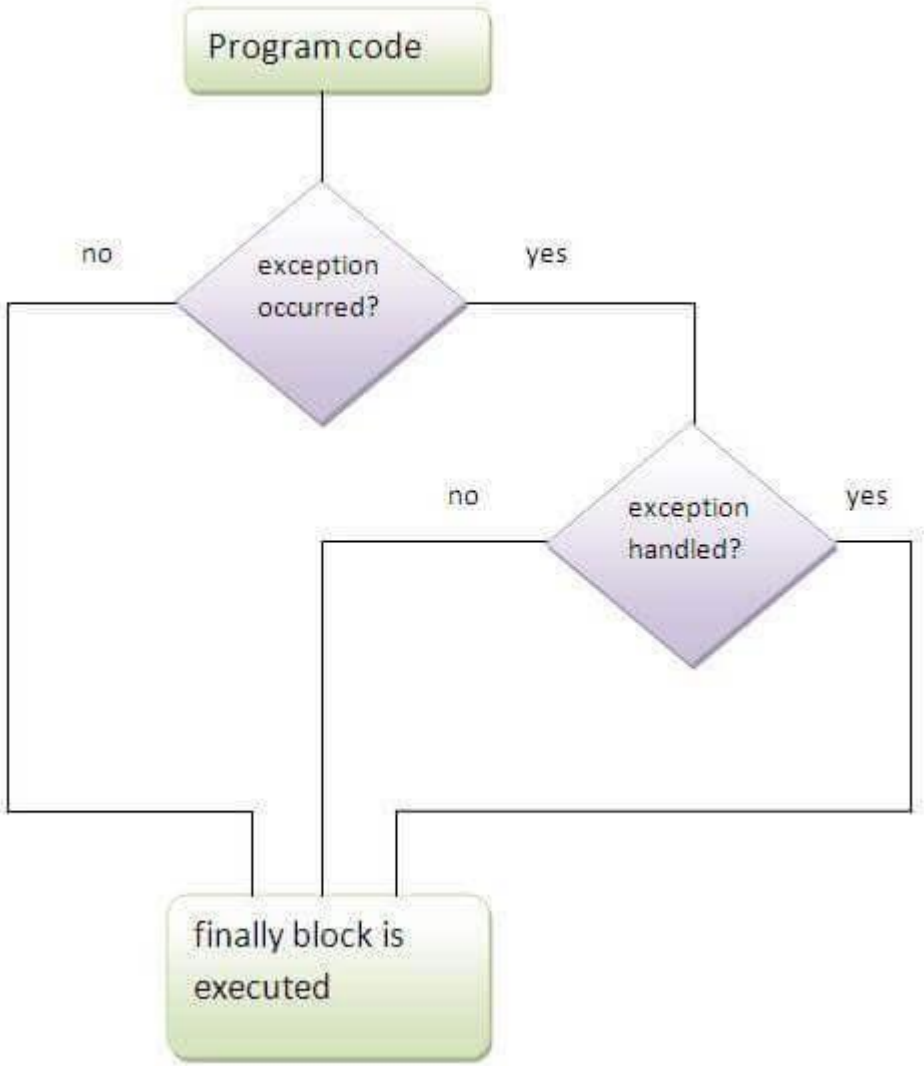
```
going to divide
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for leng
other statement
normal flow..
```

Java Finally block

- Contains all the **crucial statements that must be executed whether exception occurs or not.**
- The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream , file etc.
- **Syntax:**

```
try {
        //Statements that may cause an exception
    }
catch {
        //Handling exception
    }
finally {
          //Statements to be executed
    }
```

Working of java finally block



Example: 1

```
1  public class TestFinallyBlock
2  {
3   public static void main(String args[]){
4    try{
5        int data=25/5;
6        System.out.println(data);
7       }
8    catch(NullPointerException e)
9    {
10       System.out.println(e);
11   }
12   finally{
13       System.out.println("finally block is always executed");
14        }
15 System.out.println("rest of the code...");
16  }
17 }
```

```
5
finally block is always executed
rest of the code...
```

Example: 2

```
1  public class TestFinallyBlock
2  {
3   public static void main(String args[]){
4    try{
5        int data=25/0;
6        System.out.println(data);
7       }
8    catch(NullPointerException e)
9    {
10       System.out.println(e);
11   }
12   finally{
13       System.out.println("finally block is always executed");
14        }
15 System.out.println("rest of the code...");
16  }
17 }
18
```

```
finally block is always executed

Exception in thread "main" java.lang.ArithmeticException: / by zero
    at TestFinallyBlock.main(TestFinallyBlock.java:5)
```

Example: 3

```
1  public class TestFinallyBlock
2  {
3   public static void main(String args[]){
4    try{
5        int data=25/0;
6        System.out.println(data);
7       }
8    catch(ArithmeticException e)
9    {
10       System.out.println(e);
11   }
12   finally{
13       System.out.println("finally block is always executed");
14        }
15 System.out.println("rest of the code...");
16  }
17 }
18
```

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

Important points regarding finally block

1. **A finally block must be associated with a try block, you cannot use finally without a try block**. You should place those statements in this block that must be executed always.
2. Finally block is optional, however if you place a finally block then it will always run after the execution of try block.
3. In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.
4. The statements present in the **finally block** execute even if the try block contains control transfer statements like return, break or continue.

throw keyword

- In Java we have already defined exception classes such as ArithmeticException, NullPointerException, ArrayIndexOutOfBounds exception etc.
- We can define our **own set of conditions or rules and throw an exception explicitly using throw keyword.**
- For example, we can throw ArithmeticException when we divide number by 0, or any other numbers, what we need to do is just set the condition and throw any exception using throw keyword.
- Throw keyword can also be used for throwing custom exceptions
- **Syntax of throw keyword:**

```
throw new exception_class("error message");
```

Example:1

```
1  public class TestThrow1
2  {
3   static void validate(int age){
4    if(age<18)
5     throw new ArithmeticException("not valid");
6    else
7     System.out.println("welcome to vote");
8    }
9   public static void main(String args[]){
10   validate(13);
11   System.out.println("rest of the code...");
12  }
13 }
```

```
Exception in thread "main" java.lang.ArithmeticException: not valid
    at TestThrow1.validate(TestThrow1.java:5)
    at TestThrow1.main(TestThrow1.java:10)
```

Example:2

```
public class ThrowExample {
   static void checkEligibilty(int stuage, int stuweight){
      if(stuage<12 && stuweight<40) {
         throw new ArithmeticException("Student is not eligible for registration");
      }
      else {
         System.out.println("Student Entry is Valid!!");
      }
   }

   public static void main(String args[]){
     System.out.println("Welcome to the Registration process!!");
     checkEligibilty(10, 39);
     System.out.println("Have a nice day..");
 }
}
```

```
Welcome to the Registration process!!

Exception in thread "main" java.lang.ArithmeticException: Student is not eligible for registration
    at ThrowExample.checkEligibilty(ThrowExample.java:4)
    at ThrowExample.main(ThrowExample.java:13)
```

User-defined Custom Exception in Java

- In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.
- Using the custom exception, we can have your own exception and message.
- Here, we have to passed a string to the constructor of superclass i.e. Exception class that can be obtained using getMessage() method on the object we have created.

Why use custom exceptions?

- Java exceptions cover almost all the general type of exceptions that may occur in the programming. However, we sometimes need to create custom exceptions.
- Following are few of the reasons to use custom exceptions:
- To catch and provide specific treatment to a subset of existing Java exceptions.
- Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.
- In order to create custom exception, we need to extend Exception class that belongs to java.lang package.

```java
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
        super(str);
    }
}

public class TestCustomException1
{

    // method to check the age
    static void validate (int age) throws InvalidAgeException{
       if(age < 18){

        throw new InvalidAgeException("age is not valid to vote");
    }
       else {
        System.out.println("welcome to vote");
        }
     }

    public static void main(String args[])
    {
        try
        {

            validate(13);
        }
        catch (InvalidAgeException ex)
        {
            System.out.println("Caught the exception");

            System.out.println("Exception occured: " + ex);
        }

        System.out.println("rest of the code...");
    }
}
```

```java
public class TestThrow {
   //defining a method
   public static void checkNum(int num) {
      if (num < 1) {
         throw new ArithmeticException("\nNumber is negative, cannot calculate square");
      }
      else {
         System.out.println("Square of " + num + " is " + (num*num));
      }
   }
   //main method
   public static void main(String[] args) {
         TestThrow obj = new TestThrow();
         obj.checkNum(-3);
         System.out.println("Rest of the code..");
   }
}
```

```
Exception in thread "main" java.lang.ArithmeticException:
Number is negative, cannot calculate square
        at TestThrow.checkNum(TestThrow.java:6)
        at TestThrow.main(TestThrow.java:16)
```

```java
public class TestThrowAndThrows
{
   // defining a user-defined method
   // which throws ArithmeticException
   static void method() throws ArithmeticException
   {
      System.out.println("Inside the method()");
      throw new ArithmeticException("throwing ArithmeticException");
   }
   //main method
   public static void main(String args[])
   {
      try
      {
         method();
      }
      catch(ArithmeticException e)
      {
         System.out.println("caught in main() method");
      }
   }
}
```

```
Inside the method()
caught in main() method
```

Difference between throw and throws

throw	throws
Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
throw is used within the method.	throws is used with the method signature.
We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.