Welcome you all

JAVA PROGRAMMING

DAY : 7

D.Sakthivel
Assistant Professor & Trainer,
KGiSL Micro College
KGiSL Campus, Coimbatore – 641 035.

Day 7

Exception Handling

- Exceptions Overview
- Catching Exceptions
- The finally Block
- Exception Methods
- Defining and Throwing Exceptions
- Errors and Runtime Exceptions

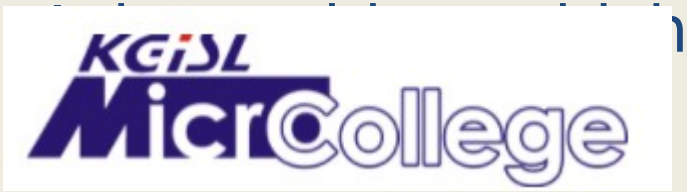
Exception Handling:

- Exception handling is the process of handling the errors and nonconforming states, while the program is executed.
- *Mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

What is Exception in Java?

Dictionary Meaning: Exception is an abnormal condition.

- In Java, **an exception is an event that disrupts the normal flow of the program**.
- [illegible] is thrown at runtime.



What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage of Exception Handling

- The core advantage of exception handling is **to maintain the normal flow of the application**.
- An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

Let's consider a scenario:

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

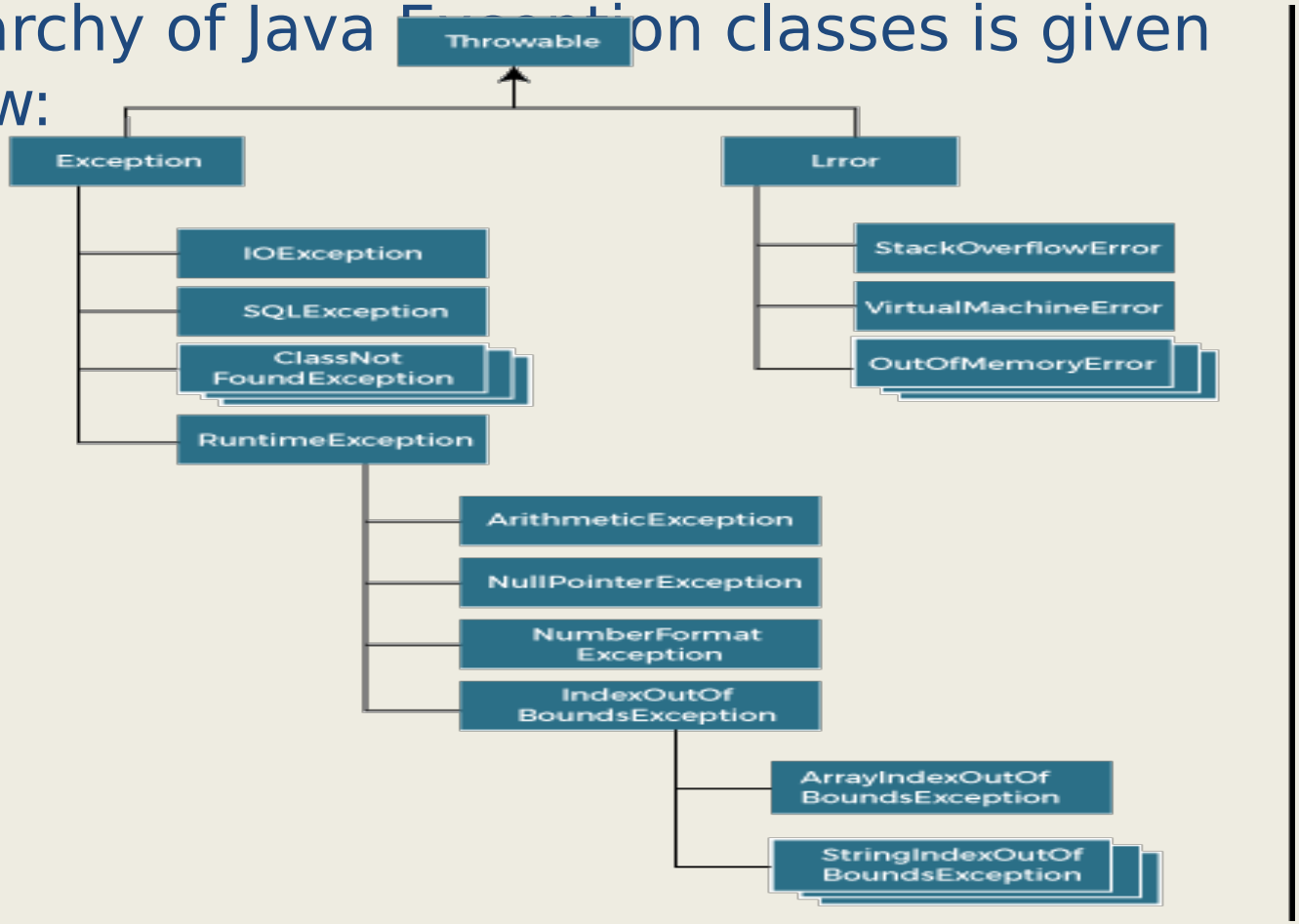
statement 9;

statement 10;

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

Hierarchy of Java Exception classes

The **java.lang.Throwable class** is the root class of Java Exception hierarchy inherited by two subclasses: **Exception and Error**. The hierarchy of Java Exception classes is given below:

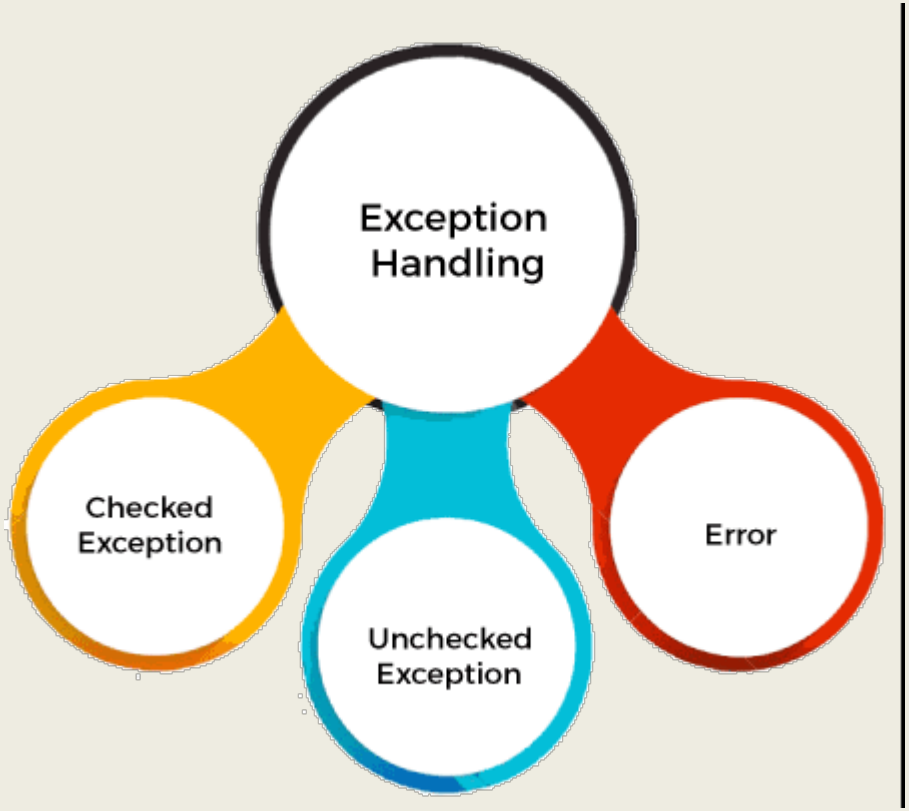


Types of Java Exceptions

There are mainly two types of exceptions: **checked and unchecked.**

An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

- Checked Exception
- Unchecked Exception
- Error



Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions.

For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions.

For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

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.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Java Exception Handling Example

```
public class JavaExceptionExample{
  public static void main(String args[])
{
  try{
      //code that may raise exception
      int data=100/0;
   }
catch(ArithmeticException e)
{
      System.out.println(e);
}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

1) A scenario where **ArithmeticException occurs**

If we divide any number by zero, there occurs an ArithmeticException.

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

2) A scenario where **NullPointerException occurs**

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

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

3) A scenario where **NumberFormatException** occurs
If the formatting of any variable or number is mismatched, it may result into NumberFormatException.
Suppose we have a string variable that has characters; **converting this variable into digit** will cause NumberFormatException.

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

4) A scenario where **ArrayIndexOutOfBoundsException** occurs
When an array exceeds to it's size, the **ArrayIndexOutOfBoundsException** occurs.
there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

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

Java try-catch block

Java try block

- Java **try** block is used to enclose the code that might throw an exception.
- It must be used within the method.
- If an exception occurs at the particular statement in the try block, the rest of the block code will not execute.
- So, it is recommended not to keep the code in try block that will not throw an exception.
- Java try block must be followed by either catch or finally block.

Syntax of Java try-catch

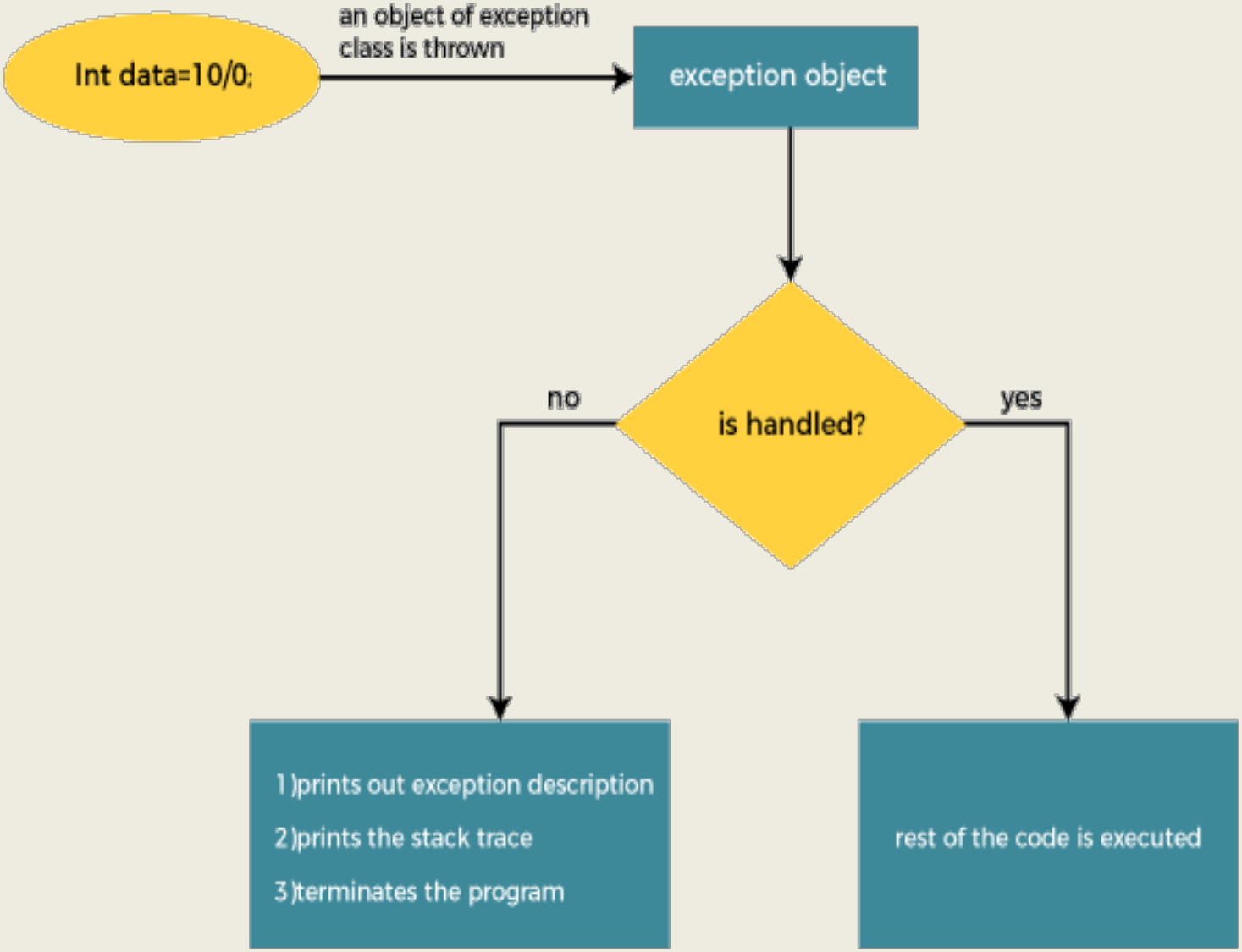
```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

Syntax of try-finally block

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

Java catch block

- Java catch block is used to handle the Exception by declaring the type of exception within the parameter.
- The declared exception must be the parent class exception (i.e., Exception) or the generated exception type.
- However, the good approach is to declare the generated type of exception.
- The catch block must be used after the try block only.
- You can use multiple catch block with a single try block.


Int data=10/0;
an object of exception class is thrown
exception object
is handled?
no
yes
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

```
public class TryCatchExample1 {

    public static void main(String[] args) {

        int data=50/0; //may throw exception

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

    }

}
```

Output:

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

Solution by exception handling

```
public class TryCatchExample2 {

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

}
```

Output:

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

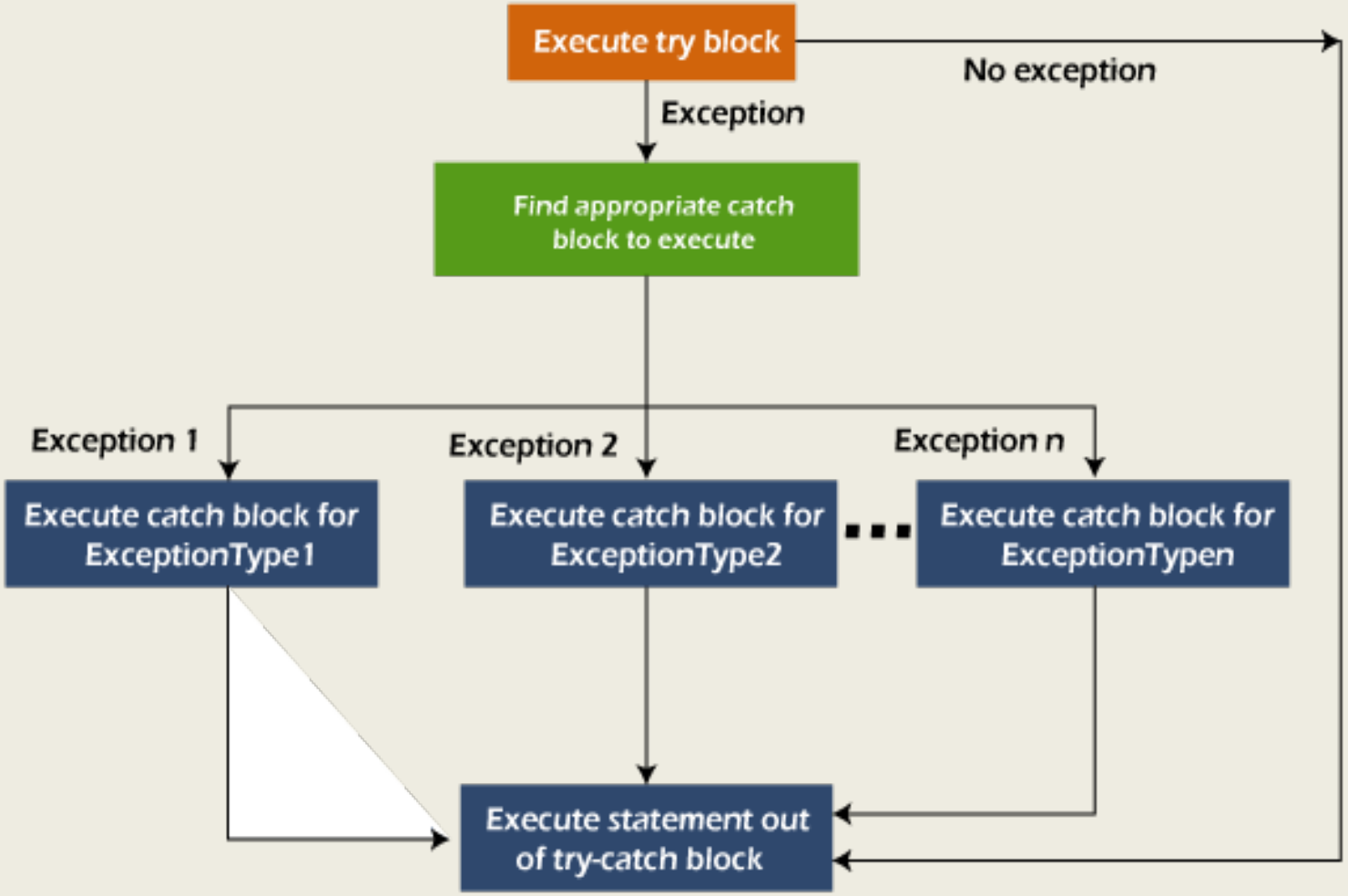
Java Multi-catch block

- **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.

Points to remember

- 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.

Java Multi-catch block



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 Exceptio
n occurs");
              }
           catch(Exception e)
              {
               System.out.println("Parent Exception occurs");
              }
           System.out.println("rest of the code");
    }
```

Output:

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

Java Multi-catch block

```
public class MultipleCatchBlock2 {
    public static void main(String[] arg
        try{
            int a[]=new int[5];
            System.out.println(a[10]);
           }
           catch(ArithmeticException e)
              {
               System.out.println("Arithmetic Exception occurs");
              }
           catch(ArrayIndexOutOfBoundsException e)
              {
               System.out.println("ArrayIndexOutOfBounds Exceptio
n occurs");
              }
           catch(Exception e)
              {
               System.out.println("Parent Exception occurs");
              }
           System.out.println("rest of the code");
    }
```

Output:

```
ArrayIndexOutOfBounds Exception occurs
rest of the code
```

Java Multi-catch block

```
public class MultipleCatchBlock4 {
    public static void main(String[] args) {
        try{
            String s=null;
            System.out.println(s.length());
           }
           catch(ArithmeticException e)
              {
               System.out.println("Arithmetic Exception occurs");
              }
           catch(ArrayIndexOutOfBoundsException e)
              {
               System.out.println("ArrayIndexOutOfBounds Exceptio
n occurs");
              }
           catch(Exception e)
              {
               System.out.println("Parent Exception occurs");
              }
           System.out.println("rest of the code");
    }
```

Output:

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

Java Nested try block

- In Java, using a try block inside another try block is permitted.
- It is called as nested try block.
- Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

For example, the **inner try block** can be used to
handle **ArrayIndexOutOfBoundsException** while the **outer try block** can handle
the **ArithemeticException** (division by zero).

Java Nested try block

Syntax:

```
//main try block
try
{
   statement 1;
   statement 2;
       //try catch block within another try block
        try
        {
                statement 3;
               statement 4;
               //try catch block within nested try block
                try
               {
                        statement 5;
                              statement 6;
               }
               catch(Exception e2)
               {
                       //exception message
                }

       }
       catch(Exception e1)
        {
               //exception message
       }
}
//catch block of parent (outer) try block
catch(Exception e3)
{
//exception message
}
```

Java Nested try block

```
public class NestedTryBlock
{
    public static void main(String args[])
    {
        //outer try block
    try
    {
         //inner try block 1
        try
        {
        System.out.println("going to divide by 0");
        int b =39/0;
        }
        //catch block of inner try block 1
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
```

Java Nested try block

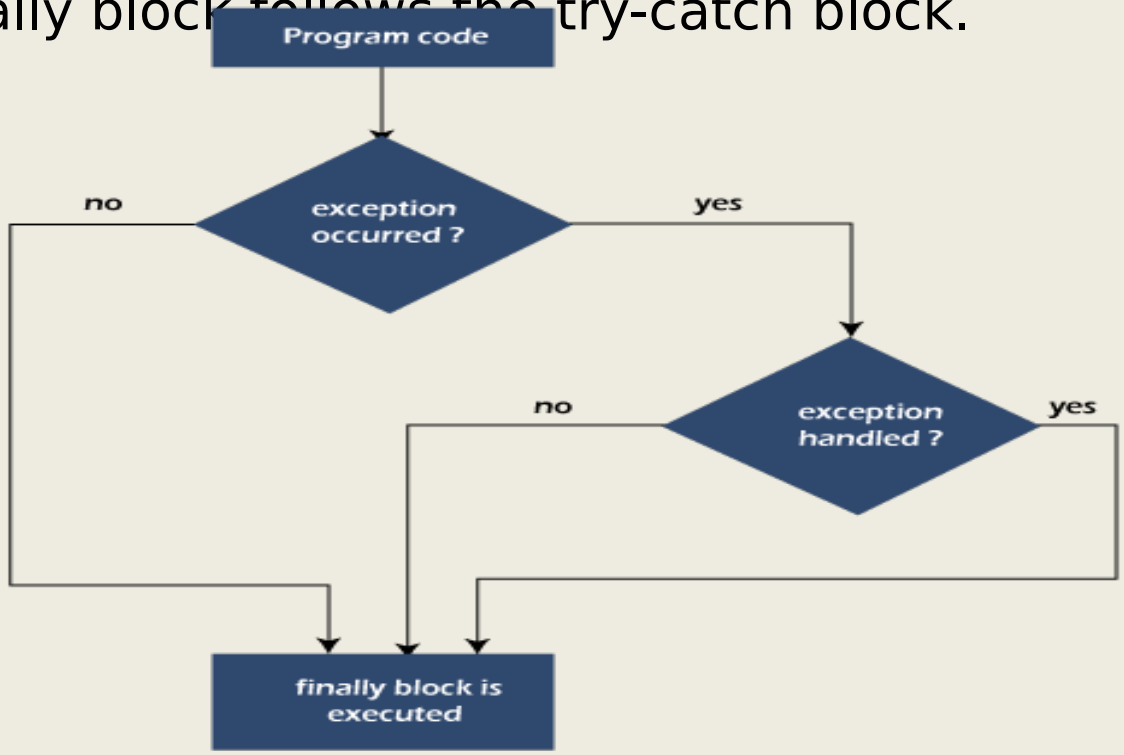
```
//inner try block 2
    try{
    int a[]=new int[5];
    //assigning the value out of array bo
     a[5]=4;
     }
      //catch block of inner try block 2
    catch(ArrayIndexOutOfBoundsException e)
    {
       System.out.println(e);
    }

    System.out.println("other statement");
  }
  //catch block of outer try block
  catch(Exception e)
  {
    System.out.println("handled the exception (outer catch)");
  }
    System.out.println("normal flow..");
 }
}
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac NestedTryBlock.java

C:\Users\Anurati\Desktop\abcDemo>java NestedTryBlock
going to divide by 0
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
other statement
normal flow..
```

Java finally block

- **Java finally block** is a block used to execute important code such as closing the connection, etc.
- Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.
- The finally block follows the try-catch block.



Why use Java finally block?

- finally block in Java can be used to put "**cleanup**" code such as closing a file, closing connection, etc.
- The important statements to be printed can be placed in the finally block.

```
class TestFinallyBlock {
  public static void main(String args[]){
  try{
//below code do not throw any exce
   int data=25/5;
   System.out.println(data);
  }
//catch won't be executed
  catch(NullPointerException e){
System.out.println(e);
}
//executed regardless of exception occurred or not
 finally {
System.out.println("finally block is always executed");
}

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

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestFinallyBlock.java

C:\Users\Anurati\Desktop\abcDemo>java TestFinallyBlock
5
finally block is always executed
rest of the code...
```

Java throw Exception

In Java, exceptions allows us to write good quality codes where the errors are checked at the compile time instead of runtime and we can create custom exceptions making the code recovery and debugging easier.

Java throw keyword

- The Java throw keyword is used to throw an exception explicitly.
- We specify the **exception** object which is to be thrown.
- The Exception has some message with it that provides the error description.
- These exceptions may be related to user inputs, server, etc.
- We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception. We will discuss custom exceptions later in this section.
- We can also define our own set of conditions and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException if we divide a number by another number.

Java throw Exception

The syntax of the Java throw keyword is given below.

throw Instance i.e.,

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

Let's see the example of throw IOException.

```
throw new IOException("sorry device error");
```

Where the Instance must be of type Throwable or subclass of Throwable. For example, Exception is the sub class of Throwable and the user-defined exceptions usually extend the Exception class.

Java throw keyword

```
public class TestThrow1 {
   //
function to check if person is eligible to vote or not

   public static void validate(int age) {
      if(age<18) {
         //
throw Arithmetic exception if not eligible to vote
         throw new ArithmeticException("Person is
not eligible to vote");
      }
      else {
         System.out.println("Person is eligible to vote
!!");
      }
   }
   //main method
   public static void main(String args
      //calling the function
      validate(13);
      System.out.println("rest of the code...");
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow1.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrow1
Exception in thread "main" java.lang.ArithmeticException: Person is not eligible t
 vote
        at TestThrow1.validate(TestThrow1.java:8)
        at TestThrow1.main(TestThrow1.java:18)
```

Throwing User-defined Exception

```
// class represents user-defined exception
class UserDefinedException extends Exception
{
    public UserDefinedException(String str)
    {
        // Calling constructor of parent Exception
        super(str);
    }
}
// Class that uses above MyException
public class TestThrow3
{
    public static void main(String args[])
    {
        try
        {
            // throw an object of user defined exception
            throw new UserDefinedException("This is user-
defined exception");
        }
```

Java throw keyword

```
public class TestThrow1 {
    //
function to check if person is eligible to vote or not

    public static void validate(int age) {
        if(age<18) {
            //
throw Arithmetic exception if not eligible to vote
            throw new ArithmeticException("Person is
not eligible to vote");
        }
        else {
            System.out.println("Person is eligible to vote
!!");
        }
    }
    //main method
    public static void main(String args[]){
        //calling the function
        validate(13);
        System.out.println("rest of the code...");
```

Java throw keyword

```
 catch (UserDefinedException ude)
        {
            System.out.println("Caught the exception");
            // Print the message from MyException object
            System.out.println(ude.getMessage());
        }
    }
}
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow3.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrow3
Caught the exception
This is user-defined exception
```

Java throws keyword

- The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception.
- So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.
- Exception Handling is mainly used to handle the checked exceptions.
- If there occurs any unchecked exception such as NullPointerException, it is programmers' fault that he is not checking the code before it being used.

Syntax of Java throws

return_type method_name() **throws** exception_class_name

Java throws keyword

Advantage of Java throws keyword

- Now Checked Exception can be propagated (forwarded in call stack).
- It provides information to the caller of the method about the exception.

There are two cases:
Case 1: We have caught the exception i.e. we have handled the exception using try/catch block.
Case 2: We have declared the exception i.e. specified throws keyword with the method.

Java throws keyword

```
import java.io.IOException;
class Testthrows1{
  void m()throws IOException{
    throw new IOException("device error");//checked exception
  }
  void n()throws IOException{
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Output:

```
exception handled
normal flow...
```

Difference between throw and throws in Java

Sr. no.	Basis of Differences	throw	throws
1.	Definition	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.
2.	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.	

Difference between throw and throws in Java

Sr. no.	Basis of Differences	throw	throws
3.	Syntax	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.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	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.

Java throw Example

```
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
   }
}
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrow
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 throws Example

```
public class TestThrows {
    //defining a method
    public static int divideNum(int m, int n) throws ArithmeticExcepti
on {
        int div = m / n;
        return div;
    }
    //main method
    public static void main(String[] args) {
        TestThrows obj = new TestThrows();
        try {
            System.out.println(obj.divideNum(45, 0));
        }
        catch (ArithmeticException e){
            System.out.println("\nNumber cannot be divided by 0");
        }

        System.out.println("Rest of the code.
    }
}
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrows.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrows

Number cannot be divided by 0
Rest of the code..
```

Java throw and throws Example

```
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");
      }
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrowAndThrows.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrowAndThrows
Inside the method()
caught in main() method
```

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 Custom Exception

- 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.
- Consider the example 1 in which InvalidAgeException class extends the Exception class.

- Using the custom exception, we can have your own exception and message.
- Here, we have 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.

Consider the following example, where we create a custom exception named WrongFileNameException:

```
public class WrongFileNameException extends Exception {
    public WrongFileNameException(String errorMessage) {
    super(errorMessage);
    }
}
```

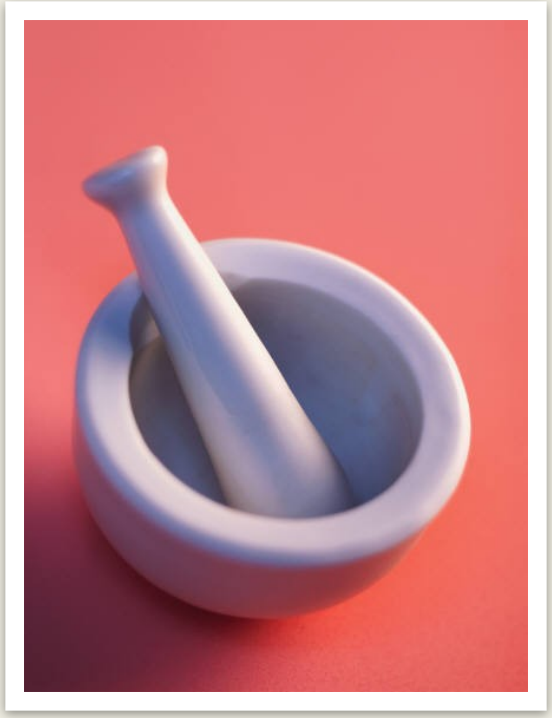
```
/ class representing custom exception
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
        super(str);
    }
}

// class that uses custom exception InvalidAgeException
public class TestCustomException1
{

    // method to check the age
```

```
/
static void validate (int age) throws InvalidAgeException
{
      if(age < 18){

       // throw an object of user defined exception
       throw new InvalidAgeException("age is not valid to v
ote");
    }
      else {
       System.out.println("welcome to vote");
       }
     }

    // main method
    public static void main(String args[])
    {
       try
       {
          // calling the method
          validate(13);
```

```
catch (InvalidAgeException ex)
        {
            System.out.println("Caught the exception");

            // printing the message from InvalidAgeException o
bject
            System.out.println("Exception occured: " + ex);
        }

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

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestCustomException1.java

C:\Users\Anurati\Desktop\abcDemo>java TestCustomException1
Caught the exception
Exception occured: InvalidAgeException: age is not valid to vote
rest of the code...
```



Thank You