Part - 2

EXCEPTION HANDLING

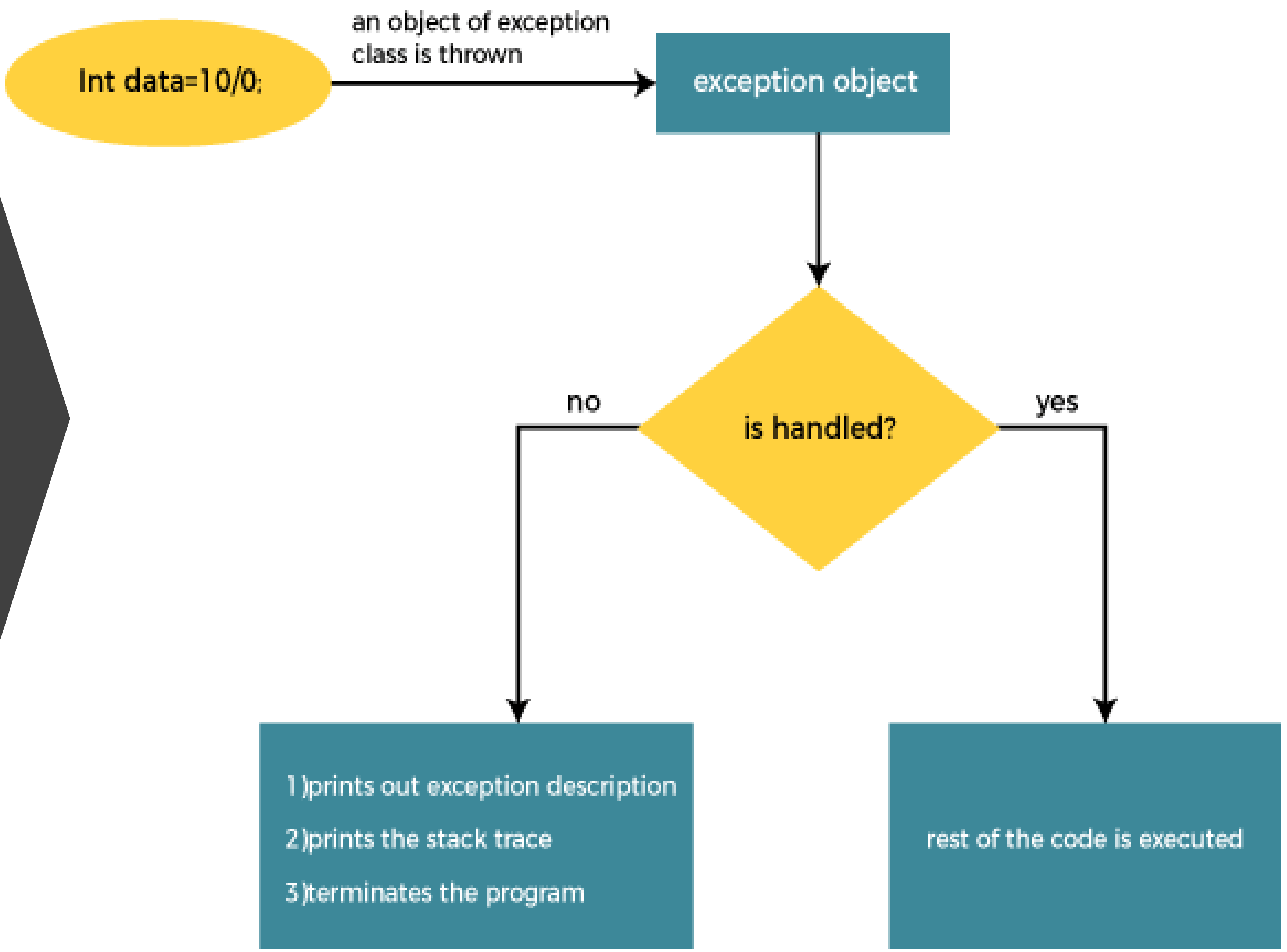
Syntax of Java try-catch

```
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
{
}
```

Internal Working of Java try-catch 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

Problem without exception handling

Let's try to understand the problem if we don't use a try-catch block.

```
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:

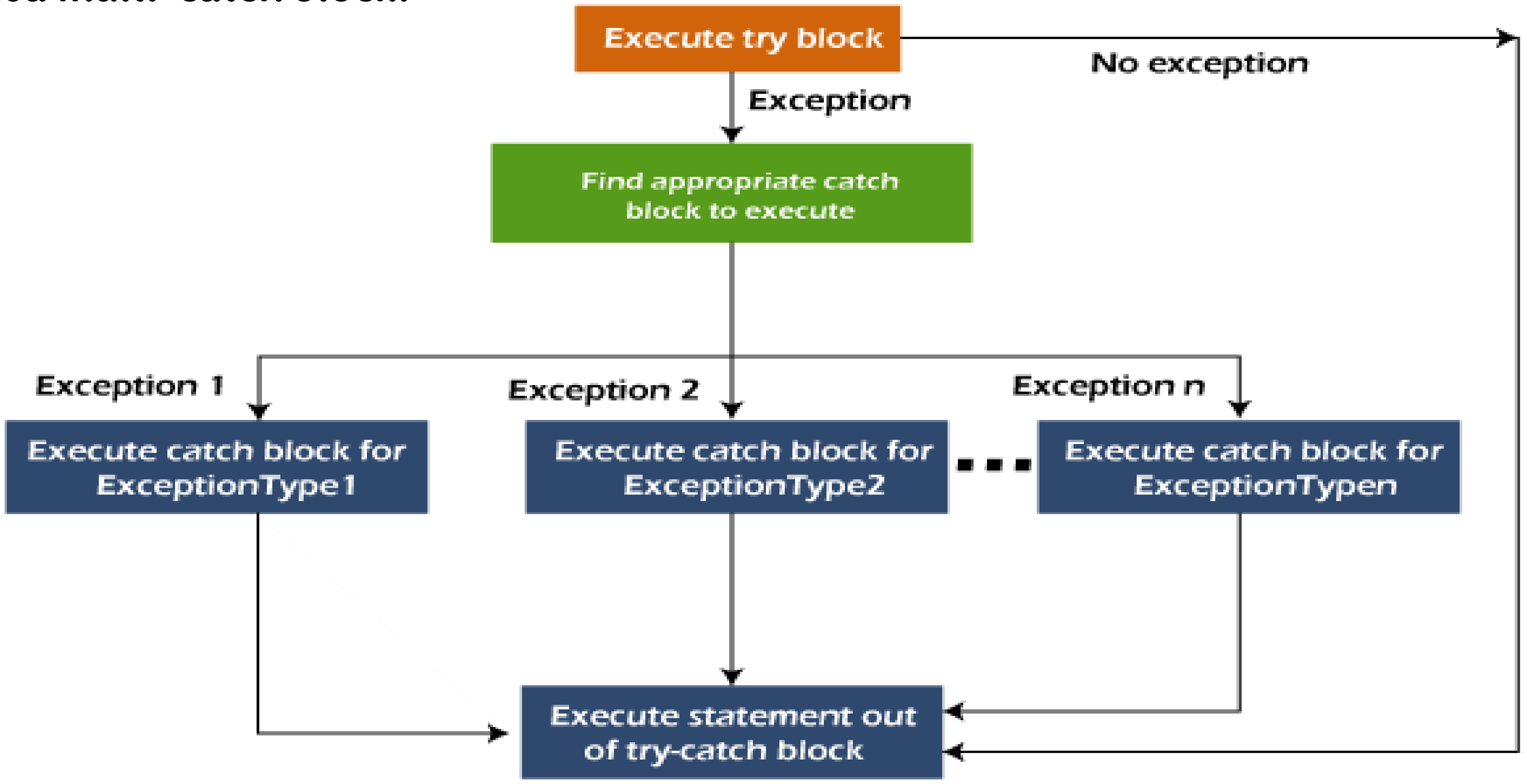
```
java.lang.ArithmeticException: / by zero
rest of the code
```

Multiple Catch Clause

Java Catch Multiple Exceptions

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.


Execute try block
No exception
Exception
Find appropriate catch block to execute
Exception 1
Exception 2
Exception n
Execute catch block for ExceptionType1
Execute catch block for ExceptionType2
Execute catch block for ExceptionTypen
Execute statement out of try-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");
   }  }
```

Output:

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

```
public class MultipleCatchBlock1 {
   public static void main(String[] args) {
      int a[] = new int[5];
      // First try block for ArrayIndexOutOfBoundsException
      try{
         a[5] = 30; // This will throw ArrayIndexOutOfBoundsException
      }
      catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("ArrayIndexOutOfBounds Exception occurs");
      }
      // Second try block for ArithmeticException
      try{
         a[0] = 30/0; // This will throw ArithmeticException
      }
      catch(ArithmeticException e) {
         System.out.println("Arithmetic Exception occurs");
      }
      // Parent catch blocks could be added here if needed
      System.out.println("rest of the code");
   }  }
```

```
public class MultipleCatchBlock2 {
   public static void main(String[] args) {
         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 Exception 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
```

Nested Try Block

❌ EXCEPTION

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).

Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

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
}
```

```
public class Example1 {
    public static void main(String[] args) {
        try {
            int dividend = 10;
            int divisor = 0;
            try {
                int result = dividend / divisor; // Attempting division
                System.out.println("Result: " + result);
            } catch (ArithmeticException e) {
                System.out.println("Inner catch block: " + e.getMessage());
            }
        } catch (Exception ex) {
            System.out.println("Outer catch block: " + ex.getMessage());
        }
    }
}
```

```
public class Example2 {
    public static void main(String[] args) {
        try {
            int[] arr = {1, 2, 3};
            try {
                int element = arr[3]; // Accessing index out of bounds
                System.out.println("Element: " + element);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Inner catch block: " + e.getMessage());
            }
        } catch (Exception ex) {
            System.out.println("Outer catch block: " + ex.getMessage());
        }
    }
}
```

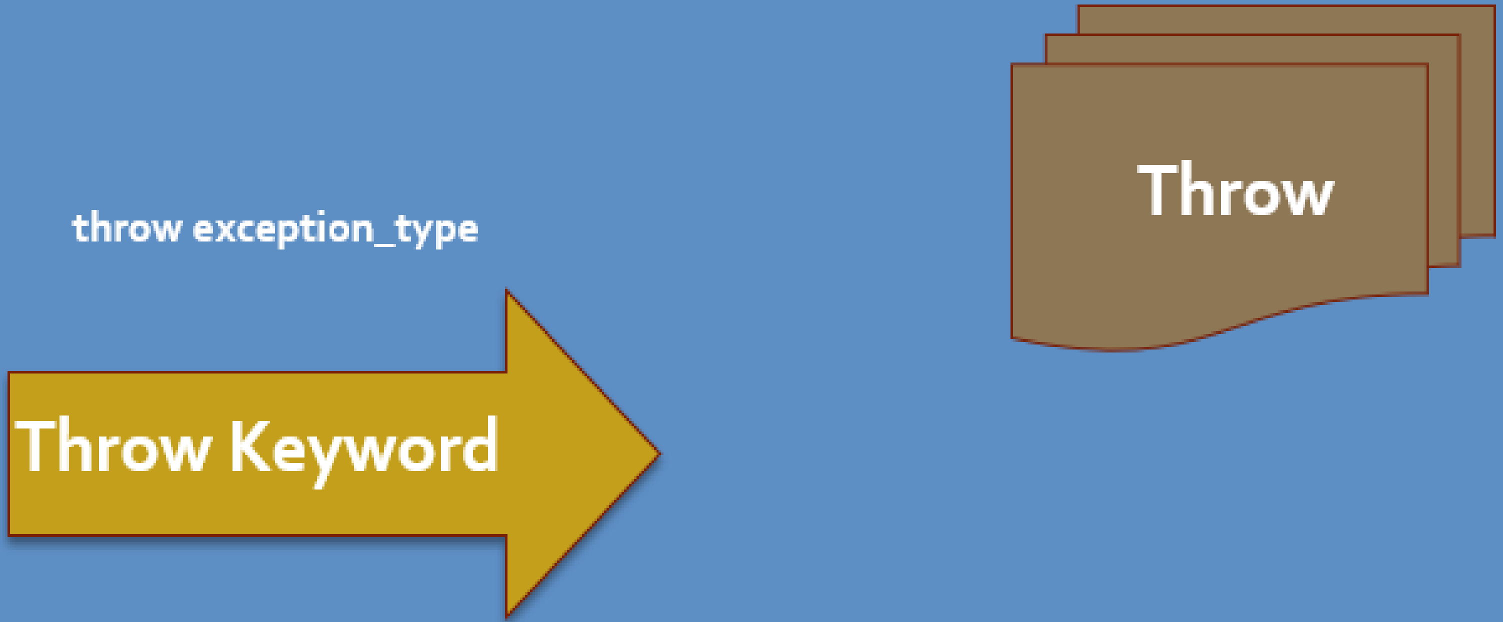
```
public class Example3 {
    public static void main(String[] args) {
        try {
            String str = null;
            try {
                int length = str.length(); // Accessing length of null string
                System.out.println("Length: " + length);
            } catch (NullPointerException e) {
                System.out.println("Inner catch block: " + e.getMessage());
            }
        } catch (Exception ex) {
            System.out.println("Outer catch block: " + ex.getMessage());
        }
    }
}
```

```
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);
  }
  //inner try block 2
  try{
  int a[]=new int[5];
  //assigning the value out of array bounds
   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..");
}    }
```

Throw Keyword in Java

throw instance;

throw exception_type



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 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. Here, we just need to set the condition and throw exception using throw keyword.

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
public class Example1 {
    public static void main(String[] args) {
        try {
            throw new Exception("Custom exception message");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
public class Example2 {
    public static void main(String[] args) {
        String str = null;
        try {
            if (str == null) {
                throw new NullPointerException("String is null");
            }
        } catch (NullPointerException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```java
public class Example6 {
    public static void main(String[] args) {
        try {
            int age = -5;
            if (age < 0) {
                throw new IllegalArgumentException("Age cannot be negative");
            }
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Throwing Unchecked Exception

In this example, we have created a method named validate() that accepts an integer as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
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...");
 }   }
```

Practice on nested try catch block

1. Design a program for calculating the monthly mortgage payment based on user input for loan amount, interest rate, and loan term. Handle arithmetic-related exceptions such as ArithmeticException using nested try-catch blocks.

Sample Input: Loan amount, interest rate, loan term (in years)

Sample Output: Monthly mortgage payment or an error message if the calculation encounters a divide-by-zero error due to invalid input or exceptional circumstances.

Practice on throw keyword

2. Suppose you are developing a social media platform where users need to be at least 18 years old to create an account. Upon registration, users are required to input their age. Write a Java program to verify the age of a user during the registration process. If the age provided is less than 18, throw a SecurityException with a message indicating that the user is underage.

Sample Input:

Enter age: 15

Sample Output:

SecurityException: User is underage (age: 15)