B. Sc. (H) Computer Science Semester II
BHCS03 – Programming in JAVA

Topic:

- **Exceptional Handling**

Exception Handling

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

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 use exception handling.

Hierarchy of Java Exception classes

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

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

Checked Exception
Unchecked Exception
Error

1) Checked Exception

   The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked

exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

There are 5 keywords which are used in handling exceptions.

1. try - The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

2. 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.

3. finally -The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.

4. throw - The "throw" keyword is used to throw an exception.

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

```
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...
```

Scenarios of Java Exceptions

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

The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";

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

4) A scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException
```

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 of try block, the rest of the block code will not execute. So, it is recommended not to keeping 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{}
```

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.

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.

Example 1

```
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");
   }
}
```

Test it Now

Output:

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

Example 2

```
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");
   }
}
```

Test it Now

Output:

ArrayIndexOutOfBounds Exception occurs
rest of the code

Example 3

In this example, try block contains two exceptions. But at a time only one exception occurs and its corresponding catch block is invoked.

```
public class MultipleCatchBlock3 {

   public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
            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:

Arithmetic Exception occurs
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.

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.

Example 1

Let's see a simple example of 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");
    }
}
```

Output:

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

Example 2

```
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
```

Example 3

In this example, try block contains two exceptions. But at a time only one exception occurs and its corresponding catch block is invoked.

```
public class MultipleCatchBlock3 {

   public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
            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:

Arithmetic Exception occurs
rest of the code

Java finally block

Java finally block is a block that is used to execute important code such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

Case 1

Let's see the java finally example where exception doesn't occur.

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Output:5
finally block is always executed
rest of the code...

Case 2

Let's see the java finally example where exception occurs and not handled.

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
```

```
  System.out.println(data);
 }
 catch(NullPointerException e){System.out.println(e);}
 finally{System.out.println("finally block is always executed");}
 System.out.println("rest of the code...");
 }
}
```

Output:finally block is always executed
Exception in thread main java.lang.ArithmeticException:/ by zero

Please solve/refer examples discussed on pages 191-194 of reference book [*].

*Schildt, H. Java: The Complete Reference. 9th edition. McGraw-Hill Education.

Difference between throw and throws in Java

There are many differences between throw and throws keywords.

A list of differences between throw and throws are given below:

1)	Java throw keyword is used to explicitly throw an exception. Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only. Checked exception can be propagated with throws.
3)	Throw is followed by an instance. Throws is followed by class.
4)	Throw is used within the method. Throws is used with the method signature.
5)	You cannot throw multiple exceptions. You can declare multiple exceptions e.g.

Java throw example

```
void m(){
throw new ArithmeticException("sorry");
}
```

Java throws example

```
void m()throws ArithmeticException{
//method code
}
```

Java throw and throws example

```
void m()throws ArithmeticException{
```

```
throw new ArithmeticException("sorry");
}
```

Please solve/refer examples discussed on pages 213-232 of reference book [*].

*Schildt, H. Java: The Complete Reference. 9th edition. McGraw-Hill Education.

References/Resources:

1. https://www.tutorialspoint.com
2. https://www.javatpoint.com
3. https://beginnersbook.com
4. https://docs.oracle.com
5. https://www.includehelp.com
6. Schildt, H. (2018). Java: The Complete Reference. 10th edition. McGraw-Hill Education.
7. Balaguruswamy, E. (2014). Programming with JAVA: A Primer. 5th edition. India: McGraw Hill Education
8. Horstmann, C. S. (2017). Core Java - Vol. I – Fundamentals (Vol. 10). Pearson Education

---

Assignment

Q1. What will be the output (write explanation also) of the below program?

a)

```
public class JavaHungry {
   public static void main(String args[])
   {
      try
      {
         System.out.print("A");
         int num = 99/0;
         System.out.print("B");
      }
      catch(ArithmeticException ex)
      {
         System.out.print("C");
      }
      catch(Exception ex)
      {
         System.out.print("D");
      }
      System.out.print("E");
   }
```

```
}
b)  public class JavaHungry
{   public static void main(String args[])  {

      try

      {

         System.out.print("A");

         int num = 99/0;

         System.out.print("B");

      }

      catch(ArithmeticException ex)

      {

         System.out.print("C");

      }

      catch(Exception ex)

      {

         System.out.print("D");

      }

      finally

      {

         System.out.print("E");

      }

      System.out.print("F");

   }

}
```

Q2. Create an exception subclass UnderAge, which prints “Under Age” along with the age value when an object of UnderAge class is printed in the catch statement. Write a class exceptionDemo in which the method test() throws UnderAge exception if the variable age passed to it as argument is less than 18. Write main() method also to show working of the program.. (Try on machine also, if possible)

Q3. Write a program to implement stack. Use exception handling to manage underflow and overflow conditions. (Try on machine also, if possible)

Q4. Can we write only try block without catch and finally blocks?

Q5. There are three statements in a try block – statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?