Programming Fundamentals

Exception Handling

Week 10| Iresh Bandara

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

Learning Outcomes

- Covers part of LO2 & LO3 for Module

- On completion of this lecture, students are expected to be able to:
  - Identify the need for arrays in a java program.
  - Construct java applications using Arrays and Array-Lists.
  - Develop java programs with 2D arrays.

What is an exception?

- An exception is a problem that arises during the execution of a program.
- It causes normal program flow to be disrupted.
- Examples :
  - Divide by zero errors
  - Accessing the elements of an array beyond its range
  - A user has entered invalid data
  - Hard disk crash
  - file that needs to be opened cannot be found.
  - Trying to read beyond the end of a file

3 categories of exceptions:

- **Checked exceptions:**

  An exception that is typically a user error or a problem that cannot be foreseen by the programmer.

  Ex: If a file is to be opened, but the file cannot be found, an exception occurs.

  These exceptions cannot be ignored at the time of compilation.

3 categories of exceptions:

- **Runtime exceptions:**

  An exception that occurs that probably could have been avoided by the programmer.

  As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.

3 categories of exceptions:

- **Errors:**

  These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error.

  Ex: If a stack overflow occurs, an error will arise.

  They are also ignored at the time of compilation

Runtime Exception Example

```
1 class DivByZero {
2  public static void main(String args[]) {
3    System.out.println(3/0);
4    System.out.println("Pls. print me.");
5  }
6 }
```

Example: Default Exception Handling

- Displays this error message;

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at DivByZero.main(DivByZero.java:3)
```

- Default exception handler
  - Provided by Java runtime
  - Prints out exception description
  - Prints the stack trace
    - Hierarchy of methods where the exception occurred
  - Causes the program to terminate

What Happens When an Exception Occurs?

- When an exception occurs within a method, the method creates an exception object and hands it off to the runtime system.
  - Creating an exception object and handing it to the runtime system is called "throwing an exception".
  - Exception object contains information about the error, including its type and the state of the program when the error occurred.

Decoding Exception Messages

```
class ArrayExceptionExample {
    public static void main(String args[]) {
        String[] names ={"Bilha", "Robert"};
        System.out.println(names[2]);
    }
}
```

- The println in the above code causes an exception to be thrown with the following exception message:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 2 at
ArrayExceptionExample.main(ArrayExceptionExample.java:4
)
```

Exception Message Format

- Exception messages have the following format:

```
[exception class]: [additional description of exception] at [class].[method]([file]:[line number])
```

Exception Messages Mini Pop-Quiz

- Exception message from array example

```
java.lang.ArrayIndexOutOfBoundsException: 2 at
ArrayExceptionExample.main(ArrayExceptionExample.jav  a:4)
```

- What is the exception class?

  `ArrayIndexOutOfBoundsException`
- Which array index is out of bounds?

  `2`
- What method throws the exception?

  `ArrayExceptionExample.main`
- What file contains the method?

  `ArrayExceptionExample.java`
- What line of the file throws the exception?

  `4`

Benefits of Java Exception Handling Framework

- Compilation cannot find all errors.
- To separate error handling code from regular code.
  - Code clarity (debugging, teamwork, etc.)
  - Worry about handling error elsewhere
- To separate error detection, reporting, and handling.
- To group and differentiate error types.
  - Write error handlers that handle very specific exceptions

Exception Terminology

- When an exception occurs, we say it was thrown or raised
- When an exception is dealt with, we say it is handled or caught
- The block of code that deals with exceptions is known as an exception handler

Throwing Exceptions

- System generated exceptions are automatically thrown by the Java run-time system.
- To manually throw an exception, use the keyword throw.
- Any exception that is thrown out of a method must be specified as such by a throws clause.

Catching Exceptions

- You can use a try-catch block to handle exceptions that are th

```
try {
   // code that might throw exception
}
catch ([Type of Exception] <object name>)
{
   // what to do if exception is thrown
}
```

Catching Exceptions

- Try-Catch Mechanism
  - Wherever your code may trigger an exception, the normal code logic is placed inside a block of code starting with the “try” keyword:
  - After the try block, the code to handle the exception should it arise is placed in a block of code starting with the “catch” keyword.

Example

```
1 class DivByZero {
2     public static void main(String args[]){
3     try {
4           System.out.println(3/0);
5           System.out.println("Pls print me.");
6     }
7     catch(ArithmeticException e) {
8           System.out.println(e.getMessage());
9     }
10    }
11 }
```

Task One

- Rewrite the program to handle any runtime exceptions.

```
public class TaskOne {
    public static void main(String[] args){
        int a=10;
        int b=5,c=5;
        int x,y;
        x = a / (b-c);
        y = a / (b+c);
        System.out.println("y = " + y);
    }
}
```

Catching Multiple Exceptions

- You can handle multiple possible exceptions by multiple successive catch blocks (acts like switch)

```
try {
    // code that might throw multiple
     // exceptions
}
catch (Exception-type1 e) {
    // handle Exception-type1
}
catch (Exception-type2 e2) {
    // handle Exception-type2
}
```

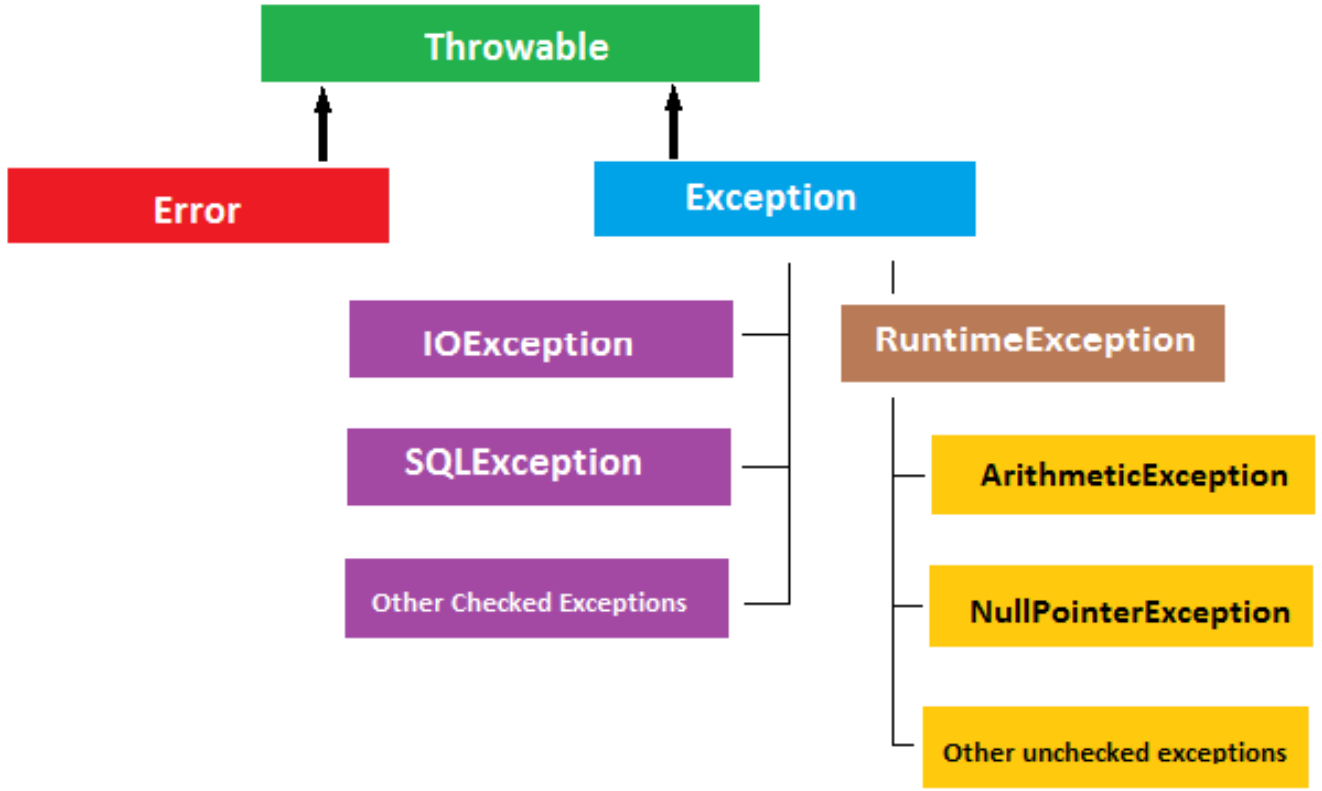
Task Two

- Rewrite the program to handle any runtime exceptions.

```
public class TaskTwo{
    public static void main(String[] args){
        int a [] = {5,10};
        int b=5;
        int x = a[2] / b - a[1];
        int y = a[1]/a[0];
        System.out.println("y = " + y);
    }
}
```

Exception Classes and Hierarchy

- Multiple catches should be ordered from subclass to superclass.



Example

```
1 class Example1 {
2   public static void main(String args[]) {
3      try {
4        int den = Integer.parseInt(args[0]);
5        System.out.println(3/den);
6      catch(Exception e2){
7        System.out.println("Any other error");
8      }catch (ArithmeticException e1) {
9        System.out.println("Divisor was 0.");
10     }      }
11   System.out.println("After exception.");
12 }
13}
```

Finally Block

- You can also use the optional finally block at the end of the try-catch block.
- It provides a mechanism to clean up regardless of what happens within the try block
  - Can be used to close files or to release other system resources
- finally code will ALWAYS run,
  - No matter what!
  - Even if a return or break occurs first
  - Except for System.exit( )

Try-Catch-Finally Block

```
try {
   // code that might throw exception
}
catch ([Type of Exception] e) {
   // what to do if exception is thrown
}
finally {
   // statements here always get
   // executed, regardless of what
   // happens in the try block
}
```

Throwing Exceptions: The throw Keyword

- Java allows you to throw exceptions (generate exceptions)

```
throw <exception object>;
```

- An exception you throw is an object
  - You have to create an exception object in the same way you create any other object
  - Example:

```
throw new ArithmeticException("testing...");
```

Example

```
class ThrowDemo {
  public static void main(String args[]){
        String input = "invalid input";
        try {
           if (input.equals("invalid input")) {
                throw new RuntimeException("throw demo");
        } else {
           System.out.println(input);
        }
        System.out.println("After throwing");
        } catch (RuntimeException e) {
                System.out.println("Exception caught:" + e);
        }
  }}
```

Task Three

- Rewrite the program to handle any runtime exceptions using **throw** keyword.

```
public class TaskThree {
    public static void main(String[] args){
        int a=10;
        int b=5,c=5;
        int x,y;
        x = a / (b-c);
        y = a / (b+c);
        System.out.println("y = " + y);
    }
}
```

Throwing Exceptions: The throws Keyword

- If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception.
- Syntax:

```
type methodName(parameterList)
        throws exceptionList
{
    <methodBody>
}
```

Example

```
public class ThrowsDemo {
    static void chkDiv ()throws ArithmeticException{
        int a=10,b=0;
        if(b==0){
           throw new ArithmeticException("Illegal");
        }else{
            System.out.println(a/b);
        }
    }
    public static void main(String arg[]){
        try{
                    chkDiv();
        }catch(ArithmeticException e){
            System.out.println("Caught"+e);
        }
    } }
```

Task Four - Rewrite the program to handle any runtime exceptions using throws keyword.

```
class TaskFour{
      static void divide(String s1[]) {
      int x=0,y=0;
      x=Integer.parseInt(s1[0]);
      y=Integer.parseInt(s1[1]);
      int result = x/y;
            System.out.println(result);
      }
      public static void main(String arg[]){
            String b[]={"10","0"};
            divide(b);
      }}
```

Summary

- An array is a group of variables of the same data type and referred to by a common name.
- Specific array elements are referred to by using array's name and the element's index.
- In Java, the length of an array is fixed at the time of its creation.
- When an array is passed to a method, only its reference is passed.
- 2-D arrays are used to represent tables, matrices, game boards, etc.
- ArrayLists are commonly used instead of arrays, because they expand automatically when new data is added to them.

Thank you

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY