CSEN401 – Computer Programming Lab

Topics:
Exception Handling

Prof. Dr. Slim Abdennadher

Exception

Attack of the Exception

```
public static int average(int[] a) {
    int total = 0;
    for (int i = 0; i < a.length; i++)
          total += a[i];
    return total / a.length;
}
```

- What happens when this method is used to take the average of an empty array?
- Program **throws** an **Exception** and **fails**.

  `java.lang.ArithmeticException: / by zero`

Different Types of Errors

- It is possible to reduce the number of **logical/run-time errors** to an acceptably small number.
- But a number of other problems (many of which may be external to your program) may still arise.
  - The program tries to write some data to a disk file, but an abnormal **I/O** termination occurs as the disk has no free space.
  - The program tries to connect to a particular Internet address, but a **connection** cannot be established.
  - Your program instantiates some more objects, however, when the Java VM tries to allocate **memory** for the new objects, it finds the computer has run out of free memory.
- A **robust** and **well-written** program will try to deal with the problem, e.g. by informing the user of the error, or exiting gracefully.

What is an Exception?

- An **Exception** is an error event that disrupts the program flow and may cause a program to fail.
- In Java, an exception is nothing more than an **object** that contains information on the type of error that occurs.
- Some **examples**:
  - Performing illegal arithmetic
  - Illegal arguments to method
  - Accessing an out-ouf-bounds array element
  - Hardware failures
  - Writing to a read-only file

Another Exception Example

- What is the output of this program?

```
public class ExceptionExample {
    public static void main(String[] args) {
        int[] array = {1,2};
        System.out.println(array[2]);
    }
}
```

- **Output**:

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

Exception Message Details

- **Exception message format**:

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

- **Example**:

```
java.lang.ArrayIndexOutOfBoundsException: 2
        at ExceptionExample.main(ExceptionExample.java: 4)
```

  - What exception class? `ArrayIndexOutOfBoundsException`
  - Which array index is out of bounds? `2`
  - What method throws the exception? `main`
  - What file contains the method? `ExampleException.java`
  - What line of the file throws the exception? `4`

Exception Handling

- Use a `try-catch` block to **handle** exceptions that are **thrown**:

```
try {
    // code that might throw exception
}
catch ([Type of Exception]) {
    // What to do if exception is thrown
}
```

Exception Handling – Example

```
public static int average(int[] a) {
    int total = 0;
    for (int i = 0; i < a.length, i++)
          total += a[i]
    return total / a.length
}


public static void printaverage(int[] a) {
    try {
          int avg = average(a);
          System.out.println("The average is " + avg);
    }
    catch (ArithmeticException e) {
          System.out.println("error calculating the average")
    }
}
```

Catching Multiple Exceptions

- Handle **multiple possible exceptions** by **multiple successive catch blocks**:

```
try {
    // code that might throw multiple exceptions
}
catch (IOException e) {
    // handle IOException and all subclasses
}
catch (ClassNotFoundException e2) {
    // handle ClassNotFoundException
}
```

Exception Handling in Java: try - catch - finally

```
try {
  ...
} catch (Type1 id1) {
  ... // Handle Exceptions of Type1
} catch (Type2 id2) {
  ... // Handle Exceptions of Type2
} finally {
  ... // always executed for user-defined clean-up operations
}
```

- The **`try` statement** identifies a block of statements within which an exception might be thrown.
- The **`catch` statement** associated with a `try` statement and identifies a block of statements that can handle a particular type of exception. The statements are executed if an exception of a particular type occurs within the `try` block.
- The **`finally` statement** associated with a `try` statement and identifies a block of statements that are executed regardless of whether or not an error occurs within the try block.

The `finally` Block

- The `try` blocks runs to the end and no excpetion is thrown. The `finally` block is executed after the `try` block.
- An exception is thrown in the `try` block and is caught in a corresponding `catch` block. The `finally` block is executed after the the `catch` block is executed.
- An exception is thrown in the `try` block and there is no matching `catch block`. The method invocation ends. The `finally` block is executed before the the method ends.

Where to Use `try` and `catch` Blocks?

- When catching multiple exceptions, the order of the `catch` blocks can be important. **Catch the more specific exception first**.
- You can have a **sequence** of `try` and `catch` blocks.

```
try { ... }
catch { ... }
try { ... }
catch { ... }
...
```

- You can place a `try` block and its subsequent `catch` blocks **inside** a larger `try` block or inside a larger `catch` block.

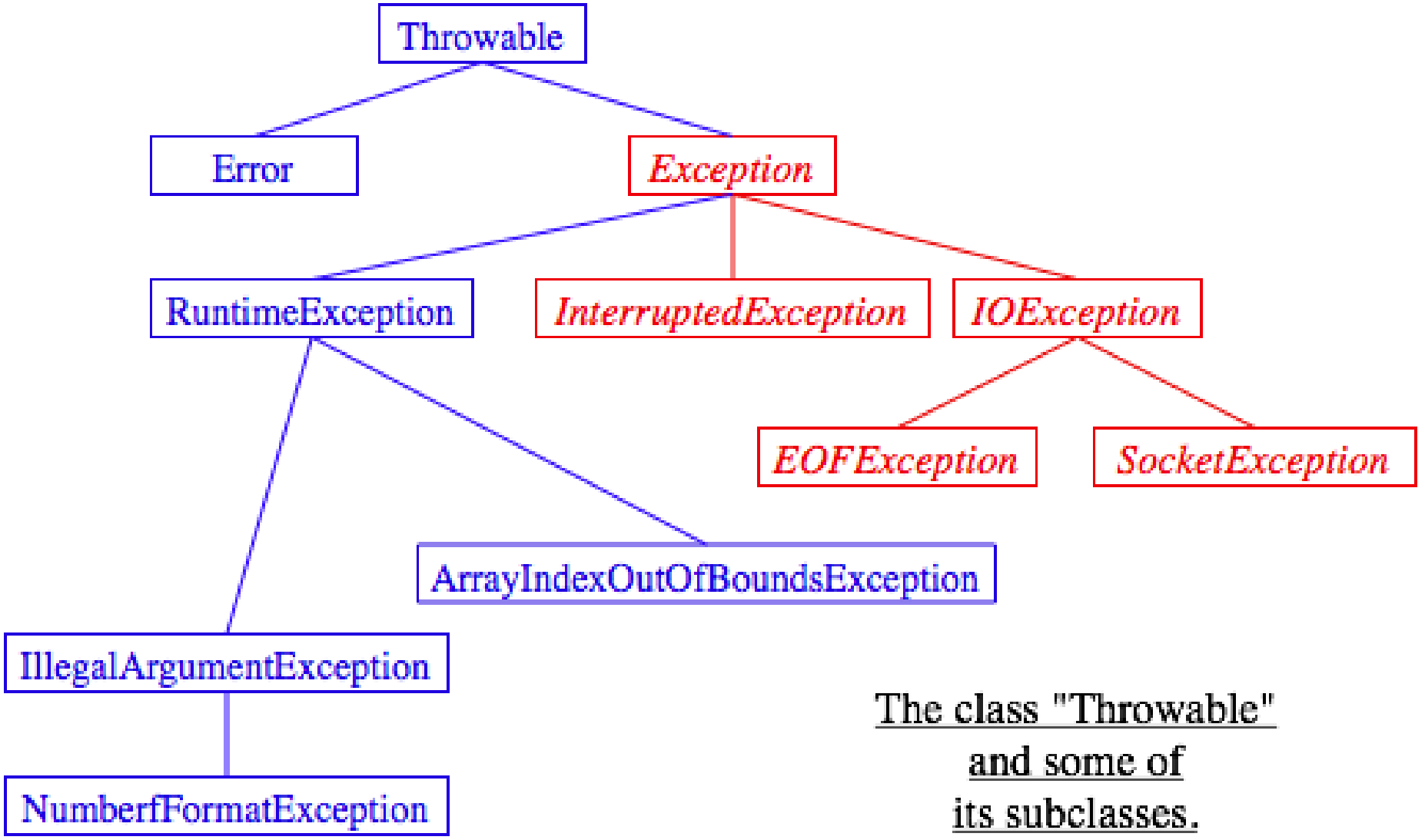
Exception Terminology

- When an exception happens we say it was **thrown** or **raised**.
- When an exception is dealt with, we say the exception is **handled** or **caught**.
- **Two types of exceptions**:
  - **Unchecked exceptions**: Usually occur because of programming errors, when code is not robust enough to prevent them.

    „Unchecked runtime exceptions represent conditions that, generally speaking, reflect errors in your program's logic and cannot be reasonably recovered from at run time.“[quote from The Java Programming Language, by Gosling, Arnold, and Holmes]
  - **Checked exceptions**: Usually occurs because of errors programmer cannot control, e.g. ivalid user input, unreadable files, hardware failures, ...

Exception Class Hierarchy

All exceptions are instances of classes that are **subclasses** of `Exception`.



Exceptions within Java – Examples

- **Unchecked Exceptions:**

```
RunTimeException
  ArithmeticException
  IndexOutOfBoundsException
    ArrayIndexOutOfBoundsException
    StringIndexOutOfBoundsException
    ...
NegativeArraySizeException
...
```

- **Checked Exceptions:**

```
ClassNotFoundException
IOException
  FileNotFoundException
  SocketException
  ...
AWTException
```

How to Deal with Checked Exceptions?

- Every method must **catch** (handle) checked exceptions or specify that it may **throw** them.
- Specify using the `throws` keyword.
- **Example**:
  -
```
void readFile(String filename) {
    try {
        FileReader reader = new Filereader("myfile.txt");
        }
    catch (FileNotFoundException e) {
         System.out.println("file was not found");
    }
}
```
  -
```
void readFile(String filename) throws FileNotFoundException {

        FileReader reader = new Filereader("myfile.txt");
        .....
}
```

The `throws` Clause

- If a method throws exceptions but does not catch them, the Java language requires that a method declares in a `throws` clause the exceptions that it may throw.
- If a method throws an exception, and the exception is not caught inside the method, then the method invocation ends immediately after the exception is thrown.
- Only exceptions that will cause a method to **complete abruptly** should appear in its `throws` clause.
- **Reason**: Delay of handling an exception.

Writing your Own Exception Handler – Exception Constructors

- **Exceptions** have at least **two constructors**:
  - **No arguments**:

    ```
    NullPointerException e = new NullPointerException();
    ```

  - **Single `String` argument**: descriptive message that appears when exception error message is printed:

    ```
    IllegalArgumentException e = new
          IllegalArgumentException("number must be positive");
    ```

Writing your Own Exception Handler

- To write your own exception handler, write a **subclass** of `Exception` or any defined exception class and write both types of **constructors**:
  - **Checked Exceptions**:

```
public class MyCheckedException extends IOException {
      public MyCheckedException() {};
      public MyCheckedException(String m)
         {super(m);}
}
```

  - **Unchecked Exceptions**:

```
public class MyUnCheckedException extends RunTimeException {
      public MyUnCheckedException() {};
      public MyUnCheckedException(String m)
         {super(m);}
}
```

Average Example – Throwing Exceptions

- Throw exception using the `throw` keyword:

```
public static int average(int[] a) {
    if (a.length == 0)
       throw new IllegalArgumentException("array is empty");
    int total = 0;
    for (int i = 0; i < a.length; i++)
          total += a[i];
    return total / a.length;
}
```

Defining an Exception Handler – Example (I)

```
public class EmptyException extends IllegalArgumentException  {

    public EmptyException() {}

    public EmptyException(String m)
    {super(m);}

}
```

- You can do more in an exception constructor, but this form is common.
- `super` is an invocation for the base class `IllegalArgumentException`.

Defining an Exception Handler – Example (II)

```
public class Average {
public static int average(int[] a) throws EmptyException {
    if (a.length == 0)
       throw new EmptyException("array is empty");
    int total = 0;
    for (int i = 0; i < a.length; i++)
 total += a[i];
    return total / a.length;
}

public static void main(String[] args) {
        int[] a = {};
        try { System.out.println(average(a)); }
        catch(EmptyException e)
        { System.out.println(e.getMessage()); }
    }
}
```

`getMessage` is a method inherited form the super class.

When to Define an Exception Class?

- If you are going to insert a `throw` statement in your code.
- When your code catches an exception, your catch blocks can tell the difference between your exceptions and exceptions thrown by methods in predefined classes.

Exception Handling Best Practices

Read more from

http://www.javacodegeeks.com/2013/07/java-exception-handling-tutorial-with-examples-and-best-practices.html

- Use Specific Exceptions
- Throw Early or Fail-Fast
- Catch Late
- Closing Resources
- Logging Exceptions
- Single catch block for multiple exceptions
- Using Custom Exceptions
- Naming Conventions and Packaging
- Use Exceptions Judiciously