Class 03 – Exceptions

CSIS 3475 Data Structures and Algorithms

The Basics

- Method creates an exception object
  - We say “throws an exception”
- Signal to program
  - Unexpected has happened
- Handle the exception
  - Detect and react

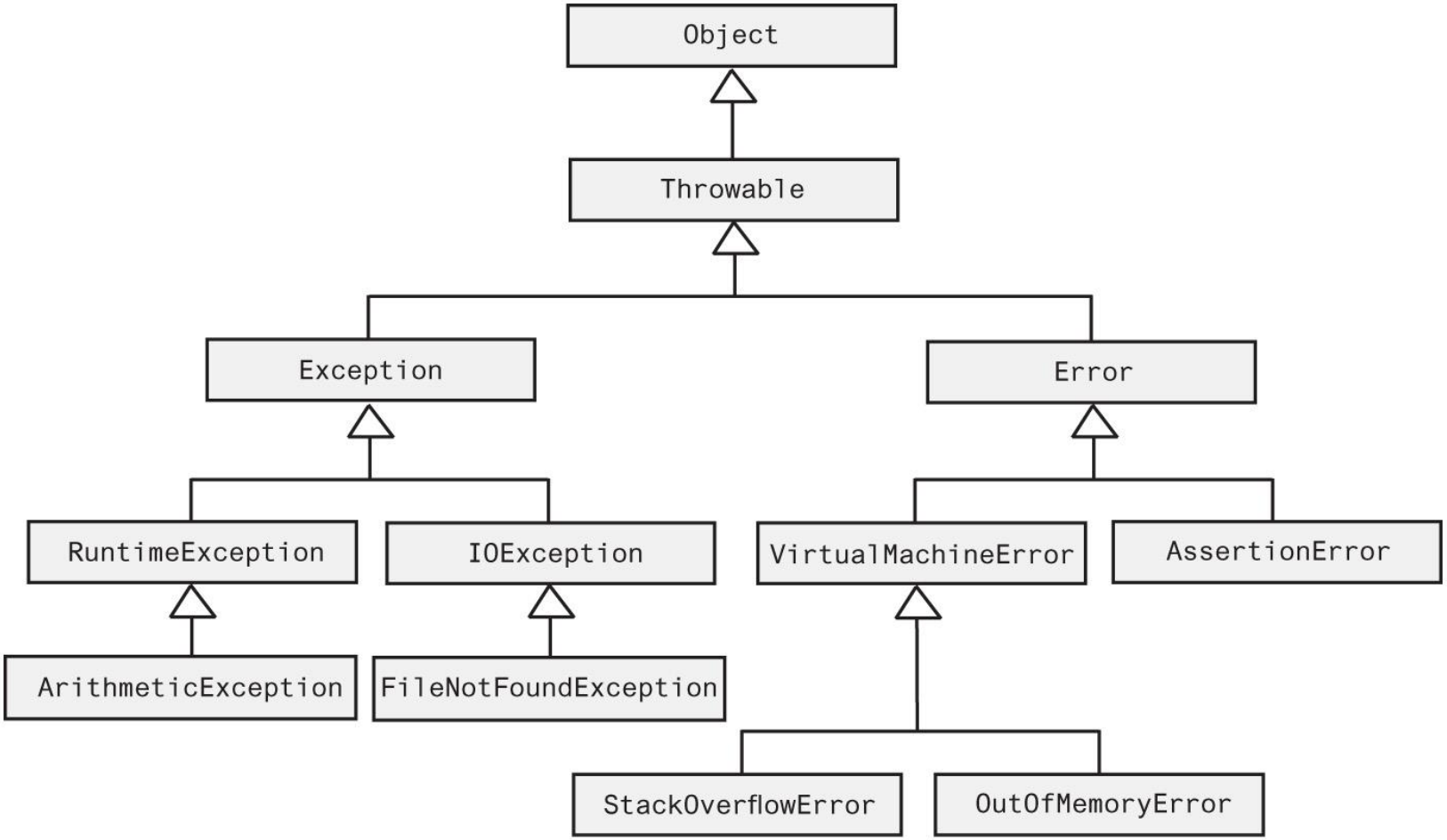
The Basics

- Checked exceptions in the Java Class Library
  - **`ClassNotFoundException`**
  - **`FileNotFoundException`**
  - **`IOException`**
  - **`NoSuchMethodException`**
  - **`WriteAbortedException`**

The Basics

- Runtime exceptions in the Java Class Library
  - **`ArithmeticException`**
  - **`ArrayIndexOutOfBoundsException`**
  - **`ClassCastException`**
  - **`IllegalArgumentException`**
  - **`IllegalStateException`**
  - **`IndexOutOfBoundsException`**
  - **`NoSuchElementException`**
  - **`NullPointerException`**
  - **`StringIndexOutOfBoundsException`**
  - **`UnsupportedOperationException`**

Java Class Exception and Error Hierarchy



Handling an Exception

- Postpone handling: The throws clause
  - If programmer not sure what action is best for a client when an exception occurs
  - Leave the handling of the exception to the method's client
- Method that can cause but does not handle checked exception must declare in its header

```
public String readString(. . .) throws IOException
```

Handle It Now: The try-catch Blocks

- **Code to handle an `IOException` as a result of invoking the method `readString`**

```
try
{
        < Possibly some code >
        anObject.readString(. . .); // Might throw an IOException
        < Possibly some more code >
}
catch (IOException e)
{
        < Code to react to the exception, probably including the following statement: >
        System.out.println(e.getMessage());
}
```

Multiple `catch` Blocks

- **Good order for `catch` blocks**

```
catch (FileNotFoundException e)
{
      . . .
}
catch (IOException e) // Handle all other IOExceptions
{
      . . .
}
```

Throwing an Exception

- A method intentionally throws an exception by executing a throw statement.
- Programmers usually create the object within the throw statement

```
throw new IOException();
```

Throwing an Exception

- If you can resolve unusual situation in a reasonable manner
  - likely can use a decision statement
- If several resolutions to abnormal occurrence possible, and you want client to choose
  - Throw a checked exception
- If a programmer makes a coding mistake by using your method incorrectly
  - Throw a runtime exception

Programmer-Defined Exception Classes

- Define your own exception classes by extending existing exception classes
  - Existing superclass could be one in the Java Class Library or one of your own
- Constructors in an exception subclass are the most important
  - Often the only methods you need to define

SquareRootException

```
/**
 * A class of runtime exceptions thrown when an attempt is made to find the
 * square root of a negative number.
 *
 * @author Frank M. Carrano
 * @author Timothy M. Henry
 * @version 5.0
 */
public class SquareRootException extends RuntimeException {
      public SquareRootException() {
            super("Attempted square root of a negative number.");
      }
      public SquareRootException(String message) {
            super(message);
      }
}
```

Using SquareRootException

```
public class OurMath {
	/**
	 * Computes the square root of a nonnegative real number.
	 *
	 * @param value A real value whose square root is desired.
	 * @return The square root of the given value.
	 * @throws SquareRootException if value < 0.
	 */
	public static double squareRoot(double value) throws SquareRootException {
		if (value < 0)
			throw new SquareRootException();
		else
			return Math.sqrt(value);
	}

}
```

SquareRootException demo

- Catch the exception, the program never finishes

```
/**
 * A demonstration of a runtime exception using the class OurMath.
 *
 * @author Frank M. Carrano
 * @author Timothy M. Henry
 * @version 5.0
 */
public class OurMathDemo {
        public static void main(String[] args) {
                System.out.print("The square root of 9 is ");
                System.out.println(OurMath.squareRoot(9.0));

                System.out.print("The square root of -9 is ");
                System.out.println(OurMath.squareRoot(-9.0));

                // this should never be reached
                System.out.print("The square root of 16 is ");
                System.out.println(OurMath.squareRoot(16.0));
        }
}
```

JoeMath

```
public class JoeMath {
	/**
	 * Computes the square root of a real number.
	 *
	 * @param value A real value whose square root is desired.
	 * @return A string containing the square root.
	 */
	public static String squareRoot(double value) {
		String result = "";
		// take the square root
		// if exception thrown, then make
		// the value positive, and append an i to result string
		// for an imaginary value
		try {
			Double temp = OurMath.squareRoot(value);
			result = temp.toString();
		} catch (SquareRootException e) {
			Double temp = OurMath.squareRoot(-value);
			result = temp.toString() + "i";
		}

		return result;
	}
}
```

JoeMathDemo

```
/**
 * A demonstration of a runtime exception using the class JoeMath.
 *
 * @author Frank M. Carrano
 * @author Timothy M. Henry
 * @version 5.0
 */
public class JoeMathDemo {
      public static void main(String[] args) {

            // negative numbers should have i appended to the result

            System.out.print("The square root of 9 is ");
            System.out.println(JoeMath.squareRoot(9.0));

            System.out.print("The square root of -9 is ");
            System.out.println(JoeMath.squareRoot(-9.0));

            System.out.print("The square root of 16 is ");
            System.out.println(JoeMath.squareRoot(16.0));

            System.out.print("The square root of -16 is ");
            System.out.println(JoeMath.squareRoot(-16.0));
      }
}
/*
The square root of 9 is 3.0
The square root of -9 is 3.0i
The square root of 16 is 4.0
The square root of -16 is 4.0i

*/
```

The finally Block

- This code shows the placement of the `finally` block

```
try
{
        < Code that might throw an exception, either by executing a throw statement or by
        calling a method that throws an exception >
}
catch (AnException e)
{
< Code that handles exceptions of type AnException or a subclass of AnException >
}
< Possibly other catch blocks to handle other types of exceptions >
finally
{
< Code that executes after either the try block or an executing catch block ends >
}
```

The `finally` Block

- Whether an exception occurs or not, `closeRefrigerator` is called within the `finally` block.

```
public class GetMilk {
	public static void main(String[] args) {
		try {
			openRefrigerator();
			takeOutMilk(); // will sometimes throw an exception
			pourMilk();
			putBackMilk();
		} catch (NoMilkException e) {
			System.out.println(e.getMessage());
		} finally {
			closeRefrigerator();
		}
	}

	public static void openRefrigerator() {
		System.out.println("Open the refrigerator door.");
	}

	/**
	 * Coin flip says we are out of milk
	 * @throws NoMilkException
	 */
	public static void takeOutMilk() throws NoMilkException {
		if (Math.random() < 0.5)
			System.out.println("Take out the milk.");
		else
			throw new NoMilkException("Out of Milk!");
	}

	public static void pourMilk() {
		System.out.println("Pour the milk.");
	}

	public static void putBackMilk() {
		System.out.println("Put the milk back.");
	}

	public static void closeRefrigerator() {
		System.out.println("Close the refrigerator door.");
	}
}
```

The `finally` Block

- A demonstration of a finally block output

Sample Output 1 (no exception is thrown)

```
Open the refrigerator door.
Take out the milk.
Pour the milk.
Put the milk back.
```

Sample Output 2 (exception is thrown)

```
Open the refrigerator door. Out of milk!
Close the refrigerator door.
```