

## Chapter 17

# More on Exception Handling

Lecture slides for:

*Java Actually: A Comprehensive Primer in Programming*

Khalid Azim Mughal, Torill Hamre, Rolf W. Rasmussen

Cengage Learning, 2008.

ISBN: 978-1-844480-933-2

<http://www.iit.uib.no/~khalid/jac/>

*Permission is hereby granted to use these lecture slides in conjunction with the book.*

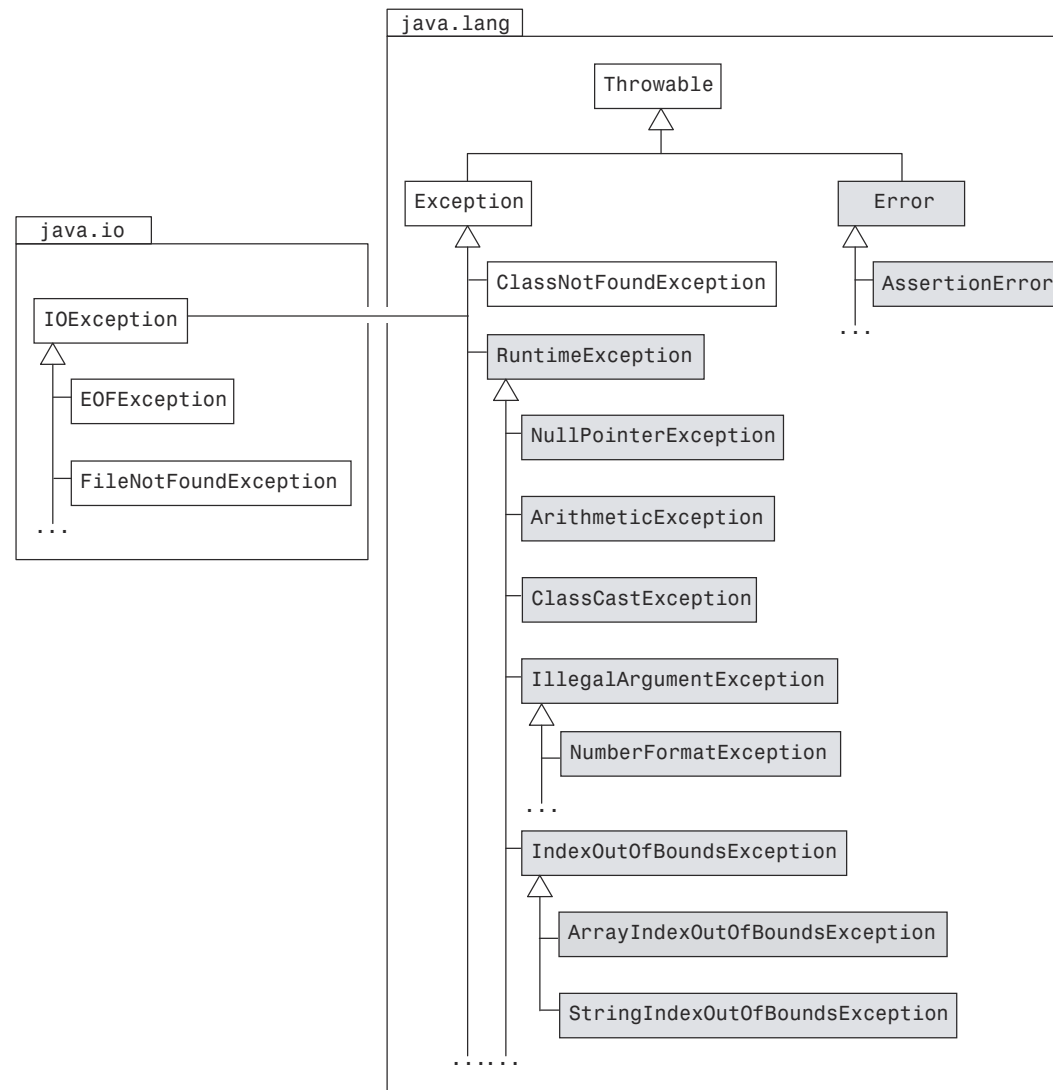
*Modified: 16/2/19*

# Overview

---

- Overview of exception classes
  - Explicitly throwing an exception: the `throw` clause
  - `try` block with multiple catch blocks
  - Typical programming errors in exception handling
  - Executing the `finally` block
-

## Partial hierarchy of exception classes (Figure 18.1)



*The grey-coloured classes (and their subclasses) represent unchecked exceptions.*

## Selected methods from the Throwable class (Table 18.1)

Method	Description
<code>String getMessage()</code>	Returns the string in the exception. The string provides more explanation of the exception.
<code>void printStackTrace()</code>	Prints on the terminal the stack trace at the time the exception was thrown.

## Class Exception and checked exceptions

- Exceptions of the type Exception class and its subclasses, with the exception of subclasses of RuntimeException class, are called *checked exceptions*.
  - The compiler checks that a method that throws a checked exception, also explicitly handles the exception.

## Class RuntimeException and unchecked exceptions

- The class RuntimeException and its subclasses represent exceptions that represent typical unforeseen errors, such as *programming errors*.
- Exceptions that are defined by the class RuntimeException and its subclasses is called *unchecked exceptions*.
  - It is not necessary that they be caught in the program, but the cause of the error must be corrected in the program.

## Class Error

- The subclass AssertionError is used to signal that an assertion does not hold during the execution.
  - Such exceptions should not be caught.

## Explicitly throwing an exception

- A program can explicitly throw an exception with the `throw` statement:  
`throw new ArithmeticException ("Distance and time cannot be < 0");`
- Executing a `throw` statement interrupts the normal execution of the program, and the exception is propagated.
- It is common to use an appropriate exception class to define an error situation and provide supplementary information about the exception in the constructor call.
- In Program 18.1, the program execution will result in an `ArithmeticException` being thrown in the method `calculateSpeed()`, (4).
  - This exception will propagate and will be handled by the catch block in `main()` method.
  - From this point, normal execution of the program will continue.

## Throwing an exception programmatically (Program 18.1)

```
public class Speed4 {  
  
    public static void main(String[] args) {  
        System.out.println("Entering main().");  
        try {  
            printSpeed(-100, 10);                // (1) Distance < 0.  
        }  
        catch (ArithmeticException exception) {    // (2)  
            System.out.println(exception + " (handled in main())");  
        }  
        System.out.println("Returning from main().");  
    }  
  
    private static void printSpeed(int kilometers, int hours) {  
        System.out.println("Entering printSpeed().");  
        int speed = calculateSpeed(kilometers, hours);  
        System.out.println("Speed = " +  
                           kilometers + "/" + hours + " = " + speed);  
        System.out.println("Returning from printSpeed().");  
    }  
}
```

```
private static int calculateSpeed(int distance, int time) {  
    System.out.println("Calculating speed.");  
    if (distance < 0 || time < 0) // (3)  
        throw new ArithmeticException("distance and time" +  
                                        " cannot be < 0"); // (4)  
    return distance/time;  
}  
}
```

- Output from the program:

Entering main().

Entering printSpeed().

Calculating speed.

java.lang.ArithmeticException: distance and time cannot be < 0 (handled in main())

Returning from main().



## Handling several types of exceptions

- If the code in a try block can throw different types of exceptions, we can specify one catch block for each type of exception after the try block (Program 18.2).
- Indexing in the string array args, (3) and (4), can throw an unchecked `ArrayIndexOutOfBoundsException` unless at least two strings are specified on the command line.
- Converting to integer with the `parseInt()` method, (3) and (4), can throw an unchecked `NumberFormatException` if any of the strings contain characters that can not be part of an integer.

## try block with several catch blocks (Program 18.2)

```
public class Speed5 {  
  
    public static void main(String[] args) {  
        System.out.println("Entering main().");  
        int arg1, arg2;                                // (1)  
        try {                                           // (2)  
            arg1 = Integer.parseInt(args[0]);           // (3)  
            arg2 = Integer.parseInt(args[1]);           // (4)  
        }  
        catch (ArrayIndexOutOfBoundsException exception) { // (5)  
            System.out.println("Specify both kilometers and hours.");  
            System.out.println("Usage: java Speed5 <kilometers> <hours>");  
            System.out.println(exception + " (handled in main())");  
            return;  
        }  
        catch (NumberFormatException exception) { // (6)  
            System.out.println("Kilometers and hours must be integers.");  
            System.out.println("Usage: java Speed5 <kilometers> <hours>");  
            System.out.println(exception + " (handled in main())");  
            return;  
        }  
        printSpeed(arg1, arg2);                          // (7)  
        System.out.println("Returning from main().");  
    }  
}
```

```

private static void printSpeed(int kilometers, int hours) {
    System.out.println("Entering printSpeed().");
    try {
        int speed = calculateSpeed(kilometers, hours);
        System.out.println("Speed = " +
                           kilometers + "/" + hours + " = " + speed);
    }
    catch (ArithmeticException exception) {
        System.out.println(exception + " (handled in printSpeed())");
    }
    System.out.println("Returning from printSpeed().");
}

private static int calculateSpeed(int distance, int time) {
    System.out.println("Calculating speed.");
    if (distance < 0 || time < 0)
        throw new ArithmeticException("distance and time cannot be < 0");
    return distance/time;
}
}

```

- Output from the program:

```
> java Speed5 100
Entering main().
Specify both kilometers and hours.
Usage: java Speed5 <kilometers> <hours>
java.lang.ArrayIndexOutOfBoundsException: 1 (handled in main())
> java Speed5 200 4u
Entering main().
Kilometers and hours must be integers.
Usage: java Speed5 <kilometers> <hours>
java.lang.NumberFormatException: For input string: "4u" (handled in main())
> java Speed5 200 -10
Entering main().
Entering printSpeed().
Calculating speed.
java.lang.ArithmeticException: distance and time cannot be < 0 (handled in printSpeed())
Returning from printSpeed().
Returning from main().
> java Speed5 200 0
Entering main().
Entering printSpeed().
Calculating speed.
java.lang.ArithmeticException: / by zero (handled in printSpeed())
Returning from printSpeed().
Returning from main().
```

## Typical programming errors in exception handling

- A parameter type in a catch block can shadow other exception types in the subsequent catch blocks.
  - For example, the superclass `RuntimeException` in the catch block (1) shadows the subclass `ArithmeticException` in the catch block (2):

```
try { ... }  
catch (RuntimeException    exception1) { ... }    // (1)  
catch (ArithmeticException exception2) { ... }    // (2)
```

- Exceptions of type `ArithmeticException` are caught by the catch block (1) and never by the catch block (2), since objects of subclasses can be assigned to a superclass reference.
  - The compiler will warn of such cases.
- Do not catch all exceptions in one catch block by using more general exception classes, such as `Exception` and `RuntimeException`.
  - Using specific exception classes, often with multiple catch blocks, are recommended in the try-catch construct to improve program understanding.

## Defining new exceptions

- It is recommended to define new subclasses of the class `Exception`.
- The new exception is then automatically checked by the compiler.

```
class SpeedCalculationException extends Exception {  
    SpeedCalculationException(String str) {  
        super(str);  
    }  
}
```

- It is usually sufficient to define only a constructor that takes a string parameter.

## Executing the finally block

- A finally block can occur with a try block.
  - A catch block does not necessarily need to be specified.
- The code in the finally block is always executed if the try block is executed.
  - It does not matter if an exception was thrown or not.
- "Cleanup Code" may be put in a finally block so it will always be executed.
- The class Speed7 illustrates executing a finally block.
  - A finally block is specified at (4), with the corresponding try block at (1).
  - Printout shows that the code in the finally block will always be executed if the try block is executed.

```

public class Speed7 {

    public static void main(String[] args) {
        System.out.println("Entering main().");
        try {
            //      printSpeed(100, 20);
            printSpeed(-100,20);
        }
        catch (SpeedCalculationException exception) {
            System.out.println(exception + " (handled in main())");
        }
        finally {
            System.out.println("Command to use: java Speed7");
        }
        System.out.println("Returning from main().");
    }
}

```

//(1)  
 //(2a)  
 //(2b)  
 //(3)  
 //(4)



```

private static void printSpeed(int kilometers, int hours)
                                throws SpeedCalculationException {           //(5)
    System.out.println("Entering printSpeed().");
    double speed = calculateSpeed(kilometers, hours);
    System.out.println("Speed = " +
        kilometers + "/" + hours + " = " + speed);
    System.out.println("Returning from printSpeed().");
}

private static int calculateSpeed(int distance, int time)
                                throws SpeedCalculationException {           //(6)
    System.out.println("Calculating speed.");
    if (distance < 0 || time <= 0)
        throw new SpeedCalculationException("distance and time " +
                                                "must be > 0");               //(7)
    return distance/time;
}
}

```

- Running the program with the code line (2a):

```
    printSpeed(100, 20);                                // (2a)
```

gives the following output:

Entering main().

Entering printSpeed().

Calculating speed.

Speed = 100/20 = 5.0

Returning from printSpeed().

Command to use: java Speed7

Returning from main().

- Running the program with the code line (2b):

```
    printSpeed(100, 0);                                // (2b)
```

gives the following output:

Entering main().

Entering printSpeed().

Calculating speed.

SpeedCalculationException: distance and time must be > 0 (handled in main())

Command to use: java Speed7

Returning from main().

- Note that the execution of the method printSpeed() was stopped *after* the finally block was executed.

## Exception handling and inheritance

- A method in the subclass can only specify in its throws clause *a subset of the exception types* specified in the throws clause of the method it is overriding from a superclass.

```
// Class A
public void methodWithManyExceptions() throws Exception1, Exception2, Exception3 {...}
...
```

```
// In subclass B which extends superclass A
public void methodWithManyExceptions() throws Exception1, Exception3 {...}
```

## Summary of exception handling

- "Throw and catch" rule for checked exceptions:
  - A method can catch the exceptions it throws, and handle them in catch blocks.
  - The checked exceptions that a method can throw and not catch, must be declared in a throws clause of the method.
- A method can programmatically throw an exception using the throw statement.

```
... methodName(...) throws Exception1, Exception2, ... , Exceptionn {
```

```
...
```

```
try { // code resulting in one of the following exceptions being thrown:
```

```
    // Exception1, ... , Exceptionn or Exceptiona, ..., Exceptionm
```

```
    throw new Exceptioni(); // throws an exception of type Exceptioni
```

```
}
```

```
catch (Exceptionj ej) { // catches exception of type Exceptionj
```

```
    // code to handle Exception ej
```

```
}
```

```
...
```

```
catch (Exceptionk ek) { // catches exception of type Exceptionk
```

```
    // code to handle Exception ek
```

```
}
```

```
finally { ... } // Code that is always executed if try block is executed.
```

```
...
```

```
}
```

try-catch-finally **block**