Exception Handling

Reference:
Sections 5.5 - 5.8

Overview

Exception Handling:
- Exception Hierarchy
- Exceptions: checked and unchecked
- try-catch-finally block
- throw-statement
- throws-clause
- Scenarios through the try-catch-finally block
- Exception and Inheritance

Exceptions

- Exceptions are runtime errors which indicate exceptional conditions.
- These exceptional conditions are usually not part of the primary behavior of a program.
- Examples of such exceptions are:
  - index out of bounds, division by zero, illegal parameter
  - file not found, read or write error on a file, using null-literal to reference an object.
  - network link failure, security violation
- Exception handling is based on the "throw and catch" paradigm.
  - an exception is thrown when a situation warrants it, and is caught by an exception handler.

If an exception is not handled by any exception handler in the program, it is handled by the default exception handler (i.e. the runtime system).
- For terminal-based applications, the program is aborted after the exception has been handled by the default exception handler.
- For GUI-based applications, the program continues after the exception has been dealt with by the default exception handler — the user interaction is handled by a different thread (the AWT thread).
- In all cases, the default exception handler prints the trace of the method activation stack.

java.lang

 Throwable

 Error

 Exception

 java.io

 IOException

 ...

 InterruptedIOException ... RuntimeException

 ArithmeticException ... NullPointerException

...
## Exception Hierarchy

- Exceptions are objects of classes which extend the class `Throwable`.
- New exception types are created by extending (subclasses of) the `Exception` class.
- A **checked exception** is an exception which is not `RuntimeException` or `Error` (or a subclass of these two classes).

```
java.lang
    Throwable
        Error
            Exception
                java.io
                    IOException
                        ...
                    ...
                    ...
                ...
                InterruptedException
            ...
                RuntimeException
                        IOExpection
                ...
                ArithmeticException
                        ...
                NullPointerExpection
```

## Example: Default Exception Handling

```java
1. // DivisionByZero1.java
2. 3. public class DivisionByZero1 {
4.      public static void division() {
5.          int num1 = 10;
6.          int num2 = 0;
7.          System.out.println(num1 + " / " + num2 + " = " + (num1/num2));
8.      }
9. }
10. 11.     public static void main(String args[]) {
12.         division();
13.     }
14. }
```

Output on the terminal:
```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at DivisionByZero1.division(DivisionByZero1.java:8)
    at DivisionByZero1.main(DivisionByZero1.java:12)
```

**Exception** `java.lang.ArithmeticException` is thrown in line 8, and handled by the default exception handler which dumps the contents of the method activation stack.

## try-catch-finally Construct

```
try {
    <statements>
} catch (<exception type> <parameter>) {
    <statements>
} ...
    catch (<exception type> <parameter>) {
    <statements>
} finally {
    <statements>
}
```

The catch block and the finally block are optional, but at least one of them must be specified with a try block.

## try-catch-finally Execution

```
Activity Diagram in UML

1. [no exception]
2. [exception1]
3. [exception2]
4. [exceptionn]
```

- Normal execution continues
- Execution aborted and exception propagated
Example: Explicit Throwing and Catching of an Exception

```
1. // DivisionByZero3.java
2. public class DivisionByZero3 { 
3.     public static void division() { 
4.         int num1 = 10; 
5.         int num2 = 0; 
6.         try { 
7.             if (num2 == 0) throw new ArithmeticException(); 
8.             System.out.println(num1 + " / " + num2 + " = " + (num1 / num2)); 
9.         } 
10.        catch (ArithmeticException Exception) { 
11.            System.out.println(Exception); 
12.        } 
13.    } 
14.    public static void main(String args[]) { 
15.        division(); 
16.    } 
17. }
```

Output on the terminal:
java.lang.ArithmeticException

Example: Explicit Definition, Throwing and Handling of Exception

```
1. // DivisionByZero4.java
2. public class DivisionByZero4 { 
3.     public static void division() { 
4.         int num1 = 10; 
5.         int num2 = 0; 
6.         try { 
7.             if (num2 == 0) throw new DivisionByZero(); 
8.             System.out.println(num1 + " / " + num2 + " = " + (num1 / num2)); 
9.         } 
10.        catch (DivisionByZero Exception) { 
11.            System.out.println(Exception); 
12.        } 
13.    } 
14.    public static void main(String args[]) { 
15.        division(); 
16.    } 
17. }
18. class DivisionByZero extends Exception {  // Checked Exception. 
19.    DivisionByZero() { super("Integer division by 0"); } 
20. }
```

Output on the terminal:
DivisionByZero: Integer division by 0

"Checked" Exceptions

- Java demands that checked exceptions be declared in a throws-clause of the method if these exceptions are not handled by a try-catch-block in the method.

```
1. // DivisionByZero5.java
2. public class DivisionByZero5 { 
3.     public static void division() throws DivisionByZero { 
4.         int num1 = 10; 
5.         int num2 = 0; 
6.         if (num2 == 0) 
7.             throw new DivisionByZero(); 
8.         System.out.println(num1 + " / " + num2 + " = " + (num1 / num2)); 
9.     } 
10.    public static void main(String args[]) throws DivisionByZero { 
11.        division(); // can throw a checked exception which must be declared 
12.        // in the throws-clause of the main() method! 
13.    } 
14. } 
15. class DivisionByZero extends Exception {  // Checked Exception definition 
16.     DivisionByZero() { super("Integer division by 0"); } 
17. }
```

Output on the terminal:
Exception in thread "main" DivisionByZero: Integer division by 0

Example II: Exception Propagation

1a. Let us play. 
1b. Play with me. 
1c. Throw a ball. 
2. Throw a ball. 
3. Catch a ball. 

tennisball 
baseball 
golfball 

A player can only catch the ball which it can play with. After an exception is thrown, program control goes to the first method activation which can handle the exception.
Example II: Exception Propagation (cont.)

// (Checked) Exception definitions
class Tennisball extends Exception {}; class Baseball extends Exception {}; class Golfball extends Exception {};

// Tennis player
class TennisPlayer {
    public void startplay() {
        try {
            new BaseballPlayer().play();
            System.out.println("Nobody wants to play tennis!");
        } catch (Tennisball tb) {
            System.out.println("Tennis is fun!");
        }
    }
    public static void main(String args[]) {
        new TennisPlayer().startplay();
    }
}

// Golf player
class GolfPlayer {
    public void play() throws Tennisball, Baseball, Golfball {
        try {
            new BallThrower().throwBall();
            System.out.println("Nobody wants to play golf!");
        } catch (Golfball gb) {
            System.out.println("Golf is fun!");
        } catch (Baseball bb) {
            System.out.println("Baseball is fun!");
        } catch (Tennisball tb) {
            System.out.println("Tennis is fun!");
        }
    }
    public static void main(String args[]) {
        new GolfPlayer().play();
    }
}

// Ball thrower
class BallThrower {
    public void throwBall() throws Tennisball, Baseball, Golfball {
        int ballChoice = 2;
        switch (ballChoice) {
            case 1: throw new Tennisball();
            case 2: throw new Baseball();
            case 3: throw new Golfball();
        }
    }
}

// Note that execution of the method throwBall() is aborted when an exception is thrown.

Example II: Exception Propagation (cont.)

Output when ballChoice = 1 (i.e. tennisball):
Tennis is fun!

Method Activation Stack

Call sequence
BallThrower.throwBall() -> GolfPlayer.play() -> BaseballPlayer.play() -> TennisPlayer.startplay() -> TennisPlayer.main()

Throws a tennisball.
Normal execution continues here in the catch-block.

Note that the activation of both GolfPlayer.play() and BaseballPlayer.play() are "unwound".

Output when ballChoice = 2 (i.e. a baseball):
Baseball is fun!
Nobody wants to play tennis!

Method Activation Stack

Call sequence
BallThrower.throwBall() -> GolfPlayer.play() -> BaseballPlayer.play() -> TennisPlayer.startplay() -> TennisPlayer.main()

Throws a baseball.
Normal execution continues here in the catch-block.

Note that the activation of GolfPlayer.play() is "unwound".
Example II: Exception Propagation (cont.)

Output when ballChoice = 3 (i.e. a golfball):
Golf is fun!
Nobody wants to play baseball!
Nobody wants to play tennis!

Method Activation Stack

Call sequence

BallThrower.throwBall() → GolfPlayer.play() → BaseballPlayer.play() → TennisPlayer.startplay() → TennisPlayer.main()

Throws a golfball.

Normal execution continues here in the catch-block.

Example II: Exception Propagation (cont.)

Output when ballChoice > 3 or < 1:
Nobody wants to play golf!
Nobody wants to play baseball!
Nobody wants to play tennis!

Method Activation Stack

Call sequence

BallThrower.throwBall() → GolfPlayer.play() → BaseballPlayer.play() → TennisPlayer.startplay() → TennisPlayer.main()

Normal return

Example using finally block

// (Checked) Exception definitions
class Tennisball extends Exception {};
class Baseball extends Exception {};
class Golfball extends Exception {};

// Tennis player
public class TennisPlayer {
    void startplay() {
        try {
            new BaseballPlayer().play();
            System.out.println("Nobody wants to " + "play tennis!");
        } catch (Tennisball tb) {
            System.out.println("Tennis is fun!");
        } finally {
            System.out.println("I only play tennis!");
        }
    }
    public static void main(String args[]) {
        new TennisPlayer().startplay();
    }
}
// Baseball player
class BaseballPlayer {
    void play() throws Tennisball {
        try {
            new GolfPlayer().play();
            System.out.println("Nobody wants to " + "play baseball!");
        } catch (Baseball bb) {
            System.out.println("Baseball is fun!");
        } finally {
            System.out.println("I only play baseball!");
        }
    }
}

// Golf player
class GolfPlayer {
    void play() throws Tennisball, Baseball {
        try {
            new BallThrower().throwBall();
            System.out.println("Nobody wants to" + " play golf!");
        } catch (Golfball gb) {
            System.out.println("Golf is fun!");
        } finally {
            System.out.println("I only play golf!");
        }
    }
}

// Ball thrower
class BallThrower {
    void throwBall() throws Tennisball, Baseball, Golfball {
        int ballChoice = 2;
        switch (ballChoice) {
            // The code to throw different types of balls
        } }
    }

Output from the program:
I only play golf!
Baseball is fun!
I only play baseball!
Nobody wants to play tennis!
I only play tennis!

Compare this output with the earlier example without the finally block.

Exceptions and Inheritance

- An overridden method in a subclass can only specify in its throws clause **all or a subset of exceptions** specified in the throws clause of the method definition in the superclass.

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

  // In subclass B which extends class A
  public void aMethodWithManyExceptions()
  throws Exception1, Exception3 {...}

- Since all exceptions are objects, a catch block can specify an exception superclass as parameter.
  - A catch block can **shadow** other catch blocks which specify a subclass.
  - Compiler flags such situations as errors.

  ... try { ...}
  catch (Exception e) {...} // superclass shadows
  catch (ArithmeticException a) {...} // subclass.

Summary of Exception Handling

- "Throw or Declare"-rule for **checked exceptions**:
  - The checked exceptions which a method can throw but does not catch must be declared in a throws clause of the method.
  - A method can catch exceptions which it throws and handle them in catch blocks.
  - A method can explicitly throw an exception using the throw-statement.

... aMethod(...). throws Exception1, Exception2, ..., Exceptionn {
  try {
    // statements which can lead to one of the following exceptions
    // being thrown: Exception1, Exception2, ..., Exceptionn and
    // Exceptiona, ..., Exceptionm
    throw new Exception1(); // throw an exception
  } catch (Exceptiona a) { // exception handler for Exceptiona
    // code to handle exception a
  } ...
  catch (Exceptionn m) { // exception handler for Exceptionn
    // code to handle exception m
  } finally {
    // code to cleanup
  }
}