OBJECT-ORIENTED SYSTEMS DESIGN
[Exercise]: Exception Handling

2022/05/11

Department of Computer Science,
Hanyang University

Lecturer: Taeuk Kim
TA: Taejun Yoon, Seong Hoon Lim, Young Hyun Yoo

Exception Handling Basics

Chapter 9.1

What is ‘Exception Handling’ ?

- **One way to divide the task of designing and coding a method is to code two main cases separately**
  - The case where nothing unusual happens.
  - The case where exceptional things happen.

I want to design the “equals” method in the Person class!

My “equals” design

1. Check whether name is equal to other.
2. Check whether born is equal to other.
3. Check whether died is equal to other.
4. If 1~3 are all true, return true.
5. else, return false.

What is ‘Exception Handling’ ?

- **One way to divide the task of designing and coding a method is to code two main cases separately**
  - The case where nothing unusual happens.
  - The case where exceptional things happen.

…What if `other` is null?
or this object’s ‘`died`’ is null?

```
public boolean equals(Person other) {
    return (this.name.equals(other.name) &&
            this.born.equals(other.born) &&
            this.died.equals(other.died));
}
```

What is ‘Exception Handling’ ?

- **We can deal with some exceptional cases as below.**

```
public boolean equals(Person other) {
    if(other == null)
        return false;
    else
        return (this.name.equals(other.name) &&
                this.born.equals(other.born) &&
                datesMatch(this.died, other.died));
}
```

But Java library software provides a mechanism that signals when something unusual happens.

Let’s rewrite this code for our purpose!

What is ‘Exception Handling’ ?

- **The basic way of handling exceptions in Java consists of the *try-throw-catch* trio.**

First, Try this block

If something wrong happens, print our error messages, and return false.

```
public boolean equals(Person other) {
    try {
        return (this.name.equals(other.name) &&
                this.born.equals(other.born) &&
                this.died.equals(other.died));
    } catch (Exception e) {
        System.out.println("Some Exception happens!");
        return false;
    }
}
```

What is ‘Exception Handling’ ?

- **It works!**

```java
public class Main {
    public static void main(String[] args) {
        Person myPerson1 = new Person("John", new Date(2000, 1, 1), null);
        Person myPerson2 = null;
        Person myPerson3 = new Person("John", new Date(2000, 1, 1), new Date(2040, 1, 1));

        System.out.println(myPerson1.equals(myPerson2));
        System.out.println(myPerson1.equals(myPerson3));
    }
}
```

Main (1)

```
C:\Users\LSH\.jdks\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\Intel
Some Exception happens!
false
Some Exception happens!
false
```

Throwing Exceptions (`try-throw-catch mechanism`)

- **It is also possible for your own code to throw the exception.**
  - To do this, use a **`throw`** statement inside the **`try`** block in the format.

Throw an exception on purpose

Can use our error message with **`getMessage()`** method

```
public boolean equals(Person other) {
    try {
        if(other == null)
            throw new Exception("Object to be compared is null");
        return (this.name.equals(other.name) &&
                this.born.equals(other.born) &&
                this.died.equals(other.died));
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return false;
    }
}
```

Throwing Exceptions (`try-throw-catch mechanism`)

- **It works!**

```java
public class Main {
    public static void main(String[] args) {
        Person myPerson1 = new Person("John", new Date(2000, 1, 1), null);
        Person myPerson2 = null;
        Person myPerson3 = new Person("John", new Date(2000, 1, 1), new Date(2040, 1, 1));

        System.out.println(myPerson1.equals(myPerson2));
    }
}
```

Main (1)

```
C:\Users\LSH\.jdks\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\Intell
Object to be compared is null
false
```

Multiple Exception Classes Example

- **There are more exception classes than just the single class `Exception`.**
  - For example:

    `IOException`

    `NoSuchMethodException`

    `FileNotFoundException`

    …

  - But, If we implement exception handling as below, all of exceptions will be treated with the same handling block.
    - Because all of Exception types are inherited from the '`Exception`' superclass.

```
catch (Exception e) {
    System.out.println("Exception occurs!");
    System.exit(0);
}
```

Multiple Exception Classes Example

```java
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = 1, b = 1;

        while(true) {
            try {
                System.out.println("Enter two integers");
                // Can throw an Exception (non-integer input)
                a = sc.nextInt();
                b = sc.nextInt();

                // Can throw an Exception (divide by zero)
                System.out.println("A / B = " + (a / b));
            } catch (Exception e) {
                System.out.println("Exception occurs!");
                System.exit(0);
            }
        }
    }
}
```

Let's represent this more specifically!

Multiple Exception Classes Example

- **'try' block might throw `'InputMismatchException'` and `'ArithmeticException'`.**
  - So, We use 2 catch block for each Exception.

```java
while(true) {
    try {
        System.out.println("Enter two integers");
        // Can throw an Exception (non-integer input)
        a = sc.nextInt();
        b = sc.nextInt();

        // Can throw an Exception (divide by zero)
        System.out.println("A / B = " + (a / b));
    } catch (InputMismatchException e) {
        // If nextInt reads non-integer input
        System.out.println("Type \"LEGAL\" integer number!");
        break;
    } catch (ArithmeticException e) {
        // If integer is divided by 0
        System.out.println("Integer can't be divided by zero!");
        continue;
    }
}
```

```
Enter two integers
12 4
A / B = 3
Enter two integers
4 0
Integer can't be divided by zero!
Enter two integers
5 a
Type "LEGAL" integer number!

Process finished with exit code 0
```

Defining Exception Classes Example

- **Instead of using a predefined class, exception classes can be programmer-defined.**

```java
package Chapter5_exercise.util;

public class MyException extends Exception {
    public MyException() {
        super("My exception happens!");
    }

    public MyException(String message) {
        super(message);
    }

    @Override
    public String getMessage() {
        return "[MyException] " + super.getMessage();
    }
}
```

Defining Exception Classes Example

- **Instead of using a predefined class, exception classes can be programmer-defined.**

Throws our new Exception

```java
while(true) {
    try {
        System.out.println("Enter two integers");
        // Can throw an Exception (non-integer input)
        a = sc.nextInt();
        b = sc.nextInt();

        // Programmer-defined Exception example
        if(a < 0 && b < 0)
            throw new MyException("a and b are both negative!");

        // Can throw an Exception (divide by zero)
        System.out.println("A / B = " + (a / b));
    } catch (InputMismatchException e) {
        // If nextInt reads non-integer input
        System.out.println("Type \"LEGAL\" integer number!");
        break;
    } catch (ArithmeticException e) {
        // If integer is divided by 0
        System.out.println("Integer can't be divided by zero!");
        continue;
    } catch (MyException e) {
        System.out.println(e.getMessage());
        break;
    }
```

Our overridden `getMessage` works!

```
Main
C:\Users\LSH\.jdks\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\Jet
Enter two integers
-2 -3
[MyException] a and b are both negative!
```

Throwing Exceptions in Methods

Chapter 9.2

Checked and Unchecked Exceptions

- **Exceptions that are subject to the catch or declare rule are called *checked* exceptions.**
  - The compiler checks to see if they are accounted for with either a catch block or a throws clause.
  - The classes `Throwable`, `Exception`, and all descendants of the class `Exception` are checked exceptions.
    - Exception: `RuntimeException`

- **All other exceptions are *unchecked* exceptions.**

- **The class `Error` and all its descendant classes are called *error classes*.**
  - Error classes are *not* subject to the Catch or Declare Rule.

Throwing an Exception in a Method

- **Checked exceptions must follow the Catch or Declare Rule.**
  - Programs in which these exceptions can be thrown will not compile until they are handled properly.

```
public class Main {
    public static void main(String[] args) {
        byte[] list = {'a', 'b', 'c', 'd'};

        System.out.write(list);
    }
}
```

Unhandled exception: java.io.IOException

Add exception to method signature Alt+Shift+Enter More actions... Alt+Enter

```
public void write(
    byte[] buf
)
throws java.io.IOException
```

Writes all bytes from the specified byte array to this stream. If automatic flushing is enabled then the `flush` method will be invoked on the underlying output stream.
Note that the bytes will be written as given; to write characters that will be translated according to the platform's default character encoding, use the `print(char[])` or `println(char[])` methods.

Checked Exception Example : IOException

Throwing an Exception in a Method

- **Checked exceptions must follow the Catch or Declare Rule.**
  - Programs in which these exceptions can be thrown will not compile until they are handled properly.

```java
public class Main {
    public static void main(String[] args) {
        byte[] list = {'a', 'b', 'c', 'd'};
        try {
            System.out.write(list);
        } catch (IOException e) {
            System.out.println("IOException occurs!");
        }
    }
}
```

Handle the exception in the method (Catch)

```java
public class Main {
    public static void main(String[] args) throws IOException {
        byte[] list = {'a', 'b', 'c', 'd'};

        System.out.write(list);
    }
}
```

Throw the exception handling to the caller method (Declare)

Practice

Exercise/WeekN/Main.java,
/MyException.java,

Practice for Today

- **Define a method '`isCoprime(int a, int b)`'**
  - Returns true if a and b are coprime (서로소).
  - It throws an exception when
    - One of a and b is less or equal to 1.
    - a and b are the same number.
    - Both a and b are larger than 10000.

- **MyException class**
  - Handle all of the exception in '`isCoprime`' with this exception class.
  - Override `getMessage()` method .
    - Print error message(your message must contain 'cause' and 'possible solution').
    - (e.g., "[ArithmeticException] Integer can't be divided by 0 ; Change divisor to non-zero value")

- **Main class**
  - Takes 2 integers as an input, then check whether two integers are coprime.
    - Format for printing result is free.
  - Handle all of Exception in main.
    - 'InputMismatchException', Exception in '`isCoprime`'.
    - Handling for 'InputMismatchException' is same as 'MyException'.
  - Ensure that requirements are satisfied.

Time for Practice

Get it started, and ask TAs if you are in a trouble.