ITI 1121. Introduction to Computer Science II

Laboratory 2

Winter 2016

Part I
Data type convension

Objectives

- Learning about the concept of data type convension;
- Becoming familiar with different kinds of data type convensions.

1 A short discussion on implicit and explicit type casting

Your teaching assistant will first lead a short discussion on implicit and explicit type casting. Please pay attention to the concepts and ask questions if it is needed.

2 Wrappers for primitive types

You now know that Java can perform certain type conversions automatically (implicitly). It does it when it knows that no information will be lost, e.g. the content of an **int** variable can be safely copied to a **long** since a **long** can be assigned any of the values of an **int** and many more. The mirror operation, assigning the value of a variable of type **long** to a variable of type **int**, cannot be executed automatically since some of the values of a **long** cannot be represented using the amount of space reserved for an **int**.

Becoming effective at programming. Throughout the semester, several tips will be presented to help you developing your programming skills. One of the most important skills is debugging. Initially, students tend to spend quite a bit of time debugging their programs, sometimes looking at the wrong segments of their program. We will have more to say about debugging strategies in the next laboratories, but for now we will focus on the error messages. Understand and learn the error messages that are reported by the compiler. A good way to do this is to create small test programs that cause the error. For instance, create a class called **Test** that has a main method. In the main method, declare a variable of type **int**, then assign the value **Long.MAX_VALUE**. This is the largest value for a **Long**, this must cause an error. Try this for yourself. See what error message comes out.

Sometimes, the logic of your program requires you to perform a type conversion. It must be done with special precautions. **Always guard a type casting with the proper test to ensure that the value is in the proper range**. When doing a type cast, you are relieving the compiler of one of its important duties, which is to make sure that all the types of the sub-expressions are compatible. It is as if you were saying "I know what I am doing, please allow me to store this value in that variable".

```
long l;
```

```
...
if ( l >= Integer.MIN_VALUE && l <= Integer.MAX_VALUE ) {
   int i = (int) l;
   ...
}
```

Type casting cannot be used for transforming a **String** into a number! Each primitive data type has a corresponding "wrapper" class. This is a class that has a similar name, for instance, the wrapper class for an **int** is called **Integer** (the classes, **Double**, **Boolean**, **Character**, etc. also exist). As the name "wrapper" suggests, such class packages a value inside an object. Like this.

```
public class Integer {
    private int value;
    public Integer( int v ) {
        value = v;
    }
    ...
}
```

Later on, the idea of packaging a primitive value inside an object will be necessary (in the lectures related to abstract data types). However, for now it is another aspect of the wrapper classes that is our focus. Each "wrapper" class also provides a collection of methods that are related to its corresponding primitive type. Why don't you see for yourself.

- Go to http://docs.oracle.com/javase/8/docs/api/overview-summary.html;
- This is the documentation of the standard library for Java 8.0;
- Go to the package **lang** (which is always implicitly imported into your program);
- http://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html;
- Scroll down a little bit, and visit the page for the class **Integer**;
- http://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html;
- Now locate the method **parseInt(String s)**;

Write a new class, called **Sum**, that converts to integers all the elements found on the command line, sums these numbers and prints the result.

```
> javac Sum.java
> java Sum 1 2 3 4 5
The sum is 15
```

Part II
Object oriented programming

Objectives

- Implementing simple classes;
- Creating associations between classes;
- Explore further the notion of encapsulation.

3 Combination

Implement a class, called `Combination`, to store three integer values (ints).

1. declare the necessary instance variables to store the three integer values;
2. create a constructor, `public Combination(int first, int second, int third)`, to initialize the values of this object.
3. implement the instance method `public boolean equals(Combination other)`, such that `equals` return true if `other` contains the same values, in the same order, as this Combination; the order is defined by the order of the parameters of the constructor, given the following,

   ```
   Combination c1;
   c1 = new Combination( 1, 2, 3 );
   Combination c2;
   c2 = new Combination( 1, 2, 3 );
   Combination c3;
   c3 = new Combination( 3, 2, 1 );
   ```

   then `c1.equals(c2)` is `true` but `c1.equals(c3)` is `false`;
4. finally, implement the method `public String toString()`, to return a **String** representation of this object, where the first, second and third values are concatenated and separated by ":" symbols. E.g.

   ```
   Combination c1;
   c1 = new Combination( 1, 2, 3 );
   System.out.println( c1 );
   ```

   displays "1:2:3".

The interface of the class **Combination** consists therefore of its constructor, the method `equals` and the method `toString`.

Ideally, the input data should be validated. In particular, all the values should be in the range 1 to 5. However, since we do not yet have the tools to handle exceptional situations, we will assume (for now) that all the input data are valid!

Note: Simpler code is better and it is worthwhile spending time cleaning up code even if it was already working

Hint: my implementation is approximately 20 lines long, not counting blank lines. This is not a contest to write the shortest class declaration. I am providing this information so that you can evaluate the relative complexity of the tasks.

4 DoorLock

Create an implementation for the class `DoorLock` described below.

1. declare an integer constant, called `MAX_NUMBER_OF_ATTEMPTS`, that you will initialize to the value 3;
2. instance variables. The class `DoorLock` must have the necessary instance variables to **i)** store an object of the class `Combination`, **ii)** to represent the property of being opened or closed, **iii)** to represent its activation state (the door lock is activated or deactivated), and **iv)** to count the number of unsuccessful attempts at opening the door;

3. the class has a single constructor, `DoorLock( Combination combination )`, which initializes this instance with a combination. When a door lock is first created, the door lock is closed. Also, when the object is first created, it is activated and the number of failed attempts at opening it should be zero;

4. implement the instance method `public boolean isOpen()` that returns `true` if this door lock is currently opened and `false` otherwise;

5. implement the instance method `public boolean isActivated()` that returns `true` if this door lock is currently activated and `false` otherwise.

6. implement the instance method `public void activate( Combination c )` that sets the instance variable "activated" to `true` if the parameter `c` is "equals" to the combination of this object;

7. finally, implement the instance method `public boolean open( Combination combination )` such that **i)** an attempt is made at opening this door lock only if this door lock is activated, **ii)** if the parameter combination is "equals" to the combination of this door lock, set the state of the door to be open, and the number of failed attempts should be reset to zero, **ii)** otherwise, i.e. if the wrong **Combination** was supplied, the number of failed attempts should be incremented by one, **iii)** if the number of failed attempts reaches `MAX_NUMBER_OF_ATTEMPTS`, this door lock should be deactivated.

Hint: my implementation is approximately 40 lines long, not counting the blank lines.

5 SecurityAgent

Implement the class `SecurityAgent` described below.

1. instance variables. A security agent is responsible for a particular door lock. Declare the necessary instance variables such that a **SecurityAgent i)** remembers (stores) a **Combination** and **ii)** has access to this particular `DoorLock`, i.e. maintains a reference to a `DoorLock` object;

2. implement a constructor with no parameter such that when a new **SecurityAgent** is created **i)** it creates a new **Combination** and stores it, **ii)** it creates a new **DoorLock** with this saved **Combination**. For the sake of simplicity, you may decide to always use the same combination:

   ```
   Combination secret;
   secret = new Combination( 1, 2, 3);
   ```

   if `secret` is the name of the instance variable that is used to remember the **Combination**. Or, you can let your **SecurityAgents** use their imagination, so that each **SecurityAgent** has a new **Combination** that it only knows.

   ```
   int first = (int) ( Math.random()*5 ) + 1;
   int second = (int) ( Math.random()*5 ) + 1;
   int third = (int) ( Math.random()*5 ) + 1;

   secret = new Combination( first, second, third );
   ```

   Valid values must be in the range 1 to 5;

   Alternative way to generate a random combination

```
java.util.Random generator = new  java.util.Random();
int first = generator.nextInt(5) + 1;
int second = generator.nextInt(5) + 1;
int third = generator.nextInt(5) + 1;
```

3. implement the instance method `public DoorLock getDoorLock()` that returns a reference to the saved **DoorLock**;
4. implement the instance method `public void activateDoorLock()` that simply reactivates the particular **DoorLock** that this **SecurityAgent** is responsible for, with the saved secret Combination.

Hint: my implementation is approximately 15 lines long, not counting the blank lines.

Test

A **Test** class is provided with this laboratory. I suggest that you only use it when all your classes have been successfully implemented and tested.

The **Test** class consists of a main method that **i)** creates a **SecurityAgent** called bob, it asks bob for an access to the **DoorLock** that bob is in charge of, then it applies a "brute force" approach to unlock the door. After three failures, it has to ask bob to re-activate the lock. When the lock has been unlocked, it prints the combination of the lock, as well as the number of attempts that were necessary to open the door lock. Here are examples of successful runs:

```
% java Test
Success!
Number of attemtps: 266
The combination is: 3:1:3

% java Test
Success!
Number of attemtps: 1
The combination is: 4:1:5

% java Test
Success!
Number of attemtps: 115
The combination is: 2:2:1

% java Test
Success!
Number of attemtps: 383
The combination is: 2:4:5

% java Test
Success!
Number of attemtps: 89
The combination is: 3:5:1
```

Warning: if the classes are not properly implemented this test can run forever, if you encounter such a situation use Ctrl-c to kill the process.

Hint: build test classes for each of the three classes that you implement.

- www.site.uottawa.ca/~turcotte/teaching/iti-1121/lectures/t01/Test.java

Part III
Quiz

Reference variables (1 mark)

```
public class Quiz {
    public static void main(String[] args) {
        String s = null;
        if (s.length() > 0) {
            System.out.println(s);
        } else {
            System.out.println("empty string");
        }
    }
}
```

Which of the following statement best characterizes the above Java program:

1. The program runs without error, but displays nothing on the console.
2. Displays a message of the form "String@5e8fce95" on the console.
3. Displays "empty string" on the console.
4. Produces a compile-time error:

   ```
   Quiz.java:4: variable s might not have been initialized
   if (s.length() > 0) {
       ^
   1 error
   ```

5. Produces a run-time error:

   ```
   Exception in thread "main" java.lang.NullPointerException
   at Quiz.main(Quiz.java:4)
   ```

Submit your answer on Blackboard Learn:

- https://uottawa.blackboard.com/

Last Modified: January 20, 2016