ITI 1121. Introduction to Computer Science

Laboratory 3

1 Part I: Data type conversion

1.1 Objectives

- Learning about the concept of data type conversion
- Becoming familiar with different kinds of data type conversions

1.2 Short discussion on implicit and explicit type casting

Short discussion lead by the TA. Please pay attention to the concepts and ask questions if it is needed.

1.3 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
```

2 Part II: Object oriented programming

2.1 Objectives

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

2.2 Combination

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

1. Declare the necessary instance variables to store the four integer values
2. Create a constructor, public Combination(int first, int second, int third, int fourth), 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, 4 );
Combination c2;
c2 = new Combination( 1, 2, 3, 4 );
Combination c3;
c3 = new Combination( 4, 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, third and fourth values are concatenated and separated by : symbols. E.g.

```
Combination c1;
c1 = new Combination( 1, 2, 3, 4 );
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 10. 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!

2.3 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 6;
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 state (the door lock is enabled or disabled)
   - 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 enabled 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 isEnabled() that returns true if this door lock is currently enabled and false otherwise.
6. Implement the instance method public void enable(Combination c) that sets the instance variable *enabled* 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 enabled
   - 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

III Otherwise, i.e. if the wrong Combination was supplied, the number of failed attempts should be incremented by one

IV If the number of failed attempts reaches MAX_NUMBER_OF_ATTEMPTS, this door lock should be disabled.

2.4 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

   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, 4);
```

   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()*10 ) + 1;
int second = (int) ( Math.random()*10 ) + 1;
int third = (int) ( Math.random()*10 ) + 1;
int fourth = (int) ( Math.random()*10 ) + 1;
secret = new Combination( first, second, third, fourth );
```

   Valid values must be in the range 1 to 10; Alternative way to generate a random combination

```
java.util.Random generator = new java.util.Random();
int first = generator.nextInt(10) + 1;
int second = generator.nextInt(10) + 1;
int third = generator.nextInt(10) + 1;
int fourth = generator.nextInt(10) + 1;
```

3. Implement the instance method public DoorLock getDoorLock() that returns a reference to the saved DoorLock

4. Implement the instance method public void enableDoorLock() that simply re-enable the particular DoorLock that this SecurityAgent is responsible for, with the saved secret Combination.

2.5 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:

1. 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.

2. After six failures, it has to ask bob to re-enable the lock.

3. 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: 3:1:3:5

% java Test
Success!
Number of attemtps: 1
The combination is: 4:1:5:2

% java Test
Success!
Number of attemtps: 115
The combination is: 2:2:1:9

% java Test
Success!
Number of attemtps: 383
The combination is: 2:4:5:7

% java Test
Success!
Number of attemtps: 89
The combination is: 6:3:5:1
```

Warning: if the classes are not properly implemented this test can run forever

Test class implementation

```
public class Test {

    public static void main( String[] args ) {

        // Creates a new security agent called bob!
        SecurityAgent bob = new SecurityAgent();

        // Ask bob to give us access to the door lock
        DoorLock aLock = bob.getDoorLock();

        // Let's find bob's secret combination
        Combination c = null;
        boolean open = false;
        int iter = 0;

        while ( !open ) {

            // bob knows the combination and will
            // re-enable the DoorLock

            if ( !aLock.isEnabled() ) {
                bob.enableDoorLock();
            }

            // let's create a new random combination

            int first = (int) ( Math.random()*10 ) + 1;
            int second = (int) ( Math.random()*10 ) + 1;
            int third = (int) ( Math.random()*10 ) + 1;
            int fourth = (int) ( Math.random()*10 ) + 1;

            c = new Combination( first, second, third, fourth );

            // if this combination opens the lock
            // we're done.
            if ( aLock.open( c ) ) {
                open = true;
            }

            iter++;
        }

        System.out.println( "Success!" );
        System.out.println( "Number of attemtps: " + iter );
        System.out.println( "The combination is: " + c );
    }
}
```

2.6 Submission

Instructions:

- Create a directory lab3 123456, where 123456 is replaced by your student number.
- Inside this directory copy the files for each of the classes of the lab (Sum, Combination, SecurityAgent and DoorLock). Only copy the source files for each class (.java files)
- Create a file README.txt which is a text file containing your name, student number and a brief description of the content of the directories:

  Student name: Jane Doe
  Student number: 123456
  Course code: ITI1121
  Lab section: B2
- Create a zip file lab2_123456.zip containing the directory lab2_123456 and all its files
- Verify that your archive is correct by uncompressing it somewhere and making sure that all the files are there.
- Submit the archive to https://uottawa.blackboard.com