CSCE 155 - Java

Lab 6.0 - Methods, Enumerated Types, and Exceptions

Prior to Lab

Before attending this lab:

1. Read and familiarize yourself with this handout.
2. Review the following free resources:
   - Oracle's Tutorial sections on methods and returning values:
     http://docs.oracle.com/javase/tutorial/java/javaOO/methods.html
     http://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html
   - Oracle's Tutorial on Exceptions:
     http://docs.oracle.com/javase/tutorial/essential/exceptions/
   - Oracle's Tutorial on enumerated types:
     http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

Peer Programming Pair-Up

To encourage collaboration and a team environment, labs will be structured in a *pair programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither

the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

1 Lab Objectives & Topics

At the end of this lab you should be familiar with the following

- Basics of enumerated types
- Writing and using methods
- Parameters and arguments in methods, returning values from methods
- How to use exceptions for error handling
- The proper use of enumerated types and methods in solving a given problem

2 Background

Enumerated Types

Enumerated types are data types that define a set of named values. Enumerated types are often ordered and internally associated with integers. The purpose of enumerated types is to organize types that have a fixed and known set of values. Without enumerated types, integer values must be used and the convention of assigning certain integer values to certain types in a logical collection must be remembered or the user is consistently referring to documentation to keep integer values in order. Enumerated types provide a human-readable "tag" to these types of elements, relieving the programmer of continually having to refer to the convention and avoiding errors.

In Java, enumerated types are associated with integers starting at zero. Enumerated types are often placed in their own source files and defined using the `enum` keyword (though in Java enumerated types are full classes). An `enum` class itself also has several static convenience methods (example: `values()` returns a zero-index array of the enumerated types).

Passing Values in Functions

In general there are two ways in which arguments can be passed to a method: by value (where a copy of the variable's value at the time of the method call is passed to the method) or by reference (a reference is passed to the method so that the method can manipulate its contents). In Java however, object references are passed by value.

In Java, any primitive variable passed to a function is passed by value. However, all object variables are references. Java passes references to objects slightly differently: it passes a copy of the reference to the function. Thus, calling an object's methods (in particular any mutators) will call the methods on the original object and any methods that change the state of the object will be realized in the calling method. However, changing the actual reference only changes the local copy of the reference.

Exceptions & Error Handling

Errors in the execution of a program are unavoidable: users may enter invalid input, or the expected resources (files or database connections) may be unavailable, etc. Errors are communicated and handled through the use of exceptions. When an error condition occurs or is detected, the program can throw an exception with a human-readable error message. The type of exception that is thrown can communicate the error in a programmatic manner. When the program executes a piece of code that could potentially result in an exception, it can be handled by using a try-catch block: we try the piece of code and catch (possibly numerous different types of) exceptions. We can deal with each exception in a different manner or simply echo (print) to the user a different message based on the error. A basic example:

```
Integer a, b;
//read in a, b
try {
  if(b == 0) {
     throw new IllegalArgumentException("Division by " +
        "zero is not valid.\n");
  } else {
     c = a / b;
  }

  BufferedWriter out = new BufferedWriter(
        new FileWriter("/etc/passwd"));
  out.write("result = "+c);

} catch(IllegalArgumentException e1) {
     System.err.println("Division by zero is undefined!");
```

```
17          System.exit(1);
18      } catch(SecurityException e2) {
19          System.err.println("You are not root, you " +
20             "can't write to the password file! ");
21          System.exit(1);
22      }
```

3 Activities

Clone the GitHub project for this lab using the following URL: https://github.com/cbourke/CSCE155-Java-Lab06

A flower shop sells various arrangements of a dozen flowers (roses, lilies, daisies) in two colors each (red or white) with a choice of bouquet or vase. You are given an (incomplete) source file, `Florist.java`, of a program that takes the order for a flower arrangement from the user and displays the cost. The program uses three enumerated types to define the type of flowers, color and flower arrangement. Your task is to complete the program and implement the getCost function to compute the cost based on the following rules.

1. The base price for each of the flowers is described in Table 1

Flower	Price
Roses	$30
Lilies	$20
Daisies	$45

Table 1: Flower Prices

2. Red Lilies and Red Daisies have an added cost of $5 and White Roses have an added cost of $10. There is no additional cost for other flower/color combinations
3. Bouquets are free of charge while vases add an additional $10 charge to the total

3.1 Defining Enumerated Types

To complete the program, perform the following tasks.

1. Insert the required items in the 3 enumerated types. The `Color` enumerated type has already been completed for you.
2. After taking input from the user, the program calls the function `getCost` to determine the cost. This function takes three input parameters, the flower type, the color and the arrangement and returns the cost (a `double`).

a) The method declaration has been provided for you. Implement the definition of this function to return the cost of a given arrangement using the costs shown on the price list.

b) In the `main` method, declare the other required variable(s), complete the method call with appropriate arguments and print the cost. What's the data type of the variable cost?

3. Run your program within Eclipse and answer the questions in your handout.

3.2 Error Handling

Due to a harsh winter, Red Daisies are no longer available. We will make the appropriate changes to the `getCost()` method to accommodate this change and to communicate errors by throwing an Exception.

1. Check the input at the beginning of the function to check for an invalid choice. If the input is for red daisies, throw a new `IllegalArgumentException` with an appropriate message.
2. In the main method, add a try-catch block to catch the exception above and handle it appropriately (simply echo an error message to the user; see the previous example).
3. In addition, you should add another catch block to catch invalid inputs entered by the user. The type of exception thrown could be another `IllegalArgumentException` with a different error message.
4. Test your changes and demonstrate them to a lab instructor.

4 Separating Your Code

Up to this point, the `enum` types we used were declared in the class that was using them. As mentioned earlier, enums can be their own source files. Make your code more modular by organizing each of your enumerated types (`Flower` , `Color` , `Arrangement`) into their own source files.

1. Create class files for each `enum` , and remove the original declarations inside the `Florist` class, or comment them out
2. Be sure to store your new `enum` classes in the same package (`unl.cse.florist`) as `Florist.java`
3. Make any necessary changes to your program and demonstrate it to a lab instructor

5 Advanced Activity (Optional)

Large projects require even more abstraction and tools to manage source code and specify how it gets built. For Java, a popular build tool is Apache Ant (http://ant.apache.org/). Ant is a build utility that builds Java projects as specified in a special XML file (`build.xml`). The build file specifies how pieces get built and the inter-dependencies on components. Familiarize yourself with Ant by reading the following tutorials.

- http://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html
- http://www.vogella.com/articles/ApacheAnt/article.html

Provided in project is an example `build.xml` file. Modify it for the code base you created in this lab and use it to compile and run the code from the command line. In particular, from the command line:

1. Place all source files into a folder named `src` in the same directory as the `build.xml` file (it should be like this already)
2. Modify the `build.xml` file appropriately by specifying what your main executable class is. To do this, modify the `main.class` property's value to the fully qualified (full path name) class that you wish to run.
3. Compile your project by executing the following command: `ant compile`
4. Run your project by executing the following command: `ant run`