Lab 9 : Testing and Multi-Class Programs

Topics covered `enum`, testing, multi-class programs

Exercise 1: Write an `enum` to hold the days of the week. Use this to write a program that takes one command-line argument that is a number representing the day of the week, and print out what day that number represents, as well as if that day is a weekend or not.

Note: If you have an `enum` called `Weekdays`, you can get an array of all possible values for the `enum` with the method `Weekdays.values()`.

```
> java WhatDayIs 3
   Day 3 of the week is: WEDNESDAY. It is not a weekend.
> java WhatDayIs 0
   Day 0 of the week is: SUNDAY. It is a weekend.
```

Exercise 2: Change the program to take in two arguments: the current day of the week and a number of days after that. The program should then output what day of the week it will be that number of days from the starting day.

```
> java DaysFromNow 1 9
   The current day is MONDAY, and in 9 days time it will be WEDNESDAY.
```

Exercise 3: Design a set of tests for the following method:

```
1  /*
2   * Calculate a logarithm to any given base.
3   *    value - the value to find the log of
4   *    base - the base for the logarithm
5   */
6  public static double logToBaseN(double value, int base)
```

What kind of testing is this?

Exercise 4: Design a set of tests for the following method:

```
1  /*
2   * Get the index of the first pair of double letters that are lowercase
3   * in an input string. If no such pair exists, return -1.
4   */
5  public static int findFirstLowercasePair(String input) {
6     int index = 0;
7     while(index < input.length()) {
8        char ch1 = input.charAt(index);
9        char ch2 = input.charAt(index + 1);
10       if(ch1 == ch2) {
11          if(Character.isLowerCase(ch1)) {
12             return index;
13          }
14       }
15       index = index + 1;
16    }
17    return -1;
18 }
```

What kind of testing is this? Can you design a test case that will point out the error in this code?

Exercise 5: Count the number of execution paths in the following code. Don't try to figure out what it does – you'll be there all day!

```
int foo(int a, int b, int c) {
   if (b < 10) {
      if (  (a + b) > 10 ) {
         return bar(a, b);
      }
      if ( ( a - b ) > c ) {
         return baz(a, b, c, a - b);
      }
   } else if ( b > 15) {
      return baz( 0, a, c, b);
   } else  {
      return a + b * 9 - ( c + 1 ) / 2;
   }
   return a * (b + 1);
}

int bar( int x, int y ) {
   if ( x > 0 && y < 17) {
      return baz(0, 0, 2, x);
   } else if ( y >= 17 ) {
      return  x * y / (x+1);
   } else {
      return 3 * x;
   }
}

int baz( int c, int d, int e, int f ) {
   if ( e != 0 )
      return d + e + 5;
   if (c < d)
      return bar(d, e);
   else if (c > d)
      return foo(f, 11, 2);
   else
      return 7 + d;
}
```

Exercise 6: Consider a class that represents a person. A person has the following attributes: first, middle, and last names, age, height, number of eyes, number of legs, favourite cereal and town of birth.

Come up with as many class invariants as you can for a person. Which attributes have constant rules across all people? Can you come up with any other class invariants for attributes not listed here?

Exercise 7: For each of the three classes you created lab 7 and lab 8 (`Pet`, `OrderedPair` and `PasswordGenerator`), design a set of tests that will ensure the classes behave as expected. Make sure that each of the constructors and methods in each class work as expected. If you haven't already, you should now be able to use these tests to write the classes.

Exercise 8: You are going to write a program that manages a list of contacts (like the one in your phone). You will need to write the following classes:

- `ContactList` - a class that holds a collection of `Contact` objects. This class should be able to save its contents to a file, and also load them back again.
- `Contact` - a class representing a contact. Every `Contact` has a name, an `Address`, an array of `EmailAddresses` and an array of `PhoneNumbers`.

- `Address` - an address has two street address lines, a postcode, a state and a country.
- `EmailAddress` - an email address contains a string to represent the email address, as well as whether the email is personal, work, or other.
- `PhoneNumber` - a phone number contains a string to represent the phone number, as well as whether the phone number is personal, work, or other.

The amount of detail that you put into this is up to you.

Consider where you can use enums: for example storing if an email/phone number is work, personal or other could be stored as an enum; so could an address' state and country, if you wish to limit the list.

Consider what are valid entries for email addresses, phone numbers and addresses. Add as much or as little validation code (bad input checking) as you feel is necessary to make sure the data is consistent. For example what kinds of characters should you be able to put in a phone number?

Make sure that you create a set of tests and perform regression testing. Every time that you update your program, run the tests again to make sure that you haven't broken any existing functionality.

Extension 1: Write an `enum` class that represents valid coin denominations. The `enum` should have a constructor so that you can give each denomination a value. What kind of data would this store? What kind of methods would you want to be able to perform on a coin?

Try writing the payment part of a vending machine using this `enum`, and see if you can make the vending machine only accept valid coin denominations this way.

Extension 2: Look into how you can store and manipulate images in Java. In particular, the `BufferedImage` class will be a good place to look. Modify your contact list program to also store an optional contact picture for each `Contact`. How would you store this when saving the file?