Test 3 (Final Exam) Review Question Key CSC 111B - Processing - Fall 2010

1. A Vehicle is composed of the following five variables:
   String manufacturer, String make, int year, int numberOfWheels , double engineSizeInLiters

   a. Write a constructor for the Vehicle class that takes parameters that set all 5 attributes

   See the screenshot below.

   b. Write a constructor for Vehicle that takes all parameters except year. It should by default set year equal to the current year (2010).

   See the screenshot below.

```
class Vehicle
{
  String make;
  String manufacturer;
  int year;
  int numberOfWheels;
  double engineSizeInLiters;

  Vehicle(String makeInput, String manufacturerInput, int yearInput,
    int numberOfWheelsInput, double engineSizeInput)
  {
    make = makeInput;
    manufacturer = manufacturerInput;
    year = yearInput;
    numberOfWheels = numberOfWheelsInput;
    engineSizeInLiters = engineSizeInput;
  }

  Vehicle(String makeInput, String manufacturerInput,
    int numberOfWheelsInput, double engineSizeInput)
  {
    make = makeInput;
    manufacturer = manufacturerInput;
    year = 2010;
    numberOfWheels = numberOfWheelsInput;
    engineSizeInLiters = engineSizeInput;
  }
}
```

2. Which of the following are examples of a real-world inheritance relationship? Assume your two classes are the words in capital letters.

   Library and Book
   NOT INHERITANCE A library contains lots of books
   Student and Course
   NOT INHERITANCE A student is enrolled in a course
   Politician and Governor
   INHERITANCE A governor is a specialized type of politician
   Currency and Euro
   INHERITANCE The Euro is a specialized typed of currency
   Ship and AircraftCarrier
   INHERITANCE An AircraftCarrier is a specialized type of Ship
   Ship and Fleet
   NOT INHERITANCE A ship is part of a fleet (group) of ships

3. Explain briefly (in a few sentences) the relationship between the functions available in a class and the functions available in a subclass of that class.

   All of the functions from the superclass are automatically inherited (available) in the superclass. They do not need to be rewritten.

   Often times there will be new functions written in the subclass (as those functions would be part of what makes the new subclass distinct enough to be different from its parent).

   It is possible to overwrite the parent's functions that were inherited so that they act differently.

   It is possible to refer to the parent's functions using the *super* syntax.

4. Given the following class, implement the *constructor* and *getDataAt* functions according to the descriptions written in comments (//) above each function.

```
class DataHolder
{
        float[ ] data;
        int numberOfItemsInArray;

        // create the data array based on the input, and save the input size
        // into the numberOfItemsInArray variable
        DataHolder( int size)
        {
                See screenshot below
        }

        // accessor to return the content of an array position
        // or -1 if the requested position is not valid for the array
        public double getDataAt(int position)
        {

                See screenshot below
        }
}
```

```
class DataHolder
{
  float[ ] data; // declare the array with name data, type float[]
  int numberOfItemsInArray;

  // create the data array based on the input, and save the input size
  // into the numberOfItemsInArray variable
  DataHolder(int size)
  {
    data = new float[size]; // create the array the right size
                            // it was already declared
    numberOfItemsInArray = size;
  }

  // accessor to return the content of an array position
  // or -1 if the requested position is not valid for the array
  public double getDataAt(int position)
  {
     // check out of bounds (a position less than 0, or past end of array)
    if ((position < 0) || (position >= data.length))
    {
      return -1;
    }
    else
    {
      // in bounds - so just return value
      return data[position];
    }
  }
}
```

5. Below you will find a recursive function that computes x^n (x to the power n). Assume the power function is called with $x = 2$ and $n = 4$. Extend and fill in the table started below showing each function call as it is made, and the returned value from each function call.

```
int power(int x, int n) {
        if (n == 0)      { return 1; }
        else if (n == 1) { return x; }
        else  {
         // we are using integers, so / is integer division and % is remainder
                int z = power(x,n/2);
                if ((n % 2) == 0) { return z * z; }
                else { return x * z * z; }
        }
}
```

Initial function call:	power(2,4)	Returned value: ____16________
Remaining function calls and values:	power(2,2)	Returned value:______4________
	power(2,1)	Returned value:______2________

See explanation on next page

Explanation: Note you start in the Power(2,4) block (the one on the bottom); whenever “Do recursion” is seen in a description, that adds another block on top (another use of the “Power” function – each use is independent)

Stack block	Description
Power(2,2) X = 2 N = 1	At base case (n == 0 o n == 1)? YES Return X back (2)
Power(2,2) X = 2 N = 2	At base case (n == 0 o n == 1)? NO Do recursion with x => 2, n = >2/2 => 1 Get 2 back from recursion, square it, and return value Returning a 4
Power(2,4) X = 2 N = 4	At base case (n == 0 o n == 1)? NO Do recursion with x => 2, n = >4/2 => 2 Get 4 back from recursion, square it, and return value Returning a 16

16 is the FINAL ANSWER (2^4)

6. Argue for or against this statement: The number of instructions used in the machine language representation of a program will be less than the number of instructions used in the high level language representation of the same program.

   This is a false statement. High level (Processing) instructions are “English-like”. Machine language instructions (what the computer processor actually runs) are very simple. It takes more than one simple instruction to be equivalent to one Processing instruction. So, after compilation, you should have many more (but simpler) machine language instructions than the number of Processing language instructions you started with.

7. Suppose you needed to model a die with a class.

   What are some variables associated with a die ?

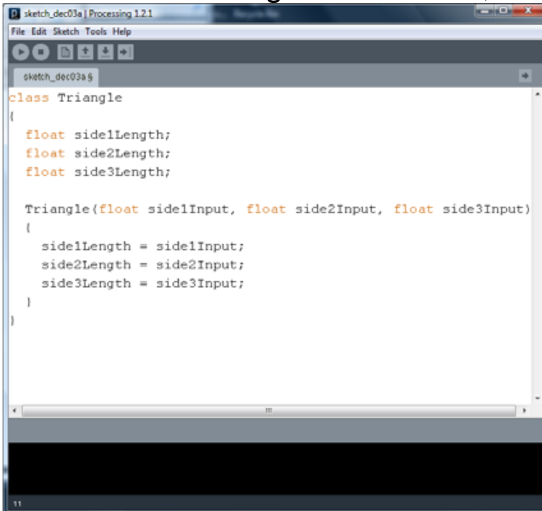
   int numberOfSides; // how many sides it has
   int currentSide; // which side is facing up
   String[ ] sideValues; // the actual values associated with each side

   What are the functions that might be made available for a die?

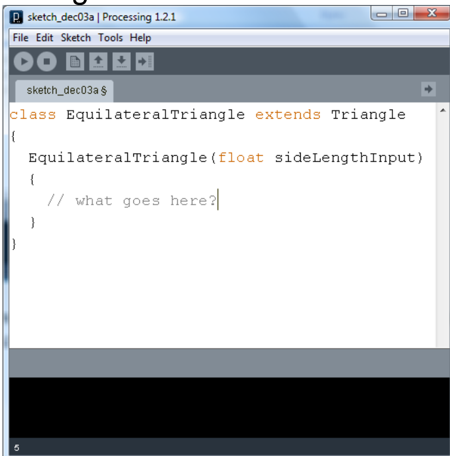
   void roll(); // choose a different side at random
   String getValue(); // figure out what the value that is up is
   The constructor -- Die(int numberOfSidesInput) --

8. [Inheritance/Classes] I argue that an EquilateralTriangle class is a specialization of a Triangle class, and thus appropriate to be a subclass of a Triangle class.

a. Given the Triangle class below,



what is the appropriate way to write the EquilateralTriangle constructor such that it employs the Triangle constructor?



super(sideLengthInput, sideLengthInput, sideLengthInput);

super is the reference to the superclass constructor. Basically we want the side length for the EquilateralTriangle (only one needs to be specified since all sides have to take on that value) to be given to the superclass, as it actually has the code which assigns values into each of the side1Length, side2Length, and side3Length variables (which EquilateralTriangle also inherits fr free).

b. Would I need to over-write the *getPerimeter()* method for EquilateralTriangle?

No, there is nothing special about computing the perimeter for an EquilateralTriangle that needs to be done differently than what is done for a regular triangle. A regular triangle probably uses (side1Length + side2Length + side3Length) which will work just fine for EquilateralTriangles. Note when we did over-write something (in CommissionedEmployee, it was because we had to change how the paychecks were being computed).

9. Assume you have a class representing a BibliographyEntry (a reference for a paper) similar to below:

```
class BibliographyEntry
{
        String authorName; // "William Turkett"
        String articleName; // "Motif Analysis of Network Intrusion Activity"
        String journalName;  // "Journal of Network Technology"
        int yearOfPublication; // 2008

        String getCitationString() // [Turkett2008]
        {

        }
}
```

The comments represent the values associated with each class variable, and the desired return value from *getCitationString().* You can assume the variables always take on such values (the author name will always be “firstName lastName”.

Explain what String functions you would use ,and why you would use them, in *getCitationString()* to generate the desired citation string (*lastNameYEAR*).

Use the *split* function (not technically a String class function, but often used on Strings) to break the authorName variable apart into the two Strings: “William” and “Turkett”. (Alternatively, you can use the String class functions *indexOf* to find the space between the names, and then *substring* to pull out Turkett, which is behind the space).

Then, use the + String concatenation operator (a String class function) to concatenate (merge together) the last name and the year variable.

10. Here’s a recursive function (f) that works with arrays, and a function (g) that “kick-starts” the f function.

```
int g(int[] array) { return f(array, 0); }

int f(int[] array, int position)  {
        if (position == (array.length-1))     {
                return array[position];
        }
        else  {
                int mTail = f(array, position+1);
                if (array[position] > mTail) return array[position];
                else return mTail;
        }
}
```

(continued on next page)

a. Suggest the purpose of the *f* function: It finds the maximum value in the array. It does this by saying – go find me the maximum of the tail of the array (the rest of the array behind where I am) (do this maximum finding recursively), and then I will compare the the current item I am sitting on to that number and return the larger of those two numbers. This larger value will be pushed backwards, so that even earlier positions in the array have something to compare against in finding the max. An example of this being used is:

   Array: [2 | 5 | 3 | 4]

   // do the recursion
   f([2,5,3,4]) ➔ hold onto 2, compare it to the result of f([5,3,4])
   f([5,3,4]) ➔ hold onto 5, compare it to the result of f([3,4])
   f([3,4]) ➔ hold onto 3, compare it to the result of f([4])
   f([4]) ➔ this is working on the last element of the original array, return a 4
   // here comes the recursion "unwinding" as the answers propagate back up
   f([3,4]) ➔ Take the returned 4, compare it to the held on 3, return the larger, which is 4
   f([5,3,4]) ➔Take the returned 4, compare it to the held onto 5, return the larger, which is 5
   f([2,5,3,4]) ➔ Take the returned 5, compare it to the held onto 2, return the larger, which is 5

   5 is the final answer (which is the largest value).

b. Explain what the base case is detecting that indicates to stop repeating: Am I on the last position in the array? The test is (position == array.length – 1) [array.length-1 is the last valid position number]