CSCI 136 Written Exam #0 **Name: _____**
Fundamentals of Computer Science II
Spring 2015

This exam consists of 6 problems on the following 7 pages.

You may use your single-sided handwritten 8 ½ x 11 note sheet during the exam. No calculators, computers, mobile devices, cell phones, or other electronic devices are permitted.

If you have a question, raise your hand and I will stop by. Since partial credit is possible, **please write legibly and show your work**.

Problem	Points	Score
1	8	
2	8	
3	10	
4	8	
5	16	
6	6	
Total	56	

1. Loops, Input (8 points). Consider the following program:

```
public class Prob1
{
   public static void main(String[] args)
   {
      final int N = Integer.parseInt(args[0]);
      int [] a = new int[N];

      for (int i = 1; i < args.length; i++)
      {
         int j = Integer.parseInt(args[i]);
         a[j] = 1;
      }

      for (int i : a)
      {
         System.out.print(i + " ");
      }
   }
}
```

Below are five example executions of the program. Give the output produced by the program. If the given input would cause a runtime exception, write "error".

Command line	**Output**
% java Prob1	error
% java Prob1 4	0 0 0 0
% java Prob1 5 0 2 4	1 0 1 0 1
% java Prob1 5 2.3	error
% java Prob1 6 2 2 3 2 3 2	0 0 1 1 0 0

2. File I/O, Collections (8 points).

a) Consider the following method that is supposed to print out each word contained in the filename passed to it. Currently it compile and runs, but it just prints the filename and not the words in the actual file.

```
00    public static void printWords(String filename)
01    {
02        Scanner scanner = null;
03
04        scanner = new Scanner(filename);
05
06        while (!scanner.hasNext());
07            System.out.println(scanner.next());
08
09        scanner.close();
10    }
```

Circle the ***four*** things that must be done to make this method work as intended.

A. Line 00: Remove the **static** keyword
B. Line 00: Change **void** to String
C. Line 02: Change **null** to filename.
D. Line 04: Change filename to **new** File(filename))
E. Surround line 04 with a **try-catch** block, catching an exception of type FileNotFoundException
F. Surround line 04 with a **try-catch** block, catching an exception of type ArrayNotFoundException
G. Line 06: Change hasNext() to next()
H. Line 06: Remove the ! symbol at the front of the condition
I. Line 06: Remove the ; symbol at the end of the line
J. Line 07: Change println to printf
K. Line 09: Change scanner to filename

b) Add the missing code to the following method so that all the values in the nums list are output to the specified filename. Each number should be on a ***separate line*** and ***rounded to two decimal places***.

```
public static void outputNums(String filename, ArrayList<Double> nums)
{
   PrintWriter writer = null;
   try
   {
      writer = new PrintWriter(filename);
   }
   catch (FileNotFoundException e)
   {
      return;
   }

   for (double d: nums)
      writer.printf("%.2f\n", d);
   writer.close();

 }
```

3. Multiple choice (10 points). Circle the single ***best*** answer:

a) You want to declare and instantiate a variable that can be used to store both the name of a color, e.g. "BrickRed" and a Java `Color` object that represents that color. The variable should be able to grow automatically as colors are added. Which of the following compiles and meets your requirements?

```
 I.  ArrayList<Color>          colors = new ArrayList<Color>();
 II. ArrayList<String, Color>  colors = new ArrayList<String, Color>();
 III. String []                colors = new String[Color.length()];
 IV. HashMap<String, Color>    colors = new HashMap<String, Color>();
 V.  HashMap<Color, String>    colors = new HashMap();
```

b) You are declaring a concrete class `Employee` that is a child class of the abstract base class `Person`. You would like to make it so a collection of Employee objects can easily be sorted (say by salary) using the Java static methods `Collections.`*`sort`* or `Arrays.`*`sort`*. Which of the following class declarations meets your requirements?

```
 I.   public class Employee extends Person implements Comparable<Employee>
 II.  public class Employee extends abstract Person implements Comparable<Integer>
 III. public class Employee implements Person extends Comparable<Employee>
 IV.  public class Employee extends Person, Comparable<Employee>
 V.   public class Employee extends Person<Integer>
```

c) All the following statements about Java exceptions are true ***EXCEPT***:

I. If a method declares that it throws a particular type of checked exception, exceptions of that type in the method do not need to be enclosed in a try-catch block.
II. Multiple catch blocks are possible, each catching a different type of exception.
III. Including a try-catch block for unchecked exceptions such as `NullPointerException` is optional.
IV. Program execution always terminates after the last line of a catch block.
V. Opening a file for reading can cause a checked exception of type `FileNotFoundException`.

d) A class has the following method:

```
public double foo(int a, double b)     { return 0.0; }
```

All the following methods would be valid overloaded versions of the method `foo` ***EXCEPT***:

```
 I.   public    int foo(int a, int b)        { return 0;   }
 II.  public    int foo(int a, double b)     { return 0;   }
 III. public double foo(String a, int b)     { return 0.0; }
 IV.  public double foo(double a, int b)     { return 0.0; }
 V.   public double foo(int a, int b, int c) { return 0.0; }
```

e) Which of the following is a boolean expression that is a valid and reliable way to test if the `String` variable `str` (which you can assume is non-null) is equal to `"Duck"`:

```
 I.   (str == "Duck")
 II.  (str != "Duck")
 III. ("Duck" = str)
 IV.  (str.equals(Duck))
 V.   (str.equals("Duck"))
```

4. Short answer (8 points).

a) Recall in the predictive keyboard assignment, we had an instance variable declared as follows:

```
private HashMap<String, Integer> wordToCount = new HashMap<String, Integer>();
```

Consider if we had instead used two parallel `ArrayList` variables to track the words and their counts:

```
private ArrayList<String>  words  = new ArrayList<String>();
private ArrayList<Integer> counts = new ArrayList<Integer>();
```

Briefly describe the performance advantage of the `HashMap` version compared to the `ArrayList` version.

The HashMap version can find the count for a given word in constant time. The ArrayList version would need to search through the list in order to find the word, thus taking time linear in the number of words.

b) Briefly describe the fundamental difference between a concrete class and an abstract class in Java. Be sure to describe what you can do with one type of class but not the other.

A concrete class has all of its methods implemented and none are declared with the abstract keyword. An abstract class may have methods declared abstract and these methods defer implementation to its children. You can instantiate (using the new operator) an object of a concrete class type, but you cannot instantiate an object of an abstract class type.

c) Briefly describe the fundamental difference between an abstract class and an interface in Java.

An abstract class can contain implemented methods and/or instance variables. Implemented methods and instance variables are not allowed in an interface (with the exception of default methods added in Java 1.8 and public static final instance variables). Interfaces are simply a list of abstract methods that all classes that implement the interface must provide implementations of.

d) Briefly describe two useful things we have seen interfaces used for in Java:

Sorting objects of a certain data type by declaring the data type implements Comparable.

Iterating over all the objects in a collection by declaring the collection data type implements Iterable.

5. **Inheritance** (16 points)

a) Consider the following class:

```
public abstract class Person
{
    private String name = "";

    public Person(String name)
    {
        this.name = name;
    }

    public String toString()
    {
        return name;
    }
}
```

Fill in the missing **super** and **this** keywords into the underlined areas to correctly implement this child class:

```
public class Employee extends Person
{
    private int salary = 0;
    private boolean retired = false;

    public Employee(String name, int salary)
    {

        _____super_____(name);


        _____this_____.salary = salary;
    }

    public Employee(String name, int salary, boolean retired)
    {

        _____this____(name, salary);


        _____this____.retired = retired;
    }

    // Return a string representation of this person, for example: John 23000 true
    public String toString()
    {

        return String.format("%s %d %b", ____super___.toString(), salary, retired);
    }
}
```

b) Given the Person and Employee classes from part a, fill in the missing code in the class declaration and instance variables for the class Student. Student is-a Person, but has additional state such as a student ID and grade point average (GPA). Be sure to demonstrate proper ***data encapsulation***.

You also need to complete the method isHonorRoll which returns true if a student's GPA is ***greater than or equal to 3.0***, false otherwise. For full credit, do this using a ***single statement*** in isHonorRoll.

```
public class Student extends Person
{
    private long id = 0;
    private double gpa = 0.0;

    public Student(String name, long id, double gpa) { /* code */ }

    public boolean isHonorRoll()
    {
        return (gpa >= 3.0);
    }
}
```

c) Provide a code snippet that declares an instantiates an array named people that contains an Employee named "Bob" and a Student named "Carol". You can choose any valid values for the other parameters required to construct the Employee and Student objects.

```
Person[] people = new Person[2];
people[0] = new Employee("Bob", 100);
people[1] = new Student("Carol", 123, 4.0);
```

d) Given the Person, Employee, and Student classes declared previously, give just the class declaration (no need for instance variables or methods) for a class StudentEmployee that is-a Student and is-a Employee (that is it inherits from both). If this is not possible, briefly explain why.

Not possible, Java does not support multiple inheritance (can't extend more than one class).

6. Methods (6 points). Match the method description on the left with the ***best*** method on the right. Not every letter will be used. Each letter will be used ***at most once***.

__J__ A method that given a two-dimensional array of integers and the index of a row in that array, returns the specified row of values.

__E__ A method that given a two-dimensional array of integers and the low and high value of an integer range, returns the percentage of integers in the array that are inside the specified range.

__L__ A method that given the size of each dimension, creates and returns a three-dimensional array of integers, initializing all elements to 1.

__D__ A method that given two sets of integers, returns all the integer values that are in common between the sets.

__A__ A method that given a list of integers, converts them to a list of the spelled version of each integer (e.g. converting integer 42 to "forty two").

__C__ A method that given two sets of integers, returns true if the two sets are exactly the same size, false otherwise.

A.
```
public static String [] foo(ArrayList<Integer> a)
```

B.
```
public static boolean foo(String a)
```

C.
```
public static boolean foo(HashSet<Integer> a,
                          HashSet<Integer> b)
```

D.
```
public static int [] foo(HashSet<Integer> a,
                         HashSet<Integer> b)
```

E.
```
public static double foo(int [][] a,
                         int b,
                         int c)
```

F.
```
public static int [] foo(String [] a)
```

G.
```
public static int foo(ArrayList<String> a)
```

H.
```
public static int foo(HashSet<String> a)
```

I.
```
public static ArrayList<Boolean> foo()
```

J.
```
public static int [] foo(int [][] a,
                         int b)
```

K.
```
public static String [] foo()
```

L.
```
public static int [][][] foo(int a,
                             int b,
                             int c)
```

M.
```
public static int[3] foo(int a,
                         int b,
                         int c)
```