Code-Writing Quiz 3 Prep

The goal of code-writing quizzes is to help students gain fluency and confidence in their code-writing.

During the quiz students will be given up to 3 very short problems and about 15 min. to implement the solution with a pencil/pen on a piece of paper. Then each student will be randomly paired with a peer and the peers will assess each-other's work. All the errors must be marked with a red pen and the quiz will be graded according to a simple rubric. The rubric will be provided. Instructor will collect all the quizzes to quickly check on the works and to post grades in the gradebook.

Students are expected to write syntactically and logically correct Java code. Import statements and main() method heading will not be needed though.

The quiz is closed-book, no-computer-access type of exercise. Students are encouraged to prepare notes on a 3x5 inch index card using both sides. That card is the only reference students can use during the quiz and eventually during the midterm and final exams. It is necessary to spend time practicing for the quiz. It's close to impossible to do well on such a quiz without prior practice. Problems provided below are very similar to the one(s) that will be appearing on the quiz.

If the student missed the quiz and wants to get credit for it, he/she must submit to the instructor hand-written code solutions to all the sample problems listed in the quiz preparation guide.

Topics to Review

- Recursion – replacing simple loop with recursive method calls
- Recursion – using recursive function in implementing recursive method (power, factorial, etc.)
- Recursion – reading recursive code and converting it into iterative
- Reading from binary file
- Writing to binary file

Sample Problems

1. Write recursive method int sum(int k) that calculates the sum of a series: 1+2+3+..+k. Make the method throw an IllegalArgumentException when k <0.
2. Write a recursive method that calculates factorial(int k).
3. Write a recursive method that calculates a power: double pow(int base, int exp)
4. Write a recursive method that prints out a numeric sequence.
   printSequence(4) prints 1 2 3 4 3 2 1
   printSequence(5) prints 1 2 3 4 5 4 3 2 1

5. Write a recursive method writeChars that accepts an integer parameter n and that prints out n characters as follows. The middle character of the output should always be an asterisk ("*"). If you are asked to write out an even number of characters, then there will be two asterisks in the middle ("**"). Before the asterisk(s) you should write out less-than characters ("<"). After the

asterisk(s) you should write out greater-than characters (">"). For example, the following calls produce the following output:

Call	Output
`writeChars(1);`	`*`
`writeChars(2);`	`**`
`writeChars(3);`	`<*>`
`writeChars(4);`	`<**>`
`writeChars(5);`	`<<*>>`
`writeChars(6);`	`<<**>>`
`writeChars(7);`	`<<<*>>>`
`writeChars(8);`	`<<<**>>>`

6. Write a recursive method `isReverse` that accepts two strings as a parameter and returns `true` if the two strings contain the same sequence of characters as each other but in the opposite order (ignoring capitalization), and `false` otherwise. For example, the string `"hello"` backwards is `"olleh"`, so a call of `isReverse("hello", "olleh")` would return true. Since the method is case-insensitive, you would also get a `true` result from a call of `isReverse("Hello", "oLLEh")`. The empty string, as well as any one-letter string, is considered to be its own reverse. The string could contain characters other than letters, such as numbers, spaces, or other punctuation; you should treat these like any other character. The key aspect is that the first string has the same sequence of characters as the second string, but in the opposite order, ignoring case. The table below shows more examples:

Call	Value Returned
`isReverse("CSE143", "341esc")`	`true`
`isReverse("Madam", "MaDAm")`	`true`
`isReverse("Q", "Q")`	`true`
`isReverse("", "")`	`true`
`isReverse("e via n", "N aIv E")`	`true`
`isReverse("Go! Go", "OG !OG")`	`true`
`isReverse("Obama", "McCain")`	`false`
`isReverse("banana", "nanaba")`	`false`
`isReverse("hello!!", "olleh")`	`false`
`isReverse("", "x")`	`false`
`isReverse("madam I", "i m adam")`	`false`

`isReverse("ok", "oko")`	`false`

You may assume that the strings passed are not `null`. You are not allowed to construct any structured objects other than `Strings` (no array, `List`, `Scanner`, etc.) and you may not use any loops to solve this problem; you must use recursion. If you like, you may declare other methods to help you solve this problem, subject to the previous rules.

7. Given a recursive method. Write an iterative method with the same functionality.

```
void writeVertical(int n)
{
   if (n < 10)

        System.out.println(n);
   else
   {

        writeVertical(n/10);
        System.ou.println(n%10);
   }
}
```

8. Given a class definition:

```
public class Rectangle {

private double length;

private double width;

public Rectangle(double len , double wid) {/*implemented*/}

public Rectangle(){/*implemented*/}

/* other methods implemented*/

}
```

1. What statements/methods must be added to the class in order to make it possible to write objects of that class into a file in binary mode
2. Create an array of 100 objects of type Rectangle. Generate 100 rectangles with random widths in range [2, 20] and random lengths in range [3, 30]. Open a file named “data.dat” for writing in binary mode and write all objects into the file. If the file does not exist it must be created. All IOExceptions must be handled properly.
3. Assume you have a file called “data01.dat” filled with objects of type Rectangle stored there in binary mode. Open the file for reading, read all the objects from there and store them in ArrayList. Assume we don’t know how many objects are stored in file. All IOExceptions must be handled properly.

I assume you know all topics from previous Quizzes and remember how to solve problems listed below.

Quiz 2 Topics

- Inheritance – how to make a class that inherits from another lass.
- Protected class fields
- Writing a constructor for derived class (calling constructor of a base class first0
- Writing clone() method using the "black magic spells" of Java we discussed in class
- Writing @Override public boolean equals(Object obj) for any class.
- Writing an abstract class and abstract method(s)
- Writing an interface
- Implementing an interface

Sample Problems

Given a class described below:

```
public class Rectangle implements Cloneable{

   private double length;

   private double width;

   public Rectangle(double len , double wid) {/*implemented*/}

   public Rectangle(){/*implemented*/}

   public Rectangle clone(){/*for you to implement*/}

   @Override boolean equals(Object obj) {/*for you to implement*/}

   public String toString(){/*implemented*/}

   /* assume accessor and mutator methods implemented*/

}
```

4. Implement clone() method for Rectangle class
5. Implement equals() method for Rectangle class
6. Write a definition of a class MyRectangle that inherits from Rectangle and has a field of type int[] myArray. Class must also implement Cloneable and Relatable interfaces.
7. Write constructor *public MyRectangle(int len, int wid, int[] array*) that sets all the fields of the class. Make sure to
   a. call the constructor of the base class first

    b. make a deep copy of an array given to you.
8. Implement clone() for MyRectangle. Make sure to do it according to the “black magic recipe” we discussed in class.
9. Implement `@Override boolean equals(Object obj)` for MyRectangle
10. Implement *toString()* method for MyRectangle class
11. Use Rectangel and MyRectangle classes to illustrate the concept of polymorphism in OOP.
12. Write a definition of MyAbstract class that has a field of type int and an abstract public method *int doStuff()*
13. Write a definition of a MidTermable interface that has the following methods: *boolean readTextBook(); boolean practiceCodeWriting();*
14. Add a field to the interface int N. Make sure the field is defined as a CONSTANT

Quiz 01 Topics

- Writing class constructors – no-argument constructors and constructors that take parameter
- Writing class definitions for classes with fields of <u>non-primitive type</u>
- Accessor and mutator methods
- Accessor method for reference field (not of primitive type)
- Mutator methods that throw exceptions
- Writing copy constructor that creates deep copy of non-primitive fields
- Writing equals() method
- Writing toString() method
- STATIC methods (make sure to know the limitations here!)
- STATIC fields

Sample Problems

Given a class described below:

```
public class Rectangle {

   private double length;

   private double width;

   public Rectangle(double len , double wid) {/*implemented*/}

   public Rectangle(){/*implemented*/}

   public clone() {/*implemented*/}

   @Override boolean equals(Rectangle) {/*implemented*/}

   public String toString(){/*implemented*/}
```

```
    /* accessor and mutator methods implemented*/
}
```

Write a class MyClass that has 2 fields: Rectangle r and int aField. Implement the following methods / fields to the new class:

1. Accessor for field r. Make a deep copy of the field when returning it.
2. Constructor MyClass(int a, Rectangle b) that sets both fields. Make a deep copy of object b when copying it!
3. No-argument constructor that sets aField to 0, and r field to a rectangle object with length and width set to 1.
4. Mutator for aField. The value stored in that field must not be negative. Throw Illegal ArgumentException when the negative value is detected.
5. copy constructor for MyClass
6. toString() method for MyClass
7. equals() method for MyClass
8. A static field to MyClass that counts how many objects of that class were created. Make adjustments to the constructors as needed to support the object counting feature of MyClass.
9. A static accessor method to return the static field

Write a class MyClass that has 2 fields: Rectangle r and int aField. Implement the following methods / fields to the new class:

10. Accessor for field r. Make a deep copy of the field when returning it.
11. Constructor MyClass(int a, Rectangle b) that sets both fields. Make a deep copy of object b when copying it!
12. No-argument constructor that sets aField to 0, and r field to a rectangle object with length and width set to 1.
13. Mutator for aField. The value stored in that field must not be negative. Throw Illegal ArgumentException when the negative value is detected.
14. copy constructor for MyClass
15. toString() method for MyClass
16. equals() method for MyClass
17. A static field to MyClass that counts how many objects of that class were created. Make adjustments to the constructors as needed to support the object counting feature of MyClass.
18. A static accessor method to return the static field