

CMSC 131: Fall 2004

Final Exam Practice Questions

Disclaimer: The following are questions that try to provide you with some practice material for the final exam. By no means do they represent the only material you should know for the final exam. We will not be providing answers to this set of questions however, feel free to see the instructors or TAs for assistance during office hours. Some of the questions can be considered challenging.

General Questions

1. What is the difference between overriding and overloading?
2. Can every class in Java have a main method?
3. Can you have a private class?
4. What is the difference between the following two declarations?

```
final static int SIZE = 10;  
  
final int SIZE = 10;
```
5. What is the difference between using composition vs. using inheritance?
6. What is the difference between a class and an instance of a class?
7. What are the advantages/disadvantages of the StringBuffer class vs. String?
8. What are the advantages/disadvantages of ArrayList class vs. Java arrays?
9. What is the value of x after executing the following statement?

```
int x = (1 & 0) | 1;
```
10. What is the value of z after executing the following statements?

```
int y = 4;  
int z = ( y < 10 ? (y > 2 ? 1 : 2 ) : 3 );
```

11. What is output by the following code fragment?

```
try {
    String x = null;
    System.out.println( x.length() );
}
catch ( NullPointerException e ) {
    System.out.println( "Null Exception" );
}
catch ( Exception e ) {
    System.out.println( "Exception" );
}
System.out.println( "Why are we here?" );
```

Code Analysis

1. Using the TableSeatingChart class (available online in the Code Snippets section) draw a memory map that represents the contents of the heap and stack up to the point in the main program where "***STOP**" appears.
2. You be the compiler. All these classes appear in the same package. For each statement of the main program, indicate whether it generates a compilation error, and if not, what is printed. (When showing the execution, assume that all statements generating compilation errors have been removed.)

```
// in file Foo.java
public class Foo {
    protected void pro( int x ) { System.out.println("Foo:pro:" + x); }
    public void pub( int x ) { System.out.println("Foo:pub:" + x); }
}

// in file Bar.java
public class Bar extends Foo {
    protected void pro( int x ) { System.out.println("Bar:pro:" + x); }
    public void pub( int x ) { System.out.println("Bar:pub:int:" + x); }
    public void pub( double d ) { System.out.println("Bar:pub:dbl:" + d); }
}

// in file Driver.java
public static void main(String[] args) {
    Foo a = new Foo( );
    Foo b = new Bar( );
    Bar c = new Bar( );
    a.pro( 1 );
    a.pub( 2 );
    a.pub( 3.0 );
    b.pro( 4 );
    b.pub( 5 );
    b.pub( 6.0 );
    c.pro( 7 );
    c.pub( 8 );
    c.pub( 9.0 );
}
```

3. How would your answer to the above problem differ if Foo and Bar had been defined in a different package than Driver?

Implementation

1. Write a static method that produces an identical (deep) copy of a two-dimensional array of doubles. The signature of the method is:

```
public static double[][] duplicate(double[][] array);
```

2. Write a static method that is given a 2-dimensional array of doubles, and finds the first row that consists of an increasing sequence of values, that is, it finds the smallest i such that:

```
array[i][0] < array[i][1] < array[i][2] < ...
```

If such a row exists, a copy of the contents of that row are returned as a one dimensional array. Otherwise null is returned. The signature of the method is:

```
public static double[] firstIncreasing(double[][] array);
```

3. Implement a boolean method theSame() that determines whether two two-dimensional arrays of String objects have the same objects. Your solution should be efficient in the sense that as soon as it determines that two elements differ, no further comparisons should take place.
4. Write a method that is given a two-dimensional (ragged) array of String objects and returns a two-dimensional (ragged) array of String objects where all the null entries have been removed. For example, if the original array has the data (NULL represents a null reference):

```
John    (null)  Mary  George (null)
(null)  Pete   Rick
(null)  (null) (null)
```

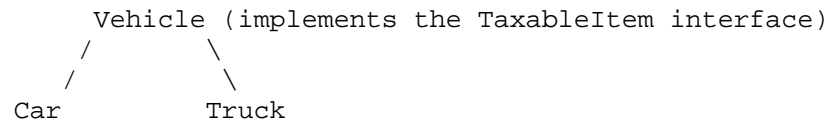
the result generated by your method will be a two-dimensional array with three rows.

```
John  Mary  George
Pete  Rick
(this row is empty)
```

5. Implement the classes and interface associated with the inheritance hierarchy below. We are providing the basic idea but it is up to you to develop the full set of classes/interfaces. This problem is intentionally open-ended, and there is no one correct solution. Try as much as possible to have the following constructs in your code:

CONSTRUCTS: super, this, protected, overriding, overloading

Inheritance Hierarchy:



Vehicle Class:

data members: weight, maximumSpeed, numberOfDoors,
numberOfTires, color
methods: get (accessor) methods for data members
abstract methods: start(), move(), shiftGear()

Car Class:

data members: add any members you understand are necessary
methods: get/set methods for data members, isSedan(), isSportCar(),

Truck Class:

data members: maxLoadCapacity, maxSpeedLoaded
methods: get/set methods for data members,
unloadCargo(), loadCargo()

TaxableItem Interface:

methods: getTaxAmount(), getItemName()

Feel free to add any data members/methods you understand are necessary.

6. Write a method `scramble(String s)` which given a String `s`, consisting of words separated by spaces, and scrambles (permutes) the words in random order. For example, given the string:

"Now is the time for all good men to come to the aid of the party"

it might produce the scrambled string:

"time good of for to all aid the is men come the to party Now the"

7. Develop a program that implements a cookbook. We are providing a general description of the classes associated with this design. Feel free to add any methods/data members you understand are necessary. Try to practice the following concepts as you develop the implementation:

One-dimensional array of objects
get/set methods for classes
private vs. public access specifiers
Passing primitives and objects as parameters

I. Recipe class - represents a cooking recipe.

data members:

name - name associated with the recipe (e.g. "PlainCheeseCake")

instructions -

an array of String objects where each entry represents one instruction associated with the recipe.

numberOfInstructions -

integer representing the number of instructions

ingredients -

an array of String objects where each entry represents an ingredient

numberOfIngredients -

integer representing the number of ingredients

methods:

appropriate constructor

get/set methods you understand are necessary

toString() method

equals method

II. Cookbook - represents the collection of recipes

data members:

listOfRecipes - an array of Recipe objects

numberOfRecipes - an integer representing the number of recipes

methods:

appropriate constructor

get/set methods you understand are necessary

addRecipe -

adds the recipe by creating a new array that includes the current number of recipes plus the new one to add. You must keep the recipes ordered by recipe name.

findRecipe -

locates a recipe based on a recipe name.

removeRecipe -

deletes a recipe from the cookbook. It requires creating a

new array with all the original recipes except the one to delete.

toString - implement the appropriate toString method

main -

write a main method that tests your class and show how your methods work.

9. Reimplement the Cookbook class using classes from the Java Library that we studied in class. In particular, there are classes that could help you manipulate the array of recipes.