

CS125 Fall 2009 Exam 3 Practice Exam - Key

Note: There may be more than one valid solution for some of the problems presented – the answers provided in this key are not necessarily the only correct answers.

Before you start, please note: it is possible to get strong clues for answering some of the questions in the review by referring to other questions. For example, converting from code to a flow chart and vice versa. In general, you should not depend on both versions of a question showing up in the exam.

Also note: this study guide is not comprehensive. I strongly suggest reviewing: the code examples composed in class, homework assignments, material from the last exam & the self check questions in the text.

1. Write & document a public method called **reverseArray** that takes an array of Strings as a parameter and reverses the contents such that the first item becomes the last item.

// code goes here

```
/**
 * takes an array of Strings & builds a new array with the same contents in
 * reverse order
 * @param data    the array to reverse
 * @return       a reverse order copy of data
 */
public String[] reverseArray(String[] data)
{
    String[] revData = new String[data.length];
    int index = data.length - 1;
    for(int a=0; a<revData.length; a++)
    {
        revData[a] = data[index];
        index--;
    }
    return revData;
}
```

2. Write & document a public method called **getBiggest** that takes an **ArrayList** of **Doubles** as a parameter and find the largest value. If the ArrayList is empty, print some sort of error message & return zero.

// code goes here

```
/**
 * finds the biggest item in the given list - if list is empty, prints error &
 * returns zero
 * @param list the ArrayList to search
 * @return the largest element if there is one, otherwise returns zero
 */
public double getBiggest(ArrayList<Double> list)
{
    if(list.size() <= 0)
    {
        System.out.println("error - empty list!");
        return 0.0;
    }
    double largestSoFar = list.get(0);
    for(Double d : list)
    {
        if(d > largestSoFar)
            largestSoFar = d;
    }
    return largestSoFar;
}
```

3. Examine the code below & explain the difference between **copy1** & **copy2**. Provide an alternative that achieves the same result as the third line (that is, provide code that creates a copy which would have the same properties as **copy2**).

```
int[] data = new int[] {5, 10, 15, 20};
int[] copy1 = data;
int[] copy2 = data.clone();
```

copy1 is not a truly independent copy - any subsequent changes to data will result in identical changes in copy1 & vice versa. Put another way, copy1 is an alias for data.

copy2 is what is a truly independent copy - any subsequent changes to data will not affect copy2, nor will changes to copy2 have any direct impact on data.

One way to achieve the same result:

```
int[] copy3 = new int[data.length];
System.arraycopy(data, 0, copy3, 0, data.length);
```

Another way would be:

```
int[] copy4 = new int[data.length];
for(int a=0; a<data.length; a++)
    copy4[a] = data[a];
```

4. Examine the code below. Add the necessary lines of code such that each new Player object will get its own unique idNum. Note: this should not be done by changing any of the existing code.

```
public class Player
{
```

```
    // instance fields
    int score;
    int idNum;
```

```
private static int nextIdNum = 0;
```

```
/**
 * constructor - sets idNum
 */
public Player()
{
```

```
    idNum = nextIdNum;
    nextIdNum++;
```

```
}
```

```
/**
 * returns idNum
 *@return the objects idNum
 */
public int getIdNum()
{
    return idNum;
}
```

```
}
```

5. Write a **for each** loop that sums up the contents for an **ArrayList** called **values**, stores the result in double called **sum** & then prints it.

// code goes here

```
double sum = 0.0;

for(Double d : values)
    sum += d;
System.out.println("the sum is: " + sum);
```

6. Write a **for** loop that finds the smallest value in an array called **values**, and swaps it to the last position of the array

// code goes here

```
int indexOfSmallest = 0;

for(int a=1; a<values.length; a++)
{
    if(values[a] < values[indexOfSmallest])
        indexOfSmallest = a;
}

int lastIndex = values.length - 1;
int temp = values[lastIndex];
values[lastIndex] = values[indexOfSmallest];
values[indexOfSmallest] = temp;
```

7. Write a **while** loop that examines all the values in an integer array called **values**, finds the count of the values that are even & then prints it.

// code goes here

```
int evenCount = 0;
int a=0;
while(a < values.length)
{
    if(values[a] % 2 == 0)
        evenCount++;
    a++;
}

System.out.println("count of even # is: " + evenCount);
```

8. Define encapsulation.

Encapsulation is the process of hiding implementation details & has two main purposes:

1) it keeps the implementation safe from undesired modifications

E.G. we can't accidentally modify the Java Random class

2) it reduces the complexity presented to the person using the code

E.G. we don't need to worry about exactly how Random generates random numbers

9. Define cohesion.

Cohesion refers to the degree to which a body of code addresses a single problem. Put another way, it indicates the degree to which code "goes together." If a class addresses a single clear abstraction, its code is highly cohesive. If instead, a class has a lot of different code addressing a lot of on related ideas, the code has low cohesion.

10. What is a postcondition? Give an example.

A postcondition is a condition which is promised to be true when a method finishes. It is the responsibility of the programmer writing the method to ensure this happens (assuming the preconditions, if any, were true when the method was called).

A postcondition for the spray() method of the RoachPopulation assignment is that the number of roaches has been reduced by 10%.

11. What does shadowing mean? Give an example.

Shadowing refers to a situation in which a given variable (typically an instance field) gets hidden by another, more local variable with the same name. The problem with this is that the local variable name takes precedence & changes intended for the original variable happen to the local variable instead.

In the example below, shadowing in the constructor prevents the instance field from being initialized.

```
public class Fruitsnack
{
    String flavor;
    public void Fruitsnack()
    {
        String flavor = "blue";
    }
}
```

12. Write & document a class called **ArrayChecker** with a static method called **mostFrequentItem**. It should take an array of integers & return the item that occurs the most often. If more than one item is 'tied for first place,' return the one that is encountered first in the array. Including a precondition comment indicating the method does not take empty or null arrays. Note: this problem may be easier if you split up the work into to more than one method. You might also want to check our dice game code.

Examples:

the mostFrequentItem of {1,2,3,1,4,5,1,6,7} would be 1

the mostFrequentItem of {1,3,2,4,4,3,4,6,3} would be 3

```
/**
 * provides methods for analyzing arrays
 */
public class ArrayChecker
{
    /**
     * finds the count of the occurrences of value in the array
     * @param array      the array to examine
     * @param value      the value to check for
     * @return            the occurrences of value in the array
     */
    public static int getCount(int[] array, int value)
    {
        int count = 0;
        for(int v : array)
        {
            if(v == value)
                count++;
        }
        return count;
    }

    /**
     * finds the count of the occurrences of value in the array
     * @param array      the array to examine
     * @return            the occurrences of value in the array
     * @precondition     array.length > 0
     */
    public static int mostFrequentItem(int[] array)
    {
        int highestCountSoFar = 0;
        int mostFrequent = array[0];

        for(int v : array)
        {
            int currentCount = getCount(array, v);
            if(currentCount > highestCountSoFar)
                mostFrequent = v;
        }
        return mostFrequent;
    }
}
```