

About this sample exam:

- You may and SHOULD work through this exam yourself and discuss your answers with others.
- To find out the “right” answers to questions, type in the code to a compiler and run it.
- You may post on the webboard questions about this sample exam
- You may post on PeerWise questions from this sample exam
- It is not indicative of length of the actual exam
- Not all topics that might be on the exam are covered (anything on a learning goal is fair game)
- **You should study your PSAs well** (as well as review the quizzes) – we assume that you can write code “like” your PSAs quickly and easily on paper.
- You will be required to write code by hand, we will provide the String and ArrayList method tables from Javadocs pages.
- There will be more multiple choice on the exam (e.g. where this has fill in the answer, we’ll likely have multiple choice).
- There may be code reordering problems on the exam (e.g. instead of writing code, you may be given a set of code statements or fragments and asked to select the right ones and place them in order).

Want more questions to practice??? Look on PeerWise!!!

Come up with some great questions when studying? Post them on PeerWise!!! You *may* get bonus points.

THIS CLASS IS NOT GRADED ON A CURVE. IT WILL NOT HARM YOU TO POST QUESTIONS ON PEERWISE!

Learning Goals

(these represent those that we've "covered" in class and you are responsible for on the midterm)

- LG1: Describe the effect of Java's "call-by-value" parameter passing mechanism.
- LG2: Trace code to recognize that primitive type variables (ints, doubles, etc.) that are passed as parameters to methods and modified locally in those methods DO NOT have their values changed in the calling method scope.
 - Since the value is copied into the local method (but not BACK to the calling method).
- LG3: Trace code to recognize that reference type variables (including objects and arrays of anything) that are passed as parameters to methods and modified locally in those methods DO have their values changed in the calling method scope.
 - Since the REFERENCE to the object/array is copied into the local method and anything that changes the data REFERENCED by that object/array in the local method is modifying the SAME COPY as is in scope in the calling method.
- LG4: Trace code which news a new object "overwriting" the reference to an object passed as a parameter (with no effect in the calling method scope since the reference is not copied BACK to the calling method).
- LG5: Use common data protection conventions when constructing a new object that has an array instance variable (create a copy of the array for the object, not just a reference to it)
- LG6: Explain why we'd create and use a static class variable.
- LG7: Be able to find information (including Javadocs for Java 6) on the Wrapper classes so you can use them in your PSA.
- LG8: Draw a unix directory structure diagram employing the terms: root directory, file suffix, root, leaf, parent, child.
- LG9: Read and write code that uses BufferedReader to read text from a file and catch exceptions appropriately with a simple try and catch block.
- LG10: Write code to iteratively read through a file until the end of file is reached (or some other sentinel value)
 - Sentinel value (not defined in book) is a value that means "STOP HERE"
- LG11: Read and write code using BufferedWriter to write to a file and catch exceptions appropriately.
- LG12: Be able to use Javadocs to determine whether a specific method call needs to be included in a try block, because it throws an exception.
- LG13: Read, understand and use String methods to solve problems and manipulate Strings.
- LG14: Compare dynamically resizable ArrayLists to standard Java arrays.
- LG15: Write code that correctly puts Object type data into an ArrayList and gets it back out (with appropriate casting).
- LG16: Utilize "live code experimenting" techniques to teach yourself about new programming language features.
- LG17: Apply "always deliverable" methods to your programming assignments.
- LG18: Read, trace, and write code that uses the basic ArrayList methods of add, add at index, get from index, set at index, size, and remove at index
- LG19: Explain the difference between size and capacity and know how this changes when various ArrayList methods are called
- LG20: Make an argument for why the ArrayList class has certain instance variables
- (for some reason we now jump to 28)
- LG28: Be able to describe how you have implemented always deliverable methodology in your PSA assignments
- LG29: Be able to list and describe some of the "parts" to the software development process
- LG30: Explain and be able to implement one of two common code structures used to search for elements in a list.
- LG31: Be able to use the compareTo method for Objects
- LG32: Identify the indexing pattern required to move elements "up" (for add) or "down" (for remove) in an array.

This problem uses the class Species - defined on the next page.
What is printed by each (marked with bold comments) print statement?

```
public class SpeciesTester
{
    public static void main(String []args)
    {
        int aaa; int bbb;
        Species s1,s2,s3;

        s1 = new Species("Rat",300,5);

        aaa = 1; bbb = 2;
        SpeciesTester.one(aaa,bbb);
        System.out.println("aaa: "+aaa+" bbb: "+bbb);    //1st PRINT

        aaa = 1; bbb = 2;
        aaa = SpeciesTester.two(bbb,s1,aaa);
        System.out.println("aaa: "+aaa+" bbb: "+bbb);    //2nd PRINT
        System.out.println(s1);                        //3rd PRINT

        s2 = new Species("Snake", 200, 2);
        s3 = new Species("Snake", 200, 2);
        s2 = SpeciesTester.three(s3,s2);
        System.out.println(s2);                        //4th PRINT
        System.out.println(s3);                        //5th PRINT
    }

    public static void one(int aaa, int bbb)
    {
        aaa+=22;
        bbb-=11;
    }

    public static int two(int aaa, Species s, int bbb)
    {
        s.setName("Whale");
        aaa+=22;
        bbb-=11;
        return bbb;
    }

    public static Species three(Species aaa, Species bbb)
    {
        bbb = new Species("Skunk", 40, 3);
        bbb.setName("Kitten");
        return bbb;
    }
}
```

```
public class Species
{
    private int population;
    private double growthRate;
    private String name;

    public Species()
    {
        name = "No Name Yet";
        growthRate = 0.0;
        population = 0;
    }

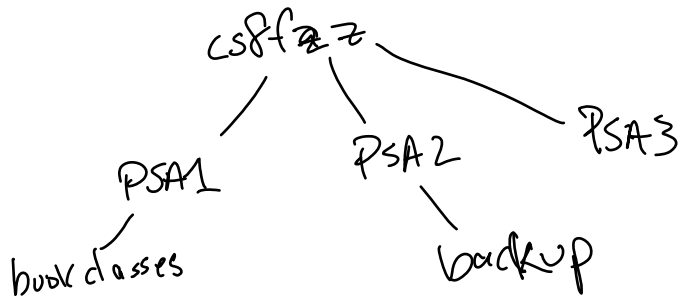
    public Species(String nName, int nPop, double nGR)
    {
        name = nName;
        growthRate = nGR;
        if (nPop >= 0)
            population = nPop;
        else
            population = 0;
    }

    public void setName(String newName) { name = newName;}

    public String toString()
    {
        return ("Name: "+name+ ", "+
                "Population: "+population+ ", "+
                "Growth Rate: "+growthRate);
    }
}
```

6. What command would you use to change directories... Assume you are in the bookClasses directory and you want to go into the backup directory?

- A) `cd backup`
- B) `cd ../backup`
- C) `cd ../PSA2/backup`
- D) `cd ../../backup`
- E) `cd ../../PSA2/backup`
- F) `cd backup`



7. Which of the following are examples of the class `RuntimeExceptions` in Java (hint, this is not a memorization question, think about what things are runtime exceptions): Circle all that apply.

- A) `CannotFindSymbolException`
- B) `IndexOutOfBoundsException`
- C) `IllegalStartOfExpressionException`
- D) `IllegalArgumentException`
- E) `NoSuchMethodException`

8. Draw the array instance variable AND the size instance variable of the `ArrayList` class that would result from executing the following code. Assume that the `ArrayList` `foo` has been initialized such that it looks like this:



```
foo.add(new Integer(7));
foo.add(new Integer(6));
foo.ensureCapacity(5);
foo.remove(2);
```

9. In the code above, what would happen if I replace the last line with `foo.remove(3);`

10. What is printed by the following code?

```
String foo = "supercalifragilistic expial idocious";
int i = 0;
int count = 0;
while ((i = foo.indexOf("c", i)) >= 0)
{
    count++;
    i++;
}
System.out.println(count);
```

11. Write code that will read and print all the lines from a file until it encounters a line containing the substring `####`. That line will not be printed. You may instantiate an `ArrayList` of basetype `String` or not. It's up to you. But use appropriate casting if needed. Use appropriate try and catch blocks.

12. Modify the code above to do the same thing, but to enter those Strings (converted into lowercase) into an ArrayList in alphabetical order and, at the end, print out that ArrayList.

Sample Input:

Vegetable
Mineral
Animal
Plant
Rock####
Cactus

Sample Output:

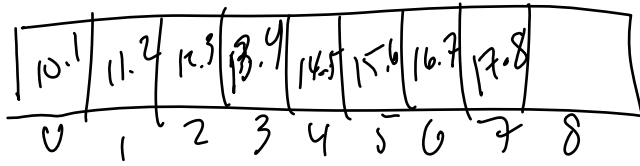
animal
mineral
plant
vegetable

13. Write a static method that takes an ArrayList parameter that contains objects of the type SensorData (class definition as given on PSA2 – on actual exam, this class definition will be provided). The method should create a new ArrayList which is a filtered version of the original array list which contains only those SensorData objects whose overall wave height is greater than the average of all the overall wave heights in the original list. Use appropriate casting as needed.

14. Convert the following code to use a Boolean variable to control the loop behavior (aka Pattern B):

```
public static int indexOf(double [] foo, double toFind)
{
    for (int i = 0; i < foo.length; i++)
    {
        if (foo[i] == toFind)
            return i;
    }
    return -1;
}
```

15. Assume we are calling the ArrayList method `add(3, 33.0)` on the following ArrayList. List the pattern of indexes that would be read from and copied into on each iteration of the loop body in the add method:



Copy Into Array Index Reading From Array Index

Loop iteration 1:

Loop iteration 2:

Add as many lines as needed:

16. Write a loop (header and body) that might be in the Sun ArrayList code to implement the add method. Assume (as we have in class) that these are the ArrayList instance variables (hint use `compareTo` appropriately)

```
private Object[] array;
private int size;
```

17. Describe the efforts you made in following the “always deliverable” process in your PSA2 homework.

18. Declare and instantiate an ArrayList of Species objects with capacity 33. Show a drawing showing the ArrayList instance variables.

19. Explain the difference between an ArrayList's size and its capacity.

20. Explain the difference between a static and a class (instance) variable, using as an example the static variable you implemented in PSA1.

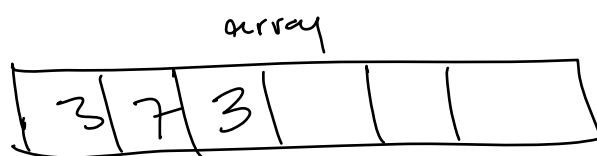
21. Reflect on one thing you have improved in your software development processes this term.

22. Write code which might be found in the String class in Sun's Java code to implement (This should return -1 if the parameter String is not found. The index it returns is the index of where the string "starts" in the calling object.)

```
int lastIndexOf(String str)
```

Returns the index within this string of the rightmost occurrence of the specified substring.

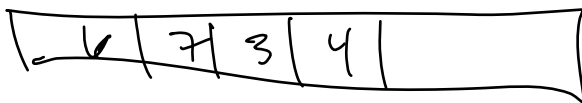
23. Assume you call the ArrayList method add in the following set of cases. In which cases would the add method of the ArrayList class call "ensureCapacity" (and explain why):



size
13

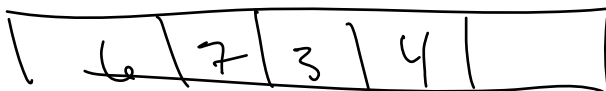
call
add(2, 13)

(Y/N)
ensureCap
called?



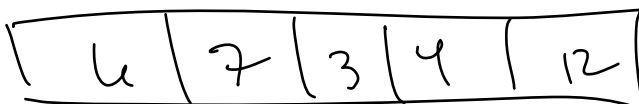
size
4

add(2, 13)



size
4

add(4, 13)



size
5

add(2, 13)

24. Write the code for trimToSize from your SmartArray class. Assume your instance variables are the following:

```
private Object[] array;  
private int size;
```