About this sample exam:

- You may and SHOULD work through this exam yourself and discuss you 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)
- You will be required to write code by hand, we will provide the String and ArrayList method tables from Javadoc 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).

Learning Goals

- 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).
- LG10: Draw a unix directory structure diagram employing the terms: root directory, file suffix, root, leaf, parent, child.
- LG11: Read and use a runtime exception call stack for debugging.
- LG12: Read and write code that uses BufferedReader to read text from a file and catch exceptions appropriately with a simple try and catch block.
- LG14: 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”
- LG15: 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.
• LG16: Calculate the number of different values that can be stored in a variable of a specific size (number of bytes). Apply this to describe how many values can be represented by unicode character representation.
• LG17: Read, understand and use String methods to solve problems and manipulate Strings.
• LG18: Compare dynamically resizable ArrayLists to standard Java arrays.
• LG19: Write code that correctly puts Object type data into an ArrayList and gets it back out (with appropriate casting).
• LG20: Utilize “live code experimenting” techniques to teach yourself about new programming language features.
• LG21: 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.
• LG22: Describe debugging scenarios which you should not engage in when programming for courses at UCSD (and specifically in CSE8B).
• LG23: Apply “always deliverable” methods to your programming assignments.
• LG24: Explain why interface classes have only abstract methods.
• LG25: Use the List interface to work with ArrayLists.
• LG26: Explain the relationship between an ArrayList’s size and the number of “real elements” in that ArrayList.
• LG27: Explain why the Java exception mechanism (throw/try/catch) is better than just printing an error when you find one.
• LG28: Write code to throw standard Java RuntimeExceptions.
• LG29: Describe problem solving/iterating scenarios when foreach is preferred and when foreach is not appropriate.
• LG30: Read and trace code which uses the varargs Java feature.
• LG31: Describe situations when you would want to declare an enum type.
• LG32: Read, trace, and write code that uses 2-dimensional arrays. Specifically being able to use the length attribute appropriately.
• LG33: Draw a visualization of a 2-D array in order to trace code.
• LG34: Trace the execution of simple recursive code execution (including being able to identify the difference in action BEFORE recursive call or action AFTER recursive call).
• LG35: Identify recursive case and base case from a problem statement.
• LG36: Compare recursive programs to real world “recursive” situations.
• LG37: Write a recursive function with more than one recursive case.
• LG38: Draw a call graph for a recursive function with more than one recursive case.
• LG39: Identify subclasses and superclasses and which inherits from what by looking at a class hierarchy diagram.
• LG50: Read and understand code in a subclass that inherits methods from a superclass.
• LG51: Understand the method called “super” and when and how to call it.
• LG52: Describe when one should override a method in a subclass.
• LG53: Be able to write a method to override another in a superclass.
• LG54: Compare and contrast calls to super and this.
• LG55: Be able to name and appropriately use the four types of access supported in Java.
• LG56: Compare and contrast the three basic algorithm analysis forms: best case, average case and worst case.
• LG57: Describe what kinds of test cases should be used to define an algorithm’s worst case performance (perhaps on a specific algorithm like linear or binary search).
• LG58: Trace the execution of recursive binary search and explain in English the process.
• LG59: Write code for recursive searching methods including binary search and directory search.
• LG60: Describe the three pedagogical recommendations that were made for supporting new software developers in industry and where options exist for gaining that experience at UCSD.
• LG61: Trace the execution of a switch statement, including debugging a switch statement with a missing or misplaced break.
• LG62: Describe the kinds of conditional execution that cannot be coded with a switch statement.
• LG63: Draw an example of an inheritance hierarchy.
This problem uses the class Species - defined on the next page.
What is printed by each (marked with bold comments) print statement?

```java
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 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 RuntimeException 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
F) NoSuchFieldException

8. Draw the array 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?

```java
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 cookedValue is greater than the average of all the cookedValues in the original list. Use appropriate casting as needed.

14. Convert the following code to use a Boolean variable to control the loop behavior:

```java
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 \texttt{add(33.0, 3)} 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:

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
10.1 & 11.2 & 13.3 & 14.4 & 15.5 & 16.6 & 17.7 & 18.8 \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
\end{tabular}
\end{center}

<table>
<thead>
<tr>
<th>Copy Into Array Index</th>
<th>Reading From Array Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop iteration 1:</td>
<td></td>
</tr>
<tr>
<td>Loop iteration 2:</td>
<td></td>
</tr>
<tr>
<td>Add as many lines as needed:</td>
<td></td>
</tr>
</tbody>
</table>

16. Explain the difference between an ArrayList’s size and its capacity.
BEGIN SECOND HALF OF TERM

NOTE: Questions 17-21 indicate that you should write code. On an exam, this may appear as “write code” or as “reorder code” or as “select a line of code to complete the code” or “explain in plain English what this code does”, or “trace the following code and show the values in the instance variables when the code is finished”.

17. Write the code for a SmartArray class method that adds an entire array of elements to the current array, starting at a given index.

```java
add(int index, Object[] obj) - This method will insert new elements into this SmartArray at the given index. The elements to be inserted are passed as an Object array. If the index is valid, then each element whose location is greater than or equal to index will get shifted up by the length of the Object array. Then the new objects (from the array) will get stored starting at index. If the index is not valid, then this method should throw an IndexOutOfBoundsException with the message ("Invalid index: " + theInvalidIndex). Size should be updated appropriately. This method has no return value.
```

18. Write the code for a SmartArray class method described below:

```java
lastIndexOf(Object obj) - This method will search for the given object and return the highest index at which it lives in the SmartArray. This method behaves just like indexOf, unless the object lives in the SmartArray in multiple places. If so, then this method will return the largest index at which the object lives, whereas indexOf would return the smallest. Again, if the object is not found then this method should return -1.
```

19. Write the code for a SmartArray class method described below:

```java
containsAll(SmartArray other) - This method will check to see if ALL of the elements of the given SmartArray are contained inside this SmartArray. If there are any elements in the given SmartArray that are not contained inside this SmartArray, then this method should return false. Otherwise, this method should return true.
```

20. Write the code for a SmartArray class described below:

```java
removeConsecutives() – This method will remove all instances of consecutive identical elements in a SmartArray. That is, if the element at index i and index i+1 are the same, the element at index i+1 should be removed.
```

21. Name one SmartArray method when using a foreach loop would be preferred. Name one SmartArray method which using a foreach loop would not be recommended. Describe the differences between the looping behavior of those methods.

22. Write a declaration for a 2-D array where in each row we store the scores a student earned on the quizzes in a class (one row for each student in the class) and each column stores the grades for a specific quiz.

23. What code statement would you write to print out the number of students in the class?

24. What code statement would you write to print out the number of quizzes given in the course?

25. Write the code to calculate the average score on a subset of the quizzes. For example, the professor would edit a file and list the quiz numbers they want to find the overall average on (like 3, 5, 7 would find a single average value from all the student scores on quiz 3, 5, and 7). The result value should be written to a file. Use try and catch appropriately.
26. Suppose we are writing a program for our iPhone. The iPhone stores the day of the week as a number between 1 and 7. Write a switch statement that will print out the appropriate String for the day of the week (Monday, Tuesday, etc.) based on the integer internal storage. Make sure that your code functions properly even if someone hacks the iPhone so that the integer value if has is invalid.

27. For the following code, trace the values of the loop controlling variables for the first 7 iterations:

```java
int[][] foo = new int[40][3];
for (int i = 0; i <= foo.length; i++)
{
    for (int j = 0; j < foo[0].length; j++)
    {
        foo[j][i] = i*j;
    }
}
```

28. The code above has a bug in it.
   28A) Will this bug be a compile time or runtime error?
   28B) Explain in plain English the cause of the error?
   28C) What kind of Java exception will be thrown?
   28D) If this an error that must be caught?

29. Below is the code for a recursive solution to the Fibonacci sequence. Recall that the method takes an integer \( n \) and then returns the \( n \)th Fibonacci number. Draw the call graph for a call to \( \text{fib}(8) \). (Hint: Don’t remember what a call graph looks like? It’s a tree that shows the value passed in for parameter \( n \) each time the method is called. A sample is at: \( \text{http://phoenix.goucher.edu/~kelliher/cs18/feb23.html} \).

```java
public static int fib(int n)
{
    if (n <= 2)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```
30. Write a recursive method to print out all the numbers from 1 to n – counting UP. Call it foo. So a call to foo(10) should print:

1 2 3 4 5 6 7 8 9 10

31. Refer to the following Java class diagram:

Java Inheritance Hierarchy:

```
Vehicle
  |-- FueledVehicle
  |   |-- Car
  |       - description String
  |   |-- Bicycle
  |       - description String
  |-- NonFueledVehicle

```

31A) Name a superclass of Car

31B) Write a constructor for FueledVehicle that takes 4 parameters.

32A) True or False: Private variables are not inherited by their subclasses.

32A) True or False: Private methods are not inherited by their subclasses.

33) Suppose we want to compare linear and binary search. The code for both are in the class lecture slides. Given the following array to search suggest a value (that is indeed in the array) to search on that would support worst case algorithm analysis for linear search. Suggest a value (that is indeed in the array) to search on that would support worst case algorithm analysis for binary search. Based on your worst case analysis, which algorithm is more efficient in that it takes fewer comparisons to find a value in the worst case?

34) Describe one of the methods that you overrode in the Critter class and why overriding is a useful tool in an object-oriented programming language.