SYSC 2004 A - Object-Oriented Software Development – Fall 2017
Midterm Exam: No Aids: Closed book; No calculators: Sample Solutions

Q1(/20+3Bonus)	Q2(/23)	Q3(/16+2Bonus)	Q4(/6+2Bonus)	Deductions	Total(/55+17Bonus)

Last Name: _____ First Name: _____

Student Number: _____

Circle Lab Section: A3 (Mon 3:30pm) A4 (Thu 9:30am) A2 (Thu 4:00pm)

A1 (Fri 11:30am) A5 (Fri 3:30pm)

Please read these instructions before you answer any of the exam questions:

1. Marks will be deducted if you write in the wrong room and/or do not circle your lab section above.
2. You have 75min (1hr15min) for four questions worth **55** marks (plus **17** bonus marks) on 12 pages. Page 10 is blank and may be used as additional answer space and/or rough work.
3. **Ask a question only if you believe there is a mistake on the exam**. Otherwise, make a reasonable assumption and proceed.
4. API documentation for `ArrayList`, `LinkedList`, `Random` and `Iterator` is provided at the end of this question paper. **You may carefully remove the reference page from the exam**.

In Questions 1, 2 and 3 you are going to develop a class called `Exercises` that will allow us to model collections of data using arrays, `ArrayLists`, and `LinkedLists`.

Question 1 (20 marks: 4+5+11+3 bonus)

a) Write the Java code for the `Exercises` class. As we haven't yet discussed the fields or methods, just put ". . ." where those would be written. For full marks, your code must include comments and Javadoc comments for the class, and allow us to access the Java classes `ArrayList`, `LinkedList` and `Random`. (4 marks)

```
import java.util.*;
// or java.util.ArrayList; java.util.LinkedList; and
// java.util.Random;

/**
 * The Exercises class allows us to model collections of
 * data using arrays, ArrayLists, and LinkedLists.
 *
 * @author Lynn Marshall
 * @version 1.0 Sample Solutions; May 20th, 2015
 *
 */
public class Exercises {
        . . .
}
```

1 mark for import(s); 1 mark for description; 0.5 marks each for author and version;
1 mark for signature and braces

b) Our class will have five fields:
`maxSize`: the maximum size of all of our collections of data (Note: not a constant)
`array`: an array of integers
`count`: the number of elements currently stored in the array
`alist`: an `ArrayList` of integers
`llist`: a `LinkedList` of integers

Write the fields. Comments are required for full marks. (5 marks)

```
private int maxSize; // maximum size of the collections
private int[] array; // an array of integers
private int count; // number of elements in array
private ArrayList<Integer> alist; // ArrayList of integers
private LinkedList<Integer> llist; //LinkedList of integers
```

1 mark each; deduct 1.5 marks if "private" is missing for all five; deduct 1.5 marks if no comments; deduct 1 mark if student has <int> instead of <Integer>; deduct 1 mark if [] missing or in the wrong place.

c) Our class will have one constructor. This constructor will have no parameters. It will set`maxSize` to a random number between 5 and 15, using a Random object. The array,`ArrayList`, and `LinkedList` will all start out empty. Write the constructor. Comments, including Javadoc comments, are required for full marks. (11 marks)

```
/**
 * create an Exercise object with a maxSize between 5 and
 * 15 (randomly generated), and an empty array, ArrayList,
 * and LinkedList.
 *
 */
public Exercises() {
     Random r = new Random(); // random object
     // get a random number between 0 and 10, and add 5
     maxSize = r.nextInt(11)+5; // can combine with above
     array = new int[maxSize]; // array has length maxSize
     count = 0; // no elements in array
     // create ArrayList of integers of initial capacity maxSize (see d)
     alist = new ArrayList<Integer>(maxSize);
     // or create ArrayList of integers of initial capacity 10
     alist = new ArrayList<Integer>();
     // create LinkedList of integers
     llist = new LinkedList<Integer>();
}
```

2 marks for comments and Javadoc comments. 1 mark for method signature. 2 marks for initializing maxSize using Random object. 1.5 marks for array. 0.5 marks for count. 1.5 marks each for alist and llist. Can say: maxSize=new Random().nextInt(11)+5;

d) Using the API provided, there are two ways to create the `ArrayList` in the constructor. Give the other way (the one you didn't use in part c), and explain **which** of the two is best in this situation, and **why**. (3 bonus marks)

As per the above, the two options are:

```
// create ArrayList of integers of initial capacity
// maxSize (see part d)
alist = new ArrayList<Integer>(maxSize);
// or create ArrayList of integers of initial capacity
// 10
alist = new ArrayList<Integer>();
```

For this class it is better to use the constructor where we specify the initial capacity, as we know that, in this class, we will never exceed that. This means that the ArrayList will not need to grow, and enlarging the ArrayList is an expensive operation.

1 mark for second way; 2 marks for explanation.

Question 2 (23 marks: 12+3+5+1+1+1)

a) Write a method `addAtEnd` that takes one parameter, an integer, and returns a boolean value. The method attempts to add the integer to the end of the array, `ArrayList`, and `LinkedList`. None of these fields is permitted to contain more than `maxSize` elements. Do not assume that they all contain the same number of elements. If any of them already contain `maxSize` elements, then do not change that field or fields (but do change the ones that are not yet full). If the element is successfully added to all three fields, the method returns true. Otherwise it returns false. Comments, including Javadoc comments, are required for full marks. Note that there is more space on the next page, if needed. (12 marks)

```
/**
 * addAtEnd takes an integer and adds it to the end of the
 * array, ArrayList, and LinkedList, ensuring that none of
 * these ever have more than maxSize elements.  If the
 * integer is successfully added to all three, true is
 * returned.  Otherwise it returns false.
 *
 * @param element value to be added to the lists
 * @return true if element successfully added to all lists, false
 * otherwise
 *
 */
public boolean addAtEnd(int element) {
     boolean allAdded = true; // flag to return
     // attempt to add to array
     if (count < maxSize) { // room in array
     // or if (count < array.length)
          array[count++] = element;
          // or array[count] = element; count++;
     } else {
          allAdded = false; // no room in array
     }
     // attempt to add to ArrayList
```

```
        if (alist.size() < maxSize) { // room in ArrayList
            alist.add(element); // add at end
            // or alist.add(alist.size(),element);
            // (in this case, best to have stored
            // alist.size() in a temporary variable)
        } else {
            allAdded = false; // no room in ArrayList
        }
        // attempt to add to LinkedList
        if (llist.size() < maxSize) { // room in LinkedList
            llist.add(element); // add at end
            // or llist.add(llist.size(),element);
            // (in this case, best to have stored
            // llist.size() in a temporary variable)
            // or llist.addLast(element);
        } else {
            allAdded = false; // no room in LinkedList
        }
        return allAdded;
}
```

3 marks for comments (1 for description, 1 each for @param and @return); 1 mark for signature; 2 marks for correctly calculating and returning the boolean; 2 marks each for correctly adding element to end of each collection, with up to 2 marks deducted for logic errors in doing that. Deduct 0.5 to 1 mark for bad / awful indentation.

b) Using the API provided, there are two ways to add an item to the end of an`ArrayList`, and three ways to add an item to the end of a `LinkedList`. Write code showing the other ways (the ones you did not use above) here. You do not have to rewrite the method above; just write the code to do the additions. (3 marks)

As per the above, for an `ArrayList`:

```
alist.add(element)
alist.add(alist.size(),element)
```

For a `LinkedList`:

```
llist.add(element)
llist.add(llist.size(),element)
llist.addLast(element)
```

1 mark for each option not used in part a).

Instead of adding an element to the end of the three fields, we want to add it to the beginning. For parts c), d), e), and f) assume that the fields are not full, and that the integer to add is stored in `element`. You do not have to write complete methods or Javadoc comments, just the code required with comments explaining what you are doing.

c) Add the integer to the beginning of the array. (5 marks)

```
// move all existing items one to the right, starting
// with the last one
for (int i = count; i>0; i--) { // must use count not maxSize/length
     array[i] = array[i-1];
}
array[0] = element; // add at the beginning
count++; // increment count
```

3 marks for loop; 1 mark for setting array[0]; 1 mark for incrementing count

d) Add the integer to the beginning of the `ArrayList`. (1 mark)

```
alist.add(0,element); // add at index 0 (first)
```

e) Add the integer to the beginning of the `LinkedList`. (1 mark)

```
llist.add(0,element); // add at index 0 (first)
```
or
```
llist.addFirst(element); // add at the beginning
```

f) Add the integer to the beginning of the `LinkedList` a different way. (1 mark)

See e) above. The other option should be included here.

Question 3 (16 marks: 9+3+4+2 bonus)

a) Write a method `outputArray` that has no parameters and does not return anything. This method outputs either:
`The array is empty.`
if the array contains no integers.
Or, for example, if the array contains 7, 1 and 3:
`The array contains: 7 1 3.`
Your method must use a regular "for" loop for full marks. In addition, comments, including Javadoc comments, are required for full marks. (9 marks)

```
/**
 * If the array is empty, outputArray, outputs a message to
 * that effect, otherwise it outputs "The array contains"
 * followed by the values in the array on one line,
 * separated by spaces, and followed by ".".
 *
 */
public void outputArray() {
     if (count == 0) { // array is empty
          System.out.println("The array is empty.");
     } else {
          System.out.print("The array contains:");
          for (int i=0; i<count; i++) {
          // must use count not maxSize/length

               // print each element preceded by a space
               System.out.print(" " + array[i]);
          }
          System.out.println("."); // period, newline
     }
}
```

2 marks for comments and Javadoc comments (must start /** and end */); 1 mark for signature; 2 marks for empty case; 4 marks for non-empty case. Deduct 0.5 to 1 mark for bad / awful indentation.

b) We could write a similar method for the `ArrayList`. However, to avoid duplication you are going to assume that the `ArrayList` is not empty, and just write code (not the entire method) to print (for example):
`The ArrayList contains: 7 1 3.`
Your code must use a "for each" loop. (3 marks)

```
System.out.print("The ArrayList contains:");
// loop through using for each loop
for (Integer i : alist) { // or for (int i : alist)
     System.out.print(" " + i);
}
System.out.println(".");
```

2 marks for correct use of foreach; 1 mark for printing

c) We could write a similar method for the `LinkedList`. However, you are again going to assume that the `LinkedList` is not empty, and just write code (not the entire method) to print (for example):
`The LinkedList contains: 7 1 3.`
Your code must use an `Iterator` object and a while loop. (4 marks)

```
System.out.print("The LinkedList contains:");
// create Iterator object
Iterator<Integer> it = llist.iterator();
Integer i; // or int i (and can also be declared inside while)
// loop through using iterator
while (it.hasNext()) {
     i = it.next(); // can also declare the Integer / int here
     System.out.print(" " + i);
}
System.out.println(".");
```

1.5 marks for creating Iterator object; 0.5 for while; 2 marks for correct use of hasNext()/next(); 0 marks for printing (as it's the same as before)

d) What are two advantages of using an `Iterator` object? (2 bonus marks)

An iterator is useful if our collection does not have an order, for example a set-like collection.
An iterator is also useful if we want to delete some items of the collection after visiting them.

1 mark for each advantage.

Question 4 (6 marks: 6+2 bonus)

Here are the fields from the Player class from the Chance It game.

```
/** The player's name. */
private String name;

/** The Dice object shared by all players. */
private Dice pairOfDice;

/**
 * The value obtained by the most recent roll of the
 * dice
 */
private int currentRoll;

/**
 * The value obtained the first time that the dice are
 * rolled during a turn.  During the turn, if the
 * player rolls this value a second time, the turn ends
 * and the turn score for is 0.
 */
private int firstRoll;

/** The score obtained during the current turn. */
private int turnScore;

/**
 * The player's cumulative score; i.e., the sum of the
 * turn scores for all of the rounds played so far.
 */
private int totalScore;

/**
 * The most recent turnScore obtained by the other
 * player. This value may be useful when formulating
 * the strategy that determines when to stop rolling
 * the dice.
 */
private int opponentsTurnScore;
```

As you may remember, the `Player` class contains a private `stopRolling` method that has no parameters and returns a boolean value. This method is the player's strategy as to when he'll stop rolling the dice. You are to write this method (on the next page), complete with comments and Javadoc comments implementing the strategy that this player will stop rolling if his first roll is seven, or if his current roll is an even number, greater than seven, and at least as good as his opponent's last score. (6 marks + 2 bonus marks if you don't use any "if" statements)

```
/**
 * Implements this player's strategy of when to stop
 * rolling the dice.  Returns true if the player is
 * stopping rolling, false otherwise.
 *
 * @return true if done rolling, false otherwise
 *
 */
private boolean stopRolling() {
     // most elegant and concise way (earns 2 bonus marks)
     return (firstRoll == 7) || // first roll is 7 or
               // current roll is even, greater than 7, and
             ( currentRoll % 2 == 0 && currentRoll > 7 &&
               // greater than or equal to opponent's
               currentRoll >= opponentsTurnScore );

}

// alternative #1 (using one if statement)
if (firstRoll == 7) || (currentRoll % 2 == 0 &&
    currentRoll > 7 &&
    currentRoll >= opponentsTurnScore) {
     return true;
}
return false;

// alternative #2 (2 if's; else's are not needed)
if (firstRoll == 7) {
     return true;
} else if (currentRoll % 2 == 0 && currentRoll > 7 &&
    currentRoll >= opponentsTurnScore) {
     return true;
} else {
return false;
}
```

2 marks for comments and Javadoc comments (1 for description, 1 for @return); 4 marks for boolean expressions and return; 2 bonus marks if no "if" statements.

Extra Space (may be used for your answers or any rough work). If you use it for answers, indicate that next to the space provided for that answer above.

Reference Material: (You may remove this page from the exam.)

API Summary - Class ArrayList<E>

```
// Constructs an empty list with an initial capacity of ten.
ArrayList();

// Constructs an empty list with an initial capacity of
// initialCapacity.  initialCapacity must be positive.
ArrayList(int initialCapacity);

// Appends o (of type E) to the end of this list. Returns true if
// successful, otherwise returns false.
boolean add(E o);

// Inserts the specified element o (of type E) at the specified
// position in this list.  index must be >=0 and <= size().
void add(int index, E o);

// Returns the number of objects stored in this list.
int size();

// Returns the object (of type E) at the specified position (index)
// in the list. index must be >= 0 and < size().
// The object is not removed from the list.
E get(int index);

// Replaces the element at the specified position in this list
// with o (of type E), and returns the element that was replaced.
// index must be >=0 and < size().
E set(int index, E o);

// Returns true if this list has no elements, otherwise returns false.
boolean isEmpty();

// Removes the object (of type E) at the specified position (index)
// in the list. index must be >= 0 and < size().
E remove(int index);

// Returns an Iterator<E> of elements of this list in sequence.
Iterator<E> iterator();
```

API Summary - Class LinkedList<E>

```
// Constructs an empty list.
LinkedList();

// Appends o (of type E) to the end of this list. Returns true if
// successful, otherwise returns false.
boolean add(E o);

// Inserts the specified element o (of type E) at the specified
// position in this list.  index must be >=0 and <= size().
void add(int index, E o);

// Inserts the specified element o (of type E) at the beginning
// of this list.
void addFirst(E o);
```

```
// Inserts the specified element o (of type E) at the end of
// this list.
void addLast(E o);

// Returns the number of objects stored in this list.
int size();

// Returns the object (of type E) at the specified position (index)
// in the list. index must be >= 0 and < size().
// The object is not removed from the list.
E get(int index);

// Replaces the element at the specified position in this list
// with o (of type E), and returns the element that was replaced.
// index must be >=0 and < size().
E set(int index, E o);

// Returns true if this list has no elements, otherwise returns false.
boolean isEmpty();

// Removes and returns the object (of type E) at the beginning
// of the list.
E remove();

// Removes the object (of type E) at the specified position (index)
// in the list. index must be >= 0 and < size().
E remove(int index);

// Returns an Iterator<E> of elements of this list in sequence.
Iterator<E> iterator();
```

API Summary - Class Random

```
// Constructs a new random number generator.
Random();

// Returns a random integer between -2147483648 and 2147483647,
// inclusive.
int nextInt();

// Returns a random integer between 0 and n-1, inclusive.
int nextInt(int n);
```

API Summary - Interface Iterator

```
// Returns true if the iteration has more elements.
boolean hasNext();

// Returns the next element in the iteration.
E next();

// Removes the last element returned by the iterator.
void remove();
```