Spring '23
Dr. Hristescu |

Data Structures and Algorithms
(Review) Quiz #1

Name: _____

Pre-quiz Grade: 25 (out of 30)
Quiz Grade: 60 (out of 70)
Total: 85 (out of 100)

Question1: Complete the Java method below that **computes** and **returns** the sum of all **odd** numbers that are **less than or equal** to the value of the passed parameter. For instance, the following code snippet should display **9** and then **25**:

```
System.out.println(getSumEven(6)); //displays 9
System.out.println(getSumEven(9)); //displays 25
```

```
public static int getSumOdd( int n )     //complete this method
{




}
```

Question2: In the code snippet (3 lines) below an ArrayList variable named **numbers** is declared and populated with several items. The call should display the content of the collection in reverse order.

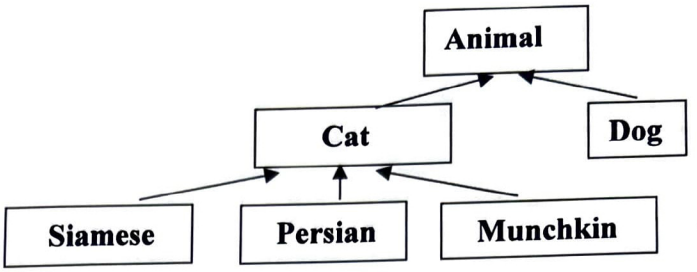
```
ArrayList<Integer> numbers = new ArrayList<Integer>();
// code that inserts items in the collection numbers omitted here
displayReverseDirectIndexAccessInBasicLoop(numbers);
```

Implement the method **displayReverseDirectIndexAccessInBasicLoop** that displays the items in the collection that is passed as parameter from the **last** one to the **first** one using direct index access in a basic for loop. Make sure that the items are separated by one space character and that this code will compile.

```
public static void displayReverseDirectIndexAccessInBasicLoop(ArrayList<Integer> items)
{
```

->over

Question3: Given the following inheritance hierarchy: **Cat** and **Dog** are subclasses of class **Animal**, while **Siamese**, **Persian** and **Munchkin** are subclasses of **Cat**.



I. Given the following 3 variable **declarations**:

```
Animal a;
Cat c;
Dog d;
```

What is the outcome for each of the following individual code snippets **immediately following the 3 declaration statements** above? Options are: **Compiles and Runs (CR), Compilation Error (CE), Run time Error (RE).** Circle **ONLY** one.

a) The given 3 declarations followed by:
```
c = new Munchkin();
```
and then followed by:
```
a = c;
```
(CR) CE RE

(CR) CE RE

b) The given 3 declarations followed by:
```
a = new Dog();
```
and then followed by:
```
d = a;
```
(CR) CE RE

(CR) CE RE

c) The given 3 declarations followed by:
```
a = new Cat();
```
and then followed by:
```
c = (Dog)a;
```
(CR) CE RE

CR (CE) RE

d) The given 3 declarations followed by:
```
a = new Cat();
```
and then followed by:
```
c = (Cat)a;
```
(CR) CE RE

CR CE (RE)

II. What is displayed by the following code fragment immediately following the given 3 declarations?

```
c = new Persian();
```

// Displays:

System.out.println(c **instanceof** Animal); ____

System.out.println(c **instanceof** Cat); ____

System.out.println(c **instanceof** Dog); ____

System.out.println(c **instanceof** Siamese); ____

System.out.println(c **instanceof** Persian); ____

System.out.println(c **instanceof** Munchkin); ____

III. What is displayed by the following code fragment immediately following the given 3 declarations?

```
a = new Cat();
```

// Displays:

System.out.println(a **instanceof** Animal); ____

System.out.println(a **instanceof** Cat); ____

System.out.println(a **instanceof** Dog); ____

System.out.println(c **instanceof** Siamese); ____

System.out.println(c **instanceof** Persian); ____

System.out.println(c **instanceof** Munchkin); ____

Spring '23 | Data Structures and Algorithms | Name: _____
Dr. Hristescu | Quiz #2

Pre-quiz Grade: 30 (out of 30)
Quiz Grade: 40 (out of 70)
Total: 75 (out of 100)

Question 1: Consider a **Fixed Array-based (FAB)** implementation of the **List ADT** with 2 data fields: **items** and **numItems**. Implement the **add** method in this class (there is no superclass other than Object). Avoid code repetition, ensure the **validity of the index** passed as parameter, check for **capacity** and throw an exception if the operation cannot be performed, consider **all possible cases** and don't leave the list object in an inconsistent state. **Use the underlying data structure. NO method calls of any kind**. Trace your code on cases that **require resizing** as well as cases that **don't**. Feel free to draw a picture to aid in your design here (optional):

```
public class ListFAB implements ListInterface
{  private Object []items;  // array of items
   private int numItems;    // number of items in list
public void add(int index, Object item) throws ListIndexOutOfBoundsException
{
```

```
}/*end add. Other methods omitted*/ }//end ListFAB
```

->over

Question 2: Consider the following classic implementations of the **List ADT**

a) **Circular Doubly Linked Structure-based (CDLS)** implementation
b) **Simply Linked Structure-based (SLS)** implementation
c) **Resizable Array-based** implementation **(RA)**

Consider that in the driver we are using a list variable **list** for a list containing **n** items of non-primitive type. Also consider that the array-based implementation uses an array of size **len** where **n <= len**

For your answers to 1, 2 and 3 below, you have to:

- **Schematically draw the object diagram.**
- What is the **total amount of memory** used by an **n** items list using this implementation? Include only the memory used for the list instance and the specific data structure that is used. DO NOT include the memory allocated for the items in the list or for the list variable **list**. Give the **EXACT amount** expressed as a **multiple of 4 bytes. Show your reasoning.**

1. For the **Circular Doubly Linked Structure-based (CDLS)** implementation:

Object diagram:

EXACT amount of memory used (show your reasoning): 4bytes *_____

2. For the **Simply Linked Structure-based (SLS)** implementation:

Object diagram:

EXACT amount of memory used (show your reasoning): 4bytes *_____

3. For the **Resizable Array-based (RA)** implementation (resizable array of given size **len**):

Object diagram:

EXACT amount of memory used (show your reasoning): 4bytes *_____

4. Circle the implementation that is **guaranteed NOT** to use the **LEAST** amount of memory? **CDLS SLS RA**

Briefly explain why:

Spring '23
Dr. Hristescu

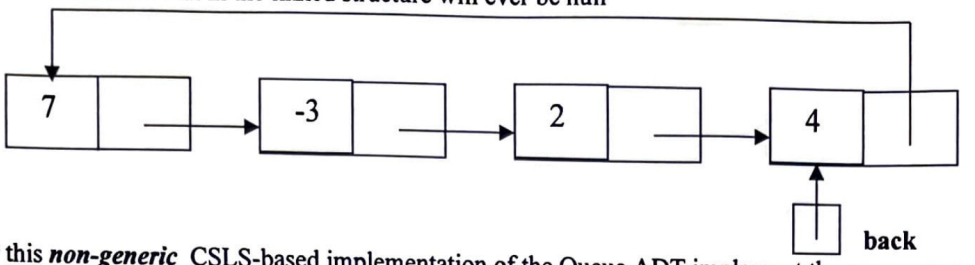
Data Structures and Algorithms
Quiz #3

Name: _____

Pre-quiz Grade:	20	**(out of 20)**
Quiz Grade:	55	**(out of 80)**
Total:	75	**(out of 100)**

Question 1: Consider the **Circular Simply-Linked Structure (CSLS)** implementation of the **Queue ADT** (example below with items 7 -3 2 and 4 below) that has **ONLY ONE** data field called **back** that references the node that contains the last item of the queue. The **CNode** has the classical simply linked structure with *encapsulated* data fields **item** and **next** and whose 1-parameter (item) constructor sets the node's next link to self-reference. Also recall that, if the queue is not empty, the node at the end of the queue, instead of having a null next link, references the node that contains the front item of the queue. Thus:

- if the queue is not empty **back** references the node at the *end* of the queue
- if the queue is not empty **back.getNext()** references the node at the *beginning* of the queue
- no **next** link in the linked structure will ever be null



a) For this ***non-generic*** CSLS-based implementation of the Queue ADT implement the **enqueue** method that adds the item at the end of the queue.
Make sure you **consider all cases**, and the method does not leave your data structure in an inconsistent state.

```
public class QueueCSLS implements QueueInterface
{
 private CNode back;   // the ONLY data field of the class, references the last node in the queue
public void enqueue(Object item)
{
```

```
} //end enqueue method
}//end class QueueCSLS
```

Check the correctness of your implementation on:

1. a general enqueue-ing a new item on the queue above as well as
2. enqueue-ing item **-9** into an **initially empty** queue. For this case show below the **Object Diagram** for the queue collection before and after inserting this item in an empty queue:

Before (queue is empty):

After (queue with one item (-9)):

-->over

Question 2: For the Resizable Array-based (RA) implementation of the **Stack ADT (StackInterface)** using the data fields below implement the following methods:

- default constructor **StackRA.** Make sure that the **initial size of the array** is **3.**
- **pop** method that **removes** the top item from the stack AND **returns** the reference to the popped item.
- **toString** method that collect the content of the stack **starting** with the **top item** of the stack and ending with the bottom item in the stack. **No calls to any StackInterface methods can be used in the implementation of these methods.**

```
public class StackRA implements StackInterface
{
  private Object[] items;
  private int top;   //the index of the top item in the stack

public StackRA()
{




} // end default constructor
. . . . . .
public Object pop()throws StackException   //also returns the reference to the popped item

{




 //end pop method
. . . . . .
public String toString()   //collects items top to bottom and returns the string representation
{




 } //end toString method
 }//end class StackRA
```

Spring '23
Dr. Hristescu

Data Structures and Algorithms
Quiz #4

Name:

Pre-quiz Grade:	10	**(out of 10)**
Quiz Grade:	60	**(out of 90)**
Total:	70	**(out of 100)**

Question 1: Binomial Coefficient computation

a) Give the **recursive** mathematical definition for the Binomial Coefficient of integers **n** and **k** (n>=k>=0)

C(n, k) =

b) Write a **Java method** for the **recursive** solution for the Binomial Coefficient (signature and body):

c) Give the *most efficient* **formula-based** definition for the Binomial Coefficient:

C(n, k) =

d) **Write a Java method** for implementing this *efficient* **formula-based** solution for the Binomial Coefficient (signature and body):

e) Which one of the 2 methods above is more efficient b) or d)? Briefly explain why:

->over

Question 2: Consider having to solve a problem that can have a **recursive** solution and an **iterative** solution.

a) **When** would you choose to implement the **iterative** solution and NOT the recursive one?

b) Give **at least one**, but possibly more **examples** from the problems discussed in class where it is advised to use an **iterative** but NOT the recursive approach and briefly explain **why**: