

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM TEST FOR CS1020
AY2012/13 Semester 2

CS1020 – Data Structures and Algorithms I

6 March 2013

Time allowed: 1 hour 30 minutes

Matriculation number:
(Write using a pen)

Grid for writing matriculation number

INSTRUCTIONS TO CANDIDATES

- 1. This test paper consists of FIFTEEN (15) questions and comprises NINETEEN (19) printed pages.
2. This is a CLOSE BOOK test. You are allowed to bring in ONE (1) piece of handwritten A4 double-sided reference sheet (no photocopies).
3. Fill in your Matriculation Number above clearly with a pen. Note that your matriculation number contains a letter at the back, for example: U084321X or A0091234E.
4. Answer all questions.
5. For MCQs (Q1 to Q12), use the OCR form provided. Shade and write down your matriculation number on the OCR form. You must use 2B pencil to shade/write on the OCR form, or the grading machine might not be able to register your shading.
6. For questions Q13 to Q15, fill in your answers in the space provided. You may use pencil or pen to write your answers.
7. You must submit both the OCR form and this document. It is your responsibility to ensure that you have submitted both to the invigilator at the end of the test.

Table with 4 columns: Section / Question, Possible, Marks, Check. Rows include A. MCQs 1-12, B. Q 13, B. Q 14, B. Q 15, and Total.

SECTION A (12 Multiple Choice Questions: 48 Marks)

Each question has only one correct answer. Shade your answers on the OCR form. 4 marks are awarded for each correct answer; no penalty for wrong answer.

1. Which of the following are NOT valid Java identifiers?

- i. \$\$\$
- ii. _500
- iii. True
- iv. final
- v. a/b

- A. Only (i) and (v)
- B. Only (ii) and (iii)
- C. Only (ii) and (iv)
- D. Only (iv) and (v)
- E. None of options (A), (B), (C), (D) is correct.

2. What is the output of the following code?

```
class TestMystery {
    public static int[] mystery(int[] arr) {
        for (int i=0; i<arr.length; i++) {
            arr[i] *= 2;
        }
        return arr;
    }

    public static void main(String[] args) {
        int[] arr = {1};
        int[] arr2 = mystery(arr);
        System.out.println(arr[0] + " " + arr2[0] + " "
            + (arr == arr2));
    }
}
```

- A. 1 1 false
- B. 2 2 true
- C. 2 1 false
- D. 1 2 true
- E. 2 2 false

3. Given a **Book** class which contains the instance method **getPages()**, and a Book object called **harryPotter1**, which of the following is a valid call?
- A. `Book.getPages();`
 - B. `harryPotter1.getPages();`
 - C. `Book.getPages(harryPotter1);`
 - D. `Book.harryPotter1.getPages();`
 - E. None of the above
4. Which of the following statements are TRUE about the modifiers in Java?
- i. Static methods in a class can be called without creating an instance of that class
 - ii. Private attributes are only accessible through accessor methods
 - iii. "public" can be included in the definition of a class, method, or variable
- A. Only (i)
 - B. Only (ii)
 - C. Only (iii)
 - D. Only (i) and (ii)
 - E. None of options (A), (B), (C), (D) is correct.
5. What is the output of the following code fragment?

```
String s1 = "123";  
String s2 = "123.45";  
String s3 = s2;  
s2 = s2.substring(0, s2.indexOf("."));  
  
System.out.println(s1.equals(s2) + " " + s2.equals(s3)  
                  + " " + s1.equals(s3));
```

- A. false true true
- B. false false true
- C. true true true
- D. true false true
- E. true false false

6. What is the output of the following code?

```
class A {
    private static int changes = 0;
    private int value;

    public A() {
        this(0);
    }

    public A(int value) {
        setValue(value);
    }

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
        changes++;
    }

    public static int getChanges() {
        return changes;
    }
}

class TestA {
    public static void main(String[] args) {
        A a1 = new A();
        A a2 = new A(5);
        a2.setValue(10);
        System.out.println(a1.getValue() + " " +
                           a2.getValue() + " " +
                           A.getChanges());
    }
}
```

- A. 0 5 2
- B. 0 10 2
- C. 0 10 3
- D. 5 5 2
- E. 5 10 3

7. Given the following interface:

```
public interface I {
    public int add(int num);
    public int minus(int num);
}
```

Which of the following classes (F1 – F4) is/are valid implementation(s) of the interface I?

```
class F1 implements I {
    int value = 0;
    public int add(int num) { return value + num; }
}
```

```
class F2 implements I {
    int value = 0;
    public int add(double num) { return value + num; }
    public int minus(double num) { return value - num; }
}
```

```
class F3 implements I {
    int value = 0;
    public int add(int num) { return value + num; }
    public int minus(int num) { return value - num; }
}
```

```
class F4 {
    int[] values = {0};
    public int add(int num) { return values[0] + num; }
    public int minus(int num) { return values[0] - num; }
}
```

- A. Only F1
- B. Only F2
- C. Only F1 and F3
- D. Only F3 and F4
- E. None of options (A), (B), (C), (D) is correct.

8. Given the following methods:

```

public void f() {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    try {
        g(n);
        System.out.println("After ggg");
    } catch (ArithmeticException e) {
        System.out.println("ArithmeticException in fff");
    } catch (IllegalArgumentException e){
        System.out.println("IllegalArgumentException in fff");
    } finally {
        System.out.println("Finally in fff");
    }
}

public void g(int n) throws ArithmeticException,
                IllegalArgumentException {
    try {
        if (n < 0)
            throw new IllegalArgumentException();
        double value = 1 / n;
    } catch(ArithmeticException e) {
        System.out.println("ArithmeticException in ggg");
    }
}

```

Which message is NOT possible to be printed by running **f()**?

- A. After ggg
- B. ArithmeticException in fff
- C. IllegalArgumentException in fff
- D. ArithmeticException in ggg
- E. Finally in fff

9. Given the following program:

```
class Q9 {
    public static void main(String[] args) {
        int pos1 = args[1].indexOf("e");
        int pos2 = args[2].lastIndexOf("e");
        System.out.println(args[3].substring(pos1, pos2));
    }
}
```

What is the output if the program is run as shown below?

```
java Q9 Celebrate forever whatsoever evergreen
```

- A. hats
 B. hatso
 C. rgree
 D. rgreen
 E. None of the above
10. Which of the following overloaded methods in API Math class is called with

Math.max(2, 3.6)

and what is the returned value?

- A. double Math.max(int a, int b); returned value is 3
 B. double Math.max(int a, int b); returned value is 4
 C. double Math.max(double a, double b); returned value is 3.6
 D. double Math.max(int a, double b); returned value is 3.6
 E. There will be an error
11. Given the following statement:

Random num = new Random();

How would we generate a random integer in the range 3 to 10 inclusive?

- i. **num.nextInt(3, 10)**
 ii. **num.nextInt(7) + 3**
 iii. **num.nextInt(8) + 3**
 iv. **num.nextInt() % 8 + 3**
 v. **num.nextDouble() * 8 + 3**
- A. Only (ii)
 B. Only (iii)
 C. Only (i) and (ii)
 D. Only (iii) and (v)
 E. Only (iii), (iv) and (v)

12. Given the following generic **Pair** class:

```
class Pair <S,T> {
    private S first;
    private T second;

    public Pair(S a, T b) { first = a; second = b; }
    public S getFirst()   { return first; }
    public T getSecond()  { return second; }
}
```

How many of the following statements can be compiled with no error?

- i. `Pair <String, int> pair = new Pair <String, int> ("Salary", 3500);`
- ii. `Pair <Double, Integer> pair = new Pair <Integer, Integer> (3, 1458);`
- iii. `Pair <Integer, Integer> pair = new Pair <Integer, Integer> (3, 3.14);`
- iv. `Pair <Integer, Double> pair = new Pair <Integer, double> (3, 1.458);`

- A. None of the statements
- B. Only one of the statements
- C. Only two of the statements
- D. Only three of the statements
- E. All the four statements

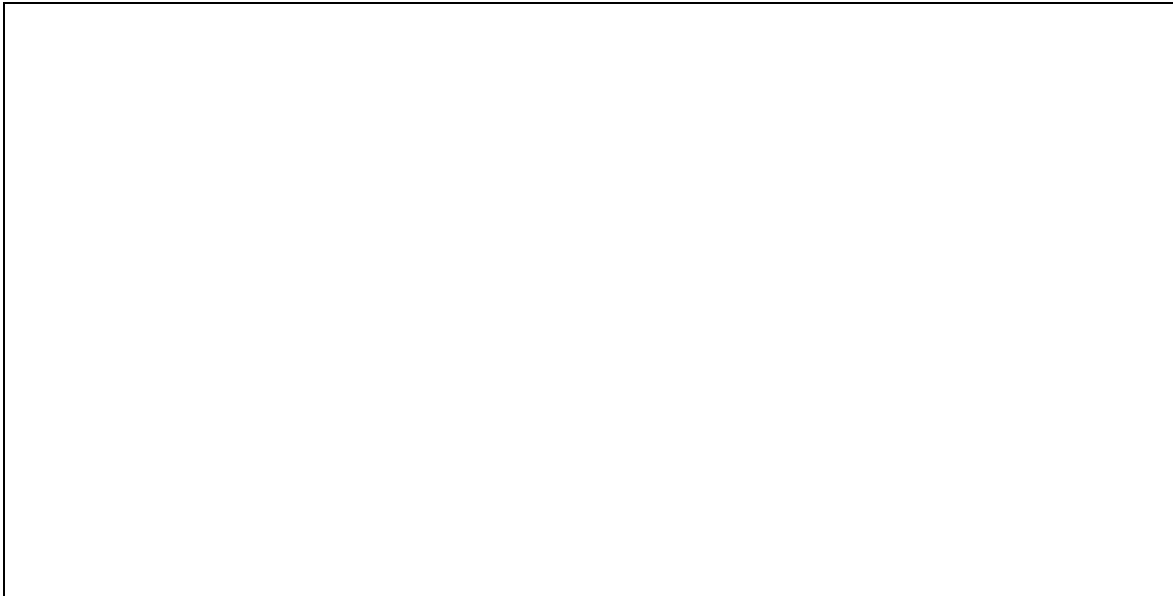
SECTION B (3 Questions: 52 Marks)

13. Given the following program, what is the output?

[6 marks]

```
class Q13 {
    public static void main(String[] args) {
        int[] arr = new int[10];

        for (int x = 0; x < 10; x++) {
            for (int y = x; y < 10; y++) {
                arr[x]++;
            }
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Answer:

14(a) The following **ListNode** class has been introduced in lecture:

```
class ListNode <E> {
    /* data attributes */
    private E element;
    private ListNode <E> next;

    /* constructors */
    public ListNode(E item) { this(item, null); }

    public ListNode(E item, ListNode <E> n) {
        element = item;
        next = n;
    }

    /* get the next ListNode */
    public ListNode <E> getNext() { return next; }

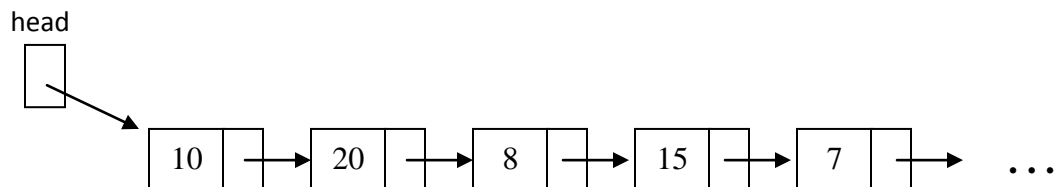
    /* get the element of the ListNode */
    public E getElement() { return element; }

    /* set the next reference */
    public void setNext(ListNode <E> n) { next = n; }
}
```

A linked list has been created, and the following code fragment is executed. Assume that **head** contains the reference to the first node of the linked list.

```
ListNode <Integer> temp = head.getNext();
head.setNext(head.getNext().getNext());
temp.setNext(head);
head = temp;
```

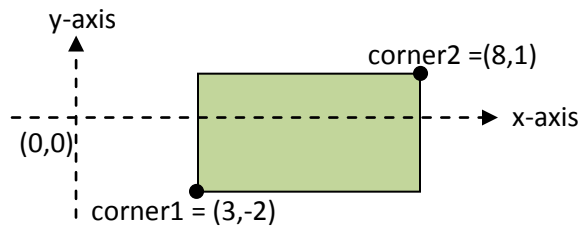
If the linked list before the code fragment above is executed is as shown below:



how is the updated linked list like after the code fragment above is executed? Draw the arrows in the diagram on the next page.

15. [30 marks] A **Rectangle** class is to be defined for rectangles whose sides are parallel to either the x-axis or y-axis. It consists of 2 attributes: **corner1** and **corner2** representing the bottom-left corner and top-right corner of a rectangle respectively. Both corners are objects of the **Point** class defined in Java API. Information on the **Point** class is given in Appendix A.

The diagram below shows an example of a Rectangle object with corner1 at (3, -2) and corner2 at (8, 1).



A client program **TestRectangle.java** is to be written to do the following:

- Complete the method **readInput()** to read in a list of data representing the rectangles. Each input line consists of 4 integers: the x- and y-coordinates of corner1, and the x- and y-coordinates of corner2. A sample input is shown below:

```

2 -2 8 3
10 7 12 8
-1 1 7 7
-5 -2 -3 1
10 7 12 8
    
```

You may assume that there are data for at least one rectangle, that all data represent valid rectangles with positive area, and the first corner read for each rectangle is its bottom-left corner. You are to use the **ArrayList** class in Java API to create and store this list of rectangles. Information on the **ArrayList** class is given in Appendix B.

- Complete the method **checkDuplicate()** to check whether the last rectangle in the list is identical to any of its preceding rectangles. If it is, remove this last rectangle from the list created.
- The program prints the list of rectangles after the removal of the duplicate rectangle (if there is one). The output of the program based on the above sample input is shown below.

```
[{(2,-2):(8,3)}, {(10,7):(12,8)}, {(-1,1):(7,7)}, {(-5,-2):(-3,1)}]
```

Note that each rectangle is printed in the following format by the **toString()** method in **Rectangle** class:

$\{(x_1, y_1) : (x_2, y_2)\}$ where x_1, y_1 are the x- and y-coordinates of corner1, and x_2, y_2 the x- and y-coordinates of corner2.

Complete the following **Rectangle.java** program. You should not modify any code that is given, or add any method that is not shown in the program. Read the comment above each method to understand what is expected. Note that your program must be general! The above sample input is just an example; your program must run on any valid input.

```
import java.awt.*;

class Rectangle {
    // Attributes: corner1: bottom-left; corner2: top-right
    Private Point corner1, corner2;

    // Constructors
    // Default constructor creates a rectangle at corners (0,0) and (1,1)
    // You are to write a single statement using 'this'
    public Rectangle() { // 2 marks
        }

    // Constructor to create rectangle at corners indicated by pt1 and pt2
    // You should call the setCorner() method
    public Rectangle(Point pt1, Point pt2) { // 2 marks
        }


    // Set respective corner (which==1: corner1; which==2: corner2)
    // You may assume that which is either 1 or 2
    public void setCorner(int which, Point pt) {
        if (which == 1)
            corner1 = pt;
        else
            corner2 = pt;
    }

    // Get respective corner (which==1: corner1; which==2: corner2)
    // You may assume that which is either 1 or 2
    public Point getCorner(int which) { // 4 marks
        }
}
```

```
// Overriding toString() method
public String toString() {

    return "(" + getCorner(1).x + "," + getCorner(1).y + "):(" +
        getCorner(2).x + "," + getCorner(2).y + ")";

}

// Overriding equals() method
public boolean equals(Object obj) { // 6 marks
    
}
}
```

Complete the following **TestRectangle.java** program. You should not modify any code that is given, or add any method that is not shown in the program. Read the comments above each method to understand what is expected.

```
import java.util.*;
import java.awt.*;

class TestRectangle {

    public static void main(String[] args) {
        ArrayList<Rectangle> rectangles = readInput();
        checkDuplicate(rectangles);
        System.out.println(rectangles);
    }

    // Read in data for a list of rectangles 8 marks
    readInput() {

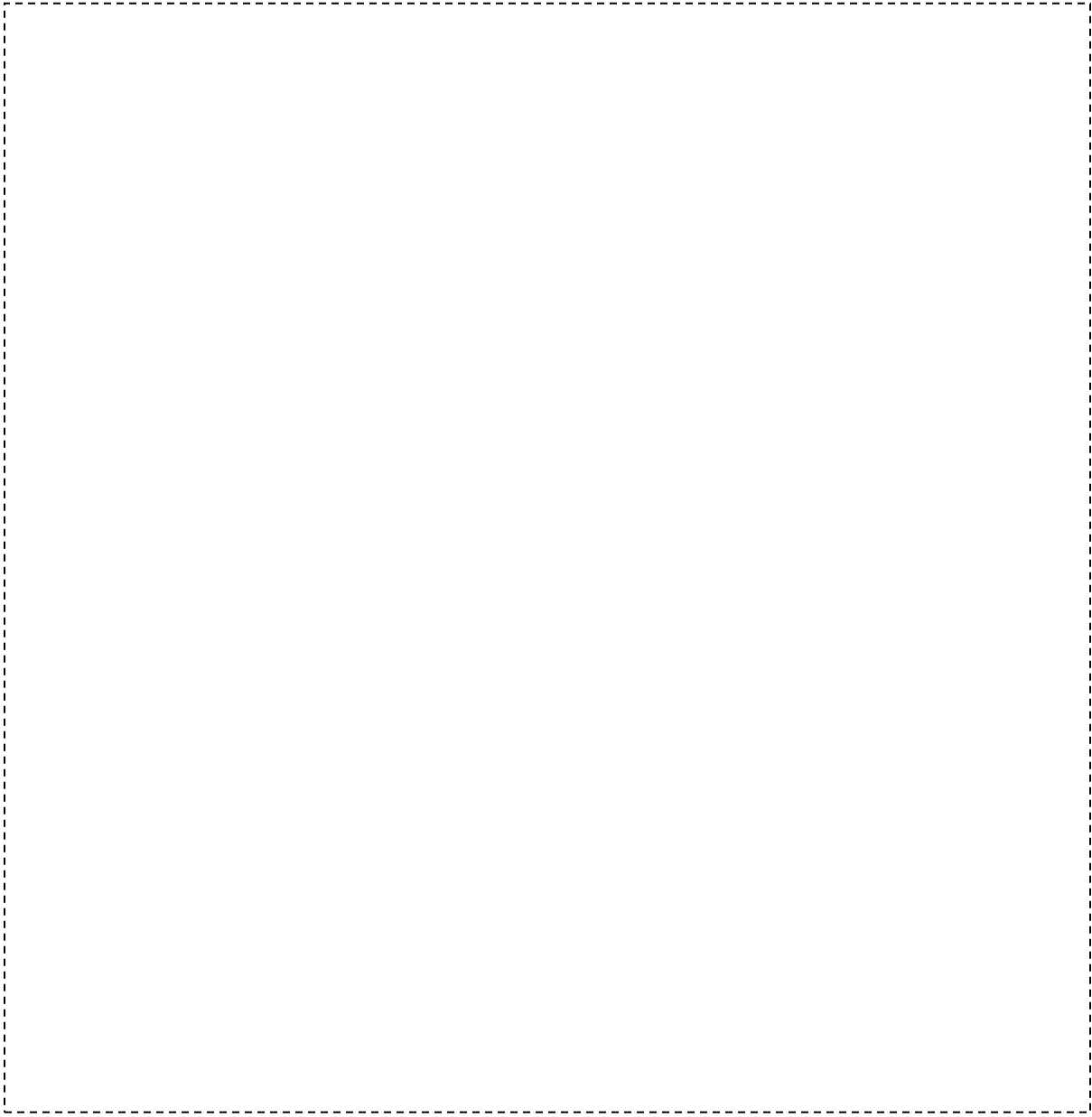
    }
}
```

Complete
the header
of the
method
here

```
// To check if the last rectangle in the list is identical to  
// any of the other preceding rectangles. If so, remove the  
// last rectangle from the list. 8 marks
```

```
public static void  
checkDuplicate( ) {
```

Complete
the
parameter
here



```
}
```

```
}
```


Appendix A: java.awt.Point*Attributes*

```
public int x
public int y
```

Constructors

Point()	Constructs and initializes a point at the origin (0, 0) of the coordinate space.
Point(int x, int y)	Constructs and initializes a point at the specified (x, y) location in the coordinate space.
Point(Point p)	Constructs and initializes a point with the same location as the specified Point object.

Methods

Modifier and Type	Method and Description
boolean	equals(Object obj) Determines whether or not two points are equal.
Point	getLocation() Returns the location of this point.
double	getX() Returns the X coordinate of this Point2D in double precision.
double	getY() Returns the Y coordinate of this Point2D in double precision.
void	move(int x, int y) Moves this point to the specified location in the (x, y) coordinate plane.
void	setLocation(double x, double y) Sets the location of this point to the specified double coordinates.
void	setLocation(int x, int y) Changes the point to have the specified location.
void	setLocation(Point p) Sets the location of the point to the specified location.
String	toString() Returns a string representation of this point and its location in the (x, y) coordinate space.
void	translate(int dx, int dy) Translates this point, at location (x, y), by dx along the x axis and dy along the y axis so that it now represents the point (x+dx, y+dy).

Appendix B: java.util.ArrayList <E>*Constructors*

ArrayList()	Constructs an empty list with an initial capacity of ten.
ArrayList(Collection<? extends E> c)	Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList(int initialCapacity)	Constructs an empty list with the specified initial capacity.

Methods

Modifier and Type	Method and Description
boolean	add(E e) Appends the specified element to the end of this list.
void	add(int index, E element) Inserts the specified element at the specified position in this list.
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	clear() Removes all of the elements from this list.
Object	clone() Returns a shallow copy of this <code>ArrayList</code> instance.
boolean	contains(Object o) Returns <code>true</code> if this list contains the specified element.
void	ensureCapacity(int minCapacity) Increases the capacity of this <code>ArrayList</code> instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
E	get(int index) Returns the element at the specified position in this list.
int	indexOf(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	isEmpty() Returns <code>true</code> if this list contains no elements.

Iterator<E>	iterator() Returns an iterator over the elements in this list in proper sequence.
int	lastIndexOf(Object o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E>	listIterator() Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<E>	listIterator(int index) Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
E	remove(int index) Removes the element at the specified position in this list.
boolean	remove(Object o) Removes the first occurrence of the specified element from this list, if it is present.
boolean	removeAll(Collection<?> c) Removes from this
protected void	removeRange(int fromIndex, int toIndex) Removes from this list all of the elements whose index is between <code>fromIndex</code> , inclusive, and <code>toIndex</code> , exclusive.
boolean	retainAll(Collection<?> c) Retains only the elements in this list that are contained in the specified collection.
E	set(int index, E element) Replaces the element at the specified position in this list with the specified element.
int	size() Returns the number of elements in this list.
List<E>	subList(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified <code>fromIndex</code> , inclusive, and <code>toIndex</code> , exclusive.
Object[]	toArray() Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	toArray(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
void	trimToSize() Trims the capacity of this <code>ArrayList</code> instance to be the list's current size.

== END OF PAPER ==