Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Οντοκεντρικός Προγραμματισμός
Βασίλης Χριστοφίδης

Τελική Εξέταση (3 ώρες)
Ημερομηνία: 28 Ιουνίου 2001

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Ασκηση 1 (10 μονάδες)

Οι προβιβάσεις τύπων (type promotions) σε μοναδιαίους και δυαδικούς τελεστές (operators) είναι μια προγραμματιστική ευκολία που όμως επιβαρύνει τον διερμηνέα (interpreter) της Java με τον υπολογισμό των κατάλληλων μετατροπών των τύπων των τελεσταίων (operands) που εμπλέκονται σε μια Java έκφαση (statement). Π.χ. σε μια αριθμητική έκφραση που χρησιμοποιεί ενα δυαδικό τελεστή οι τύποι των τελεσταίων θα μετατραπούν στον «ευρύτερο» ("widest") απο τους δύο (Βλέπε διαφάνειες μαθήματος). Αν κανένας απο τους δύο δεν είναι ευρύτερος απο τον τύπο int και οι δύο τελεσταίοι θα μετατραπούν σε int. Ο τελικός τύπος του αριθμητικού τελεστή θα είναι ο τύπος των τελεσταίων του μετά απο όλες τις απαραίτητες μετατροπές. Θεωρήστε την παρακάτω αριθμητική έκφραση

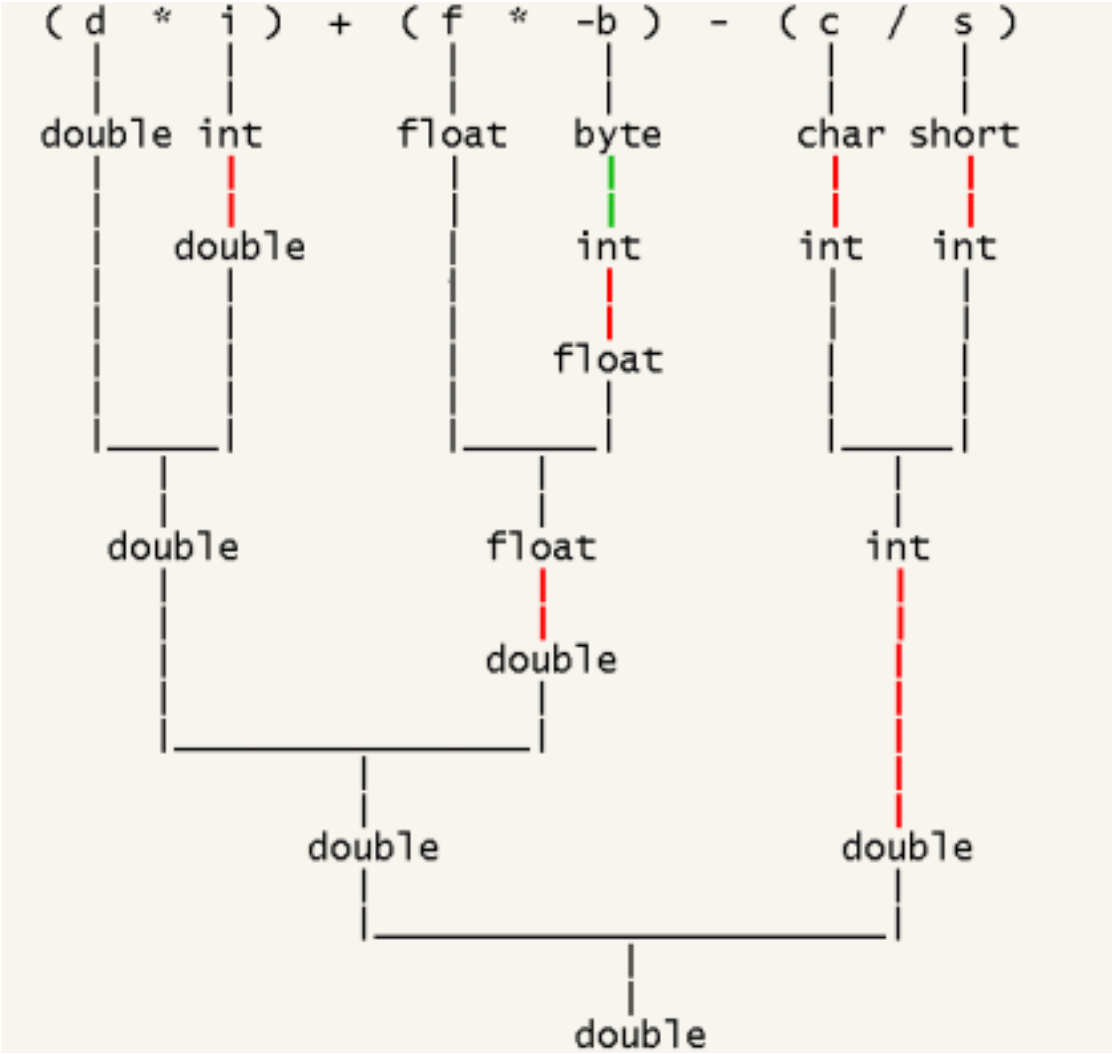
```
(d * i) + (f * -b) - (c / s)
```

όπου οι μεταβλητές έχουν τις ακόλουθες δηλώσεις τύπων:

```
double d;
int i;
float f;
byte b;
char c;
short s;
```

Δώστε σε μια δενδροειδή αναπαράσταση όλες τις μετατροπές των τύπων των τελεστών που εμπλέκονται στην αριθμητική έκφραση καθώς και τον τελικό της τύπο. Υποδείξτε καθαρά στην απάντησή σας τις μοναδιαίες και δυαδικές προβιβάσεις τύπων που επιτελούνται απο τον διερμηνέα της Java.

Λύση:
Put another way, if we let T denote the wider of the two types of operands, we then ask whether or not T is wider than int. If so, both operands and converted to T, otherwise, they are both converted to int. The resulting type of the operation is then the type of its operands. In the following figure the red bars (|) indicate binary numeric promotion while the green bars (|) indicate unary numeric promotion.



Ασκηση 2 (10 μονάδες)

Θεωρήστε την παρακάτω διεπαφή αντικειμένων (interface) Java:

```
interface I {
  int f1();
  boolean f2(int i);
}
```

Για κάθε μία απο τις παρακάτω κλάσεις αντικειμένων Java απαντήστε ΝΑΙ άμα η δήλωση της κλάσης γίνεται αποδεκτή ή ΟΧΙ απο τον μεταφραστή της Java και ποιο συγκεκριμένα απο τον ελεγκτή τύπων:

(α)

```
class C1 implements I {
  public int f1() { return 2; }
  public boolean f2(int j) { return false; }
}
```

Λύση: ΝΑΙ

(β)

```
class C2 implements I {
    public int f1(int i) { return i; }
    public boolean f2(int j) { return false; }
  }
```

Λύση: ΟΧΙ

(γ)

```
  class C3 implements I {
    public int f1() { return 2; }
  }
```

Λύση: ΟΧΙ

(δ)

```
  class C4 implements I {
    public int f1() { return 0; }
    public boolean f2(int j) { return false; }
    public void foo() { }
  }
```

Λύση: ΝΑΙ

(ε)

```
  abstract class C5 extends C4 implements I {
    public int f1(int i) { return i; }
    public double foo() { return 2.0 }
  }
```

Λύση: ΝΑΙ

Ασκηση 3 (30 μονάδες)

Σας δινεται ο πλήρης ορισμός μιας κλάσης `Person` και μέρος του ορισμού μιας κλάσης `FamilyInfo`:

```
  class Person {
   public String name; //Assume this field is
                       //unique for every Person
   public int birthYear;
   public String mother;
   public String father;
  }

  class FamilyInfo {
//keeps track of people in a family
//The data structure used is an array of Persons
   private Person[] list;
   private int familySize;//# of people currently
                          //in this list
```

```
    //Other methods not shown here would include
    //methods to build a FamilyInfo, methods for
    //adding new Persons to it, and methods for
    //searching and printing it

    //Returns the integer index in our list of the
    //Person with the given 'name', or -1 if that
    //Person isn't in our list
    ---------- int find (String name) {
        // Details omitted
    }

    //Prints the given 'name' and the names of all
    //that Person's maternal ancestors stored in
    //this FamilyInfo. That is, prints the mother,
    //grandmother, great grandmother, and so on,
    //until we have no more information. (After
    //all, we only have information on a finite
    //number of people)
    public ------printMaternalAncestors (String
                                          name) {
        // Details omitted
    }
}
```

(α) Τι είδος μεταβλητής είναι η "`list`"? Βάλτε σε ένα κύκλο την σωστή απάντηση.

instance variable **class variable**

Λύση: instance variable

(β) Η μεταβλητή `list` δεν πρέπει (και δεν είναι) δηλωμένη σαν `public`. Γιατί?

Λύση: It's used within the class to represent the family, but there are other ways to represent a family, and we might want to change our choice later; that would be impossible if other software had access to the representation. Also, allowing access to the representation is not good security-wise.

(γ) Είναι απαραίτητο το μέλος δεδομένων (instance variable) `familySize`? Αν ΟΧΙ δικαιολογήστε το γιατί.

Λύση: ΟΧΙ γιατί οι πίνακες στην Java είναι επίσης αντικείμενα και διαθέτει μια μεταβλητή δεδομένων τύπου `int` με όνομα `length` που εξυπηρετεί τον ίδιο σκοπό με την `familySize`

(δ) Η μέθοδος `find` πρέπει να δηλωθεί σαν στατική (`static`) ή όχι? Ποιό είναι το καταλληλότερο αναγνωριστικό δικαιώματος πρόσβασης (visibility modifier) για την μέθοδο `find` (δηλ τι πρέπει να γράψουμε στην αρχή της δήλωσης της μεθόδου στην θέση των `-----`)?

Λύση: ΟΧΙ `private`

(ε) Ποιός είναι ο τύπος εξόδου (return type) της μέθοδου `printMaternalAncestors` (δηλ τι πρέπει να γράψουμε στην αρχή της δήλωσης της μεθόδου στην θέση των `------`)?

Λύση: `void`

(ζ) Ας υποθέσουμε ότι έχουμε δημιουργήσει ένα στιγμιότυπο (instantiated) της μεταβλητή "`myFamily`" τύπου `FamilyInfo` και αποθηκεύσει σε αυτήν την παρακάτω πληροφορία (χρησιμοποιώντας κατάλληλες μεθόδους που για λόγους ευκολίας δεν δίνουμε σε αυτήν την άσκηση):

1. Ο Βασίλης γεννήθηκε το 1964, και οι γονείς του είναι ο Μιχάλης και η Ειρήνη.
2. Η Ειρήνη γεννήθηκε το 1938 και οι γονείς της είναι ο Γιώργος και η Παναγιώτα.

Δεν έχουμε περαιτέρω πληροφορία για τον Μιχάλη, τον Γιώργο και την Παναγιώτα.

Δώστε ένα διάγραμμα που απεικονίζει την κατάσταση (state) του αντικειμένου `myFamily` μετά απο την καταχώρηση της παραπάνω πληροφορίας.

Λύση:

Βασίλης	1964	Μιχάλης	Ειρήνη	`Person` Methods
Ειρήνη	1938	Γιώργος	Παναγιώτα	`Person` Methods
2 (`familySize`)				

(η) Συμπληρώστε το σώμα της μεθόδου `printMaternalAncestors`. Μπορείτε να χρησιμοποιήσετε για αυτό τον σκοπό μια βοηθητική μέθοδο `find` η οποία για λόγους ευκολίας θεωρήστε οτι έχει υλοποιηθεί. Η μέθοδός σας μπορεί να τερματίζει όταν βρίσκει το όνομα ενός πρόγονου για τον οποίο δεν υπάρχει σχετική πληροφορία στην λίστα.

Λύση:

```
while (true) {
  System.out.println(name);
  int index = find(name);
  if (index == -1) return; // or break
      name = list[index].mother;
  }
```

(θ) Αν η μεταβλητή `myFamily` περιέχει k πρόσωπα και καλέσουμε τη `printMaternalAncestors`,ποιός είναι ο μέγιστος αριθμός εκτελέσεων της μέθοδου `find`?

Λύση: k + 1

(ι) Το βασικό πρόβλημα της κλάσσης `Person` που σας δόθηκε στην άσκηση είναι η θεώρηση μοναδικών ονομάτων για την ταυτοποίηση των προσώπων. Εκμεταλευτείτε τον μηχανισμό αναφοράς των αντικειμένων στην Java ώστε να αναπαραστείσετε καλύτερα τη σχέση κληρονομικότητας μεταξύ φυσικών προσώπων χρησιμοποιώντας την μοναδική τους ταυτότητα σαν αντικείμενα. Π.χ. το αντικείμενο με το όνομα «Βασίλης» αναφέρεται στο αντικείμενο με το όνομα «Μιχάλης» μέσω της μεταβλητής δεδομένων `father` και στο αντικείμενο με το όνομα «Ειρήνη» μέσω της μεταβλητής δεδομένων `mother`. Πως πρέπει να τροποποιηθεί η δήλωση της κλάσσης `Person` σύμφωνα με την παραπάνω αναπαράσταση?

Λύση:

```
class Person {
 public String name;
 public int birthYear;
 public Person mother;
 public Person father;
}
```

(κ) Η κλάσση `FamilyInfo` είναι πλέον απαραίτητη? Αν ΟΧΙ προτείνετε μια νέα μέθοδο `printMaternalAncestors` εκτύπωσης των ονομάτων των προγόνων ενός προσώπου. Σε ποια κλάσση πρέπει να δηλωθεί, ποιες οι παράμετροί της και ο τύπος εξόδου της και ποια είναι η καινούργια υλοποίησή της?

Λύση: ΟΧΙ Καινούργια μέθοδος `public void printMaternalAncestors()` στην κλάσση `Person` με το ακόλουθο σώμα:

```
If (this.mother != null) {
  System.out.println(this.mother.name);
  this.mother.printMaternalAncestors();
  return;}
```

Ασκηση 4 (10 μονάδες)

Θεωρήστε τις παρακάτω κλάσεις και μεθόδους Java:

```
public class C1{
 public void method1() throws
                              Exception1,Exception2{
  try{
   System.out.println("m1_1");
   //some code here that will randomly throw
   //Exception1, 2, or 3
   System.out.println("m1_2");
  }catch(Exception3 e3){
   System.out.println("m1_3");
  }
  System.out.println("m1_4");
 }
 public void method2() throws Exception1{
  try{
   method1();
   System.out.println("m2_1");
  }catch(Exception2 e2){
   System.out.println("m2_2");
  }
  System.out.println("m2_3");
 } public void method3(){
  try{
   method2();
   System.out.println("m3_1");
  }catch(Exception1 e1){
   System.out.println("m3_2");
  }
  System.out.prinltn("m3_3");
 }
public class Tester{
  public static void main(…){
    C1 c = new C1();
    c.method3();
  }
}
```

Τι θα επιστραφεί στην έξοδο του προγράμματος στις παρακάτω περιπτώσεις:

(α) Προκύπτει η Exception3
(β) Προκύπτει η Exception2
(γ) Προκύπτει η Exception1

Λύση:

(α) Προκύπτει η Exception3

m1_1

m1_3

m1_4

m2_1

m2_3

m3_1

m3_3

(β) Προκύπτει η Exception2

m1_1

m2_2

m2_3

m3_1

m3_3

(γ) Προκύπτει η Exception1

m1_1

m3_2

m3_3

Ασκηση 5 (40 μονάδες)

Θεωρήστε εναν αφαιρετικό τύπο δεδομένων (ADT) `ΑκολουθίαμεΔείκτες` (RankedSequence). Αυτός ο ADT αναπαριστά μια ακολουθία αντικειμένων για καθένα απο τα οποία συσχετίζουμε ένα δείκτη (rank) εμφάνισής του στην ακολουθία (ένα αντικείμενο μπορεί να εμφανίζεται πολλές φορές σαν στοιχείο της ακολουθίας). Το αντικείμενο στην κεφαλή της ακολουθίας έχει δείκτη 0 και αυτό στην ουρά της κ-1, όπου κ ειναι ο συνολικός αριθμός των αντικειμένων της ακολουθίας. Σημειώστε ότι η αρίθμηση των αντικειμένων της ακολουθίας είναι συνεχής δηλ δεν υπάρχουν τρύπες στην αρίθμηση. Η διεπαφή χρήσης του ADT `ΑκολουθίαμεΔείκτες` περιλαμβάνει τις παρακάτω λειτουργίες

(operations) (η μεταβλητή `o` χρησιμοποιείται για να δηλώσει ένα αντικείμενο ενώ η μεταβλητή `r` για να δηλώσει τη θέση του στην ακολουθία):

- `init()` δημιουργεί μια άδεια ακολουθία απο δείκτες.
- `isEmpty()` επιστρέφει `True` αν ο ADT δεν περιέχει δεδομένα και `False` στην αντίθετη περίπτωση.
- `length()` επιστρέφει έναν ακέραιο μεγαλύτερο η ίσο του μηδενός ο οποίος δηλώνει τον αριθμό των στοιχείων που περιέχει ο ADT.
- `positionOf(o)` επιστρέφει έναν ακέραιο που δηλώνει την θέση του πρώτου στιγμιότυπου του `o` που βρίσκουμε κατά την αναζήτηση των στοιχείων του ADT ακολουθώντας μια προκαθορισμένη σειρά διάταξης του ADT (δηλαδή απο την αρχή της ακολουθίας).
- `elementAtRang(r)` επιστρέφει το αντικείμενο που βρίσκεται στην `r` στη θέση των στοιχείων που περιέχει ο ADT.
- `insertAtRang(o,r)` εισάγει το αντικείμενο `o` στην θέση `r` των στοιχείων του ADT ακολουθώντας τα παρακάτω βήματα:
  1. Δημιουργούμε ενα νέο στοιχείο με τύπο συμβατό με αυτόν που έχει δηλωθεί ο ADT (π.χ. ακολουθία ακεραίων, συμβολοσειρών, κλπ.).
  2. Συνδέουμε το καινούργιο στοιχείο με τα υπόλοιπα στοιχεία του ADT, ετσι ώστε το καινούργιο στοιχείο να καταλαμβάνει την `r` στη θέση και τα υπόλοιπα να καταλαμβάνουν τις θέσεις `r+1`, `r+2`, κλπ. σύμφωνα φυσικά με την αρχικά προκαθορισμένη σειρά διάταξης του ADT (δηλαδή απο την αρχή της ακολουθίας).
  3. Θέτουμε το αντικείμενο `o` στη θέση του στοιχείου που δημιουργήσαμε και τοποθετήσαμε στην ακολουθία.
- `DeleteAtRang(r)` αφαιρεί `r` στο στοιχείο του ADT και αποκαθιστά την σειρά των υπόλοιπων στοιχείων (ουσιαστικά εκτελεί τις αντίστροφες εντολές που χρησιμοποιούνται στην `insertAtRang`).
- `ReplaceAtRang(r,o)` αντικαθιστά το στοιχείο στην θέση `r` με το αντικείμενο `o` χωρίς να επηρεάζεται η σειρά των υπόλοιπων στοιχείων του ADT.
- `Output()` γράφει τα δεδομένα του ADT στο δίσκο, είτε σαν αντικείμενα της χρησιμοποιούμενης γλώσσας προγραμματισμού (π.χ. χρησιμοποιώντας την Java `Object` `serialization` `interface`), είτε σαν ASCII συμβολοσειρές (π.χ. χρησιμοποιώντας την Java `Object` `toString()` `method`).

(α) Δημιουργήστε μια διεπαφή αντικειμένων (interface) Java για την αναπαράσταση του ADT `ΑκολουθίαμεΔείκτες`. Ταξινομείστε σε κατηγορίες τις παραπάνω λειτουργίες του ADT (Accessors, Transformers, κλπ.) και δηλώστε με την μορφή σχολίων Java (κατά προτίμηση σχολίων Javadoc) τις **pre, post** και **invariant** συνθήκες που πρέπει να ικανοποιούνται απο κάθε λειτουργία. Δώστε για κάθε μία από αυτές την υπογραφή (signature), το αναγνωριστικό δικαιώματος πρόσβασης (visibility modifier) καθώς και τις εξαιρέσεις (exceptions) που μπορούν να δημιουργούν οι παραβιάσεις των προηγούμενων συνθηκών.

Λύση:

```
package SoftwareInterfaces;
/**
 * This is the interface of the ADT ranked sequence.
 * <p> A ranked sequence is a sequence of elements, each of which
 * has an associated integer <em>rank</em>.  The rank is the
 * offset of the element from the beginning of the sequence.  The
 * head (i.e., first) element of the sequence has rank 0, the
 * next element has rank 1, and so forth. Each element is a Java
 * object;  primitive data must be wrapped with instances of the
 * appropriate wrapper classes (e.g., <code>Integer</code>).
 *
 * Interface Invariant:
 * for 0 <= r < length(), elementAtRank(r) == element at rank r
 * of the sequence)
 * @author XXXXXXXXXXX
 * @version 28 June 2001
 */
public interface RankedSequence
{
    /**
     * Insert a new element at the given rank into the sequence S
     * (Transformer).
     * @param o the object to be inserted
     * @param r the rank at which to insert the object o
     * @throws BoundaryViolationException (e.g., if precondition
     * not satisfied)
     * <dt><b>Preconditon:</b>
     * <dd><code>S</code> is a valid instance of RanckedSequence
     * and <dd><code>0 <= r <= length()</code>
     * <dt><b>Postcondition:</b>
     * <dd>object <code>o</code> added as element of S at rank
     * <code>r</code>. Ranks of all S elements (if any) with old
     * <code>ranks >= <code>r</code> are increased by one.
     */
    public void insertAtRank(Object o, int r)
                                throws BoundaryViolationException;
    /**
     * Delete the element at the given rank from the sequence S
     * (Transformer).
     * @param r the rank at which to delete the object
     * @throws BoundaryViolationException (e.g., if precondition
     * not satisfied)
     * <dt><b>Preconditon:</b>
     * <dd><code>S</code> is a valid instance of RanckedSequence
     * and <dd><code> 0 <= r < length()</code>
     * <dt><b>Postcondition:</b>
     * <dd>element of S at rank <code>r</code> is removed.
     * Ranks of all S elements (if any) with old <code>ranks >
     * r</code> are decreased by one.
     */
    public void deleteAtRank(int r) throws
                                      BoundaryViolationException;
    /**
     * Replace the element at the given rank in the sequence S
     * (Transformer).
     * @param r the rank at which to replace the object
     * @param o the replacement object
     * @throws BoundaryViolationException (e.g., if precondition
     * not satisfied)
     * <dt><b>Preconditon:</b>
     * <dd><code>S</code> is a valid instance of RanckedSequence
     * and <dd><code>0 <= r < length()</code>
     * <dt><b>Postcondition:</b>
     * <dd>element of S at rank <code>r</code> is replaced by
     * object <code>o</code>.
     */
```

```
    public void replaceAtRank(int r, Object o)
                                throws BoundaryViolationException;
    /**
     * Retrieve the element at the given rank from the sequence S
     * (Accessor).
     * @param r the rank whose value is to be returned
     * @throws BoundaryViolationException (e.g., if precondition
     * not satisfied)
     * <dt><b>Preconditon:</b>
     * <dd><code>S</code> is a valid instance of RanckedSequence
     * and <dd><code>0 <= r < length()</code>
     * @return the element at rank <code>r</code>
     */
    public Object elementAtRank(int r)
                                throws BoundaryViolationException;
    /**
     * Determine the length of the sequence S (Accessor).
     * @return the number of elements in S.
     * @throws BoundaryViolationException (e.g., if precondition
     * not satisfied)
     * <dt><b>Preconditon:</b>
     * <dd><code>S</code> is a valid instance of RanckedSequence
     * <dt><b>Postcondition:</b>
     * <dd><code>0 <= length()</code>
     */
    public int length();

    /**
     * Determine whether the sequence S is empty (Accessor).
     * @return true if S contains no elements; otherwise false.
     * <dt><b>Preconditon:</b>
     * <dd><code>S</code> is a valid instance of RanckedSequence
     * <dt><b>Postcondition:</b>
     * None
     */

    public boolean isEmpty();

    /**
     * Get a string representation of a sequence S (Accessor).
     * @return a string ss from a sequence s
     * <dt><b>Preconditon:</b>
     * <dd><code>s </code> is valid instances of RanckedSequence
     * <dt><b>Postcondition:</b>
     * <dd> ss is the sequence s expressed in the format:
     * <code> for 0 <= r < size(), Sequence[r]: elementAtRank(r)
     * </code>
     */
  public String toString();

} //End of interface RankedSequence
```

Σημειώστε ότι οι κατασκευαστές αντικειμένων (constructors) του ADT δεν εμφανίζονται στην αντίστοιχη διεπαφή Java που αναπαριστά την αφαιρετική τους συμπεριφορά.

(β) Ο ADT **ΑκολουθίαμεΔείκτες** αναπαριστά μια μη φυσικά περιορισμένη δομή. Δεν υπάρχει εννοιολογικά κανένα όριο στο μέγεθος που μπορεί να έχει μια ακολουθία. Κατά συνέπεια μια υλοποίηση του ADT χρησιμοποιώντας Java κλάσσεις θα πρέπει να επεκτείνει το μέγεθος της ακολουθίας όσο κάθε φορά απαιτείται απο την εφαρμογή. Προτείνετε τουλάχιστον 2 υλοποιήσεις για την διεπαφή αντικειμένων (interface) Java **ΑκολουθίαμεΔείκτες** (που δημιουργήσατε στο ερώτημα (α)) οι οποίες υποστηρίζουν αυτήν την ευελιξία στο μέγεθος των ακολουθιών. Υλοποιήστε πλήρως μια απο τις προτεινόμενες Java κλάσσεις.

Λύση:

We provide three separate implementations of the RankedSequence ADT:

1. ArrayRankedSeq, which implements the sequence by a dynamically allocated array of objects. This implementation behaves much like the built-in Vector class. Whenever the array used to represent the sequence overflows, it allocates a larger one and copies the previous array into it.
2. DoubleLinkRankedSeq, which implements the sequence by a doubly linked list.
3. VectorRankedSeq, which implements the sequence as an adapter of the Vector class from the Java API. An adapter is an abstraction (e.g., a class) that adapts one abstraction to behave like another. It translates the operations on the interface of one interface into the appropriate operations on a second abstraction. VectorRankedSeq is a class that implements the RankedSequence interface, that is, its instances are instances of the ranked sequence abstraction. However, these operations are implemented directly in terms of similar operations on an instance of the built-in Vector class in the Java API. Thus VectorRankedSeq adapts a Vector to be a RankedSequence.

```
package SoftwareInterfaces;

/**
 * A class that provides a dynamic, array-based implementation of
 * the ranked sequence abstract data type.
 * <p> This implementation uses an array to store the sequence of
 * elements. The rank of an element in the sequence is its index in
 * the array. If the insertion of a new element overflows the
 * allocated size of the array, a new larger array is allocated to
 * store the sequence.
 * @author H. XXXXXXXXX
 * @version 28 June 2001
 */

public class ArrayRankedSeq implements RankedSequence
{
    /* Implementation Invariant:  0 <= seqLength <= theSeq.length &&
     * 0 <= capacityIncr &&  (for 0 <= r < seqLength,
     *                theSeq[r] == element at rank r of the sequence)
     */

    private int seqLength;   // number of elements in the sequence
    private Object[] theSeq; // array to store the sequence elements
    private int capacityIncr;// amount to increment the array size
                             // when expanded
    private boolean debugOn; // debugging option

    private static final int DEFAULT_INITIAL_CAPACITY = 2;
    private static final int DEFAULT_CAPACITY_INCREMENT = 0;
    private static final double CAPACITY_MULTIPLIER = 2.0;

    /**
     * Constructs an empty sequence (with the default initial
     * capacity and incrementing policy).
     * <dt><b>Postcondition:</b>
     * <dd>constructs an empty sequence with initial capacity
     *     <code>DEFAULT_INITIAL_CAPACITY</code> and an incrementing
     *     policy such that the size is doubled on each expansion
     */
```

```
public ArrayRankedSeq()
   {  this(DEFAULT_INITIAL_CAPACITY, DEFAULT_CAPACITY_INCREMENT); }

   /**
    * Constructs an empty sequence with the given initial capacity
    * (and default incrementing policy).
    *
    * @param initialCapacity the number of (empty) slots in the
    * sequence initially
    * <dt><b>Preconditon:</b>
    * <dd><code>initialCapacity >= 0</code>
    * <dt><b>Postcondition:</b>
    * <dd>constructs an empty sequence with initial capacity
    * <code>initialCapacity</code> and an incrementing policy such
    *  that the size is doubled on each expansion
    */
   public ArrayRankedSeq(int initialCapacity)
   {  this(initialCapacity,DEFAULT_CAPACITY_INCREMENT); }

   /**
    * Constructs an empty sequence with the given initial capacity
    * and increment.
    * @param initialCapacity the number of (empty) slots in the
    * sequence initially
    * @param capacityIncrement capacityIncrement if positive, the
    * number of slots to add when the capacity needs to be
    * increased; if zero, double the capacity.
    * <dt><b>Preconditon:</b>
    * <dd><code>initialCapacity >= 0 && capacityIncrement >=
    * 0</code>
    * <dt><b>Postcondition:</b>
    * <dd>constructs an empty sequence with initial capacity
    *     <code>initialCapacity</code> and capacity increment
    *     <code>capacityIncrement</code>
    */
   public ArrayRankedSeq(int initialCapacity,int capacityIncrement)
   {   if (initialCapacity >= 0 && capacityIncrement >= 0) {
           theSeq = new Object[initialCapacity];
           capacityIncr = capacityIncrement;
           seqLength = 0;
           debugOn = false;}
   }

   /**
    * Insert a new element at the given rank into the sequence.
    *
    * @param r the rank at which to insert the object
    * @param e the object to be inserted
    * @throws BoundaryViolationException (e.g., if precondition
    * not satisfied)
    * <dt><b>Preconditon:</b>
    * <dd><code>0 <= r <= theSeq.length</code>
    * <dt><b>Postcondition:</b>
    * <dd>object <code>e</code> added as element at rank
    * <code>r</code>. Ranks of all elements (if any) with old ranks
    *  >= <code>r</code> are increased by one.
    */
   public void insertAtRank(int r, Object o)
                                  throws BoundaryViolationException
  { if (0 <= r && r < theSeq.length) {
        addSlot(r);
        theSeq[r] = o;
        seqLength++;}
    else throw new BoundaryViolationException
  }

   /**
    * Delete the element at the given rank from the sequence.
```

```
 * @param r the rank at which to delete the object
 * @throws BoundaryViolationException (e.g., if precondition
 * not satisfied)
 * <dt><b>Preconditon:</b>
 * <dd> <code> 0 <= r < theSeq.length</code>
 * <dt><b>Postcondition:</b>
 * <dd>element at rank <code>r</code> is removed.
 *     Ranks of all elements (if any) with old ranks >
 *    <code>r</code> are decreased by one.
 */
public void deleteAtRank(int r)
                                throws BoundaryViolationException

{ if (0 <= r && r < theSeq.length) { //thus the element exists
      removeSlot(r);
      seqLength--; }
  else throw new BoundaryViolationException
}

/**
 * Replace the element at the given rank in the sequence.
 * @param r the rank at which to replace the object
 * @param e the replacement object
 * @throws BoundaryViolationException (e.g., if precondition
 * not satisfied)
 * <dt><b>Preconditon:</b>
 * <dd><code>0 <= r < theSeq.length</code>
 * <dt><b>Postcondition:</b>
 * <dd>element at rank <code>r</code> is replaced by object
 * <code>e</code>.
 */
public void replaceAtRank(int r, Object o)
                                throws BoundaryViolationException
{ if (0 <= r && r < theSeq.length) { //thus the element exists
      theSeq[r] = o; }
  else throw new BoundaryViolationException
}

/**
 * Retrieve the element at the given rank from the sequence.
 * @param r the rank whose value is to be returned
 * @throws BoundaryViolationException (e.g., if precondition
 * not satisfied)
 * <dt><b>Preconditon:</b>
 * <dd><code>0 <= r < theSeq.length</code>
 * @return the element at rank <code>r</code>
 *
 */
public Object elementAtRank(int r)
                                throws BoundaryViolationException
{ if (0 <= r && r < theSeq.length) { //thus the element exists
    return theSeq[r]; }
  else throw new BoundaryViolationException
}

/**
 * Determine the length of the sequence.
 * @return the number of elements in the sequence.
 */
public int length()
{   return theSeq.length; }

/**
 * Determine whether the sequence is empty.
 * @return true if the sequence contains no elements;
 *  otherwise false.
 */
public boolean isEmpty()
{   return (theSeq.length == 0); }
```

```
/**
 * Release the unused capacity of the sequence.
 * <dt><b>Postcondition:</b>
 * <dd>frees any unused capacity, making the capacity equal to
 *    the sequence length.
 */
public void trimToLength()
{   if (seqLength < theSeq.length)
    {   theSeq = copyArraySegment(theSeq,0,seqLength); }
}

/**
 * Enables debugging option.
 * <dt><b>Postcondition:</b>
 * <dd>internal debugging option turned on
 */
public void enableDebug()
{  debugOn = true; }

/**
 * Disables debugging option.
 *
 * <dt><b>Postcondition:</b>
 * <dd>internal debugging option turned off
 */
public void disableDebug()
{   debugOn = false; }

/**
 * Internal method that adds a new "empty" array slot at the
 * given rank.
 * @param r the rank at which to add a new slot
 * <dt><b>Preconditon:</b
 * <dd><code>0 <= r <= seqLength</code>
 * <dt><b>Postcondition:</b>
 * <dd>array elements with old indexes in range
 * <code>[r..seqLength)</code> moved one index higher.  A new
 * array slot opened at index <code>r</code>. More capacity is
 * allocated if current array full.
 */
private void addSlot(int rank)
{   if (0 <= rank && rank <= seqLength) {
    if (seqLength < theSeq.length)
    {   for (int i = seqLength ; i > rank; i--)
        {  theSeq[i] = theSeq[i-1]; }
        theSeq[rank] = null;
    }
    else // seqLength == theSeq.length
    {   Object[] temp = new Object[determineNewCapacity()];
        int i;
        for (i = seqLength ; i > rank; i--)
        {   temp[i] = theSeq[i-1]; }
        for (i-- ; i >= 0; i--)
        {   temp[i] = theSeq[i]; }
        theSeq = temp;
    }
  }
}
/**
 * Internal method that determines the desired new size of the
 * array when increasing it is necessary.
 *
 * @return the number of slots needed in the increased size
 * array <dt><b>>Postcondition:</b>
 * <dd>return is <code>capacityIncr</code> if it is positiv;
 *     otherwise it is the current capacity multiplied by
 *     the factor <code>CAPACITY_MULTIPLIER</code>.
 */
```

```
    private int determineNewCapacity()
    {   int incr = Math.max(0,capacityIncr);
        if (capacityIncr == 0)
        {   incr = (int) ((double) theSeq.length
                              * (CAPACITY_MULTIPLIER - 1.0));
            incr = Math.max(1,incr);
        }
        return (theSeq.length + incr);
    }
    /**
     * Internal method that removes the array slot at the given
     * rank.
     * @param r the rank at which to remove the slot
     * <dt><b>Preconditon:</b>
     * <dd><code>0 <= r < length()</code>
     * <dt><b>Postcondition:</b>
     * <dd>array elements with old indexes in range
     *    <code>[r+1..seqLength)</code> are moved one index lower.
     */
    private void removeSlot(int rank)
    {if (0 <= rank && rank < seqLength) {// thus at least one exists
        for (int i = rank ; i < seqLength-1; i++)
        {  theSeq[i] = theSeq[i+1]; }
        theSeq[seqLength-1] = null;}
    }
    /**
     * Internal method that creates a copy of a segment of an array.
     * @param oldArr the array of objects to copy from
     * @param beg the beginning (i.e., smallest) index of the array
     * segment to copy
     * @param len the length of the segment to copy
     * <dt><b>Preconditon:</b>
     * <dd> <code>0 <= beg < oldArr.length && 0 <= len && </code>
     *     <code>beg + len <= oldArr.length</code>
     *
     * @return the new array of size <code>len</code> containing
     * references to elements <code>[beg..beg+len)</code> of
     * <code>oldArr</code>
     */
  private Object[] copyArraySegment(Object[] oldArr,int beg,int len)
    {   if (0 <= beg && beg < oldArr.length &&
                  0 <= len && beg + len <= oldArr.length) {
        Object[] newArr = new Object[len];
        int j = beg;
        for (int i = 0; i < len; i++)
        {   newArr[i] = oldArr[j];
            j++;
        }
        return newArr;}
    }
    /**
     * Debugging method that displays internal information from the
     * instance on the standard output.
     */
    public void DEBUG_showSequence(String msg)
    {   System.out.println("***  BEGIN sequence display:  " + msg);
        System.out.println("Length = " + seqLength + ",  " +
                           "Capacity = " + theSeq.length  + ",  " +
                           "Increment = " + capacityIncr );
        for (int i = 0; i < seqLength; i++)
        {   System.out.println(i + "\t" +
               (theSeq[i] == null ? "NULL" : theSeq[i].toString()));
        }
        System.out.println("***  END sequence display:  " + msg);
    }

}
```

```
package SoftwareInterfaces;

import java.util.*;

/**
 * A class that adapts the <code>java.util.Vector</code> class to
 * implement the ranked sequence abstract data type.
 *
 * @author XXXXXXXXXXX
 * @version 28 June 2001
 */

public class VectorRankedSeq implements RankedSequence
{
    /* Implementation Invariant:(for 0 <= r < theSeq.size(),
     * theSeq.elementAtRank(r) == element at rank r of the sequence)
     */

    private Vector theSeq;    // the Vector to store the sequence
    private boolean debugOn;  // debugging option

    /**
     * Constructs an empty sequence (with the default initial
     * capacity and incrementing policy).
     * <dt><b>Postcondition:</b>
     * <dd> constructs an empty sequence with initial capacity
     *      <code>DEFAULT_INITIAL_CAPACITY</code> and an incrementing
     *       policy such that the size is doubled on each expansion
     */
    public VectorRankedSeq()
    {   theSeq = new Vector(); }

    /**
     * Constructs an empty sequence with the given initial capacity
     * (and default incrementing policy).
     * @param initialCapacity the number of (empty) slots in the
     * sequence initially
     * <dt><b>Preconditon:</b>
     * <dd><code>initialCapacity >= 0</code>
     * <dt><b>Postcondition:</b>
     * <dd>constructs an empty sequence with initial capacity
     *      <code>initialCapacity</code> and an incrementing policy
     *      such that the size is doubled on each expansion
     */
    public VectorRankedSeq(int initialCapacity)
    {  if (initialCapacity >= 0) {;
       theSeq = new Vector(initialCapacity) }
    }

    /**
     * Constructs an empty sequence with the given initial capacity
     * and increment.
     *
     * @param initialCapacity the number of (empty) slots in the
     * sequence initially
     * @param capacityIncrement capacityIncrement if positive, the
     * number of slots to add when the capacity needs to be
     * increased; if zero, double the capacity.
     * <dt><b>Preconditon:</b>
     * <dd><code>initialCapacity >= 0 && capacityIncrement >=
     * 0</code>
     * <dt><b>Postcondition:</b>
     * <dd>constructs an empty sequence with initial capacity
     *      <code>initialCapacity</code> and capacity increment
     *      <code>capacityIncrement</code>
     */
   public VectorRankedSeq(int initialCapacity,int capacityIncrement)
    {   if (initialCapacity >= 0 && capacityIncrement >= 0) {
        theSeq = new Vector(initialCapacity, capacityIncrement); }
```

```
 }


 /**
  * Insert a new element at the given rank into the sequence.
  * @param r the rank at which to insert the object
  * @param e the object to be inserted
  * @throws BoundaryViolationException (e.g., if precondition
  * not satisfied)
  * <dt><b>Preconditon:</b>
  * <dd><code>0 <= r <= theSeq.size()</code>
  * <dt><b>Postcondition:</b>
  * <dd>object <code>e</code> added as element at rank
  *    <code>r</code>. Ranks of all elements (if any) with old
  *     ranks >= <code>r</code>are increased by one.
  */
 public void insertAtRank(int r, Object o)
                              throw new BoundaryViolationException
 { if (0 <= r && r <= theSeq.size()) {
       theSeq.insertElementAt(o,r); }
   else throw new BoundaryViolationException
 }

 /**
  * Delete the element at the given rank from the sequence.
  * @param r the rank at which to delete the object
  * @throws BoundaryViolationException (e.g., if precondition
  * not satisfied)
  * <dt><b>Preconditon:</b>
  * <dd> <code> 0 <= r < theSeq.size()</code>
  * <dt><b>Postcondition:</b>
  * <dd>element at rank <code>r</code> is removed.
  *     Ranks of all elements (if any) with old ranks >
  *     <code>r</code> are decreased by one.
  */
 public void deleteAtRank(int r)
                             throw new BoundaryViolationException
 {   if (0 <= r && r < theSeq.size()) {// thus the element exists
     theSeq.removeElementAt(r);}
   else throw new BoundaryViolationException
}

 /**
  * Replace the element at the given rank in the sequence.
  *
  * @param r the rank at which to replace the object
  * @param e the replacement object
  * @throws BoundaryViolationException (e.g., if precondition
  * not satisfied)
  * <dt><b>Preconditon:</b>
  * <dd><code>0 <= r < theSeq.size()</code>
  * <dt><b>Postcondition:</b>
  * <dd>element at rank <code>r</code> is replaced by object
  * <code>e</code>.
  */
 public void replaceAtRank(int r, Object o)
                             throw new BoundaryViolationException
 { if (0 <= r && r < theSeq.size()) {// thus the element exists
     theSeq.setElementAt(o,r); }
   else throw new BoundaryViolationException
}

 /**
  * Retrieve the element at the given rank from the sequence.
  * @param r the rank whose value is to be returned
 * @throws BoundaryViolationException (e.g., if precondition
  * not satisfied)
  * <dt><b>Preconditon:</b>
  * <dd><code>0 <= r < theSeq.size()</code>
```

```
     * @return the element at rank <code>r</code>
     */
    public Object elementAtRank(int r)
                                throw new BoundaryViolationException
    {   if (0 <= r && r < theSeq.size()){ // thus the element exists
        return theSeq.elementAt(r);}
      else throw new BoundaryViolationException
    }

    /**
     * Determine the length of the sequence.
     * @return the number of elements in the sequence.
     */
    public int length()
    {   return theSeq.size(); }


    /**
     * Determine whether the sequence is empty.
     *
     * @return true if the sequence contains no elements;
     *     otherwise false.
     */
    public boolean isEmpty()
    {   return theSeq.isEmpty(); }

    /**
     * Release the unused capacity of the sequence.
     * <dt><b>Postcondition:</b>
     * <dd>frees any unused capacity, making the capacity equal to
     *     the sequence length.
     */
    public void trimToLength()
    {   theSeq.trimToSize(); }

    /**
     * Enables debugging option.
     * <dt><b>Postcondition:</b>
     * <dd>internal debugging option turned on
     */
    public void enableDebug()
    {  debugOn = true; }

    /**
     * Disables debugging option.
     * <dt><b>Postcondition:</b>
     * <dd>internal debugging option turned off
     */
    public void disableDebug()
    {   debugOn = false; }

    /**
     * Debugging method that displays internal information from the
     * instance on the standard output.
     */
    public void DEBUG_showSequence(String msg)
    {   System.out.println("***  BEGIN sequence display:  " + msg);
        System.out.println("Length = " + theSeq.size() + ",  " +
                           "Capacity = " + theSeq.capacity());
        for (int i = 0; i < theSeq.size(); i++)
        {   System.out.println(i + "\t" + theSeq.elementAt(i)); }
        System.out.println("***  END sequence display:  " + msg);
    }

}
```