Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Οντοκεντρικός Προγραμματισμός
Βασίλης Χριστοφίδης

Τελική Εξέταση (3 ώρες)
Ημερομηνία: 24 Ιουνίου 2003

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (15 μονάδες)

(α) Θεωρήστε το παρακάτω πρόγραμμα Java:

```
/*1 */ class A extends Exception{ }
/*2 */ class B extends Exception{ }
/*3 */ class Main{
/*4 */ public static void main(String[] args){
/*5 */  try{
/*6 */     try{
/*7 */       if(args.length==0) throw new A();
/*8 */       else throw new B();
/*9 */       }
/*10*/     catch(A a){
/*11*/       System.out.println("11");
/*12*/       }
/*13*/     finally{
/*14*/       System.out.println("14");
/*15*/       }
/*16*/     System.out.println("16");
/*17*/     }
/*18*/  catch(B b){
/*19*/     System.out.println("19");
/*20*/     }
/*21*/  System.out.println("21");
/*22*/ }
/*23*/}
```

1. Τι τυπώνεται στην στάνταρ έξοδο του προγράμματος όταν στην εντολή της γραμμής 7 προκύπτει μια εξαίρεση τύπου A? Εξηγήστε σύντομα την απάντησή σας.

Λύση:
It prints 11 14 16 21.
The exception is caught by the inner handler. After that, the control flow continues normally.

2. Τι τυπώνεται στην στάνταρ έξοδο του προγράμματος όταν στην εντολή της γραμμής 8 προκύπτει μια εξαίρεση τύπου B? Εξηγήστε σύντομα την απάντησή σας.

Λύση:
It prints 14 19 21.
The exception propagates to the outer handler; on the way, the inner finally block gets executed. After the outer handler, the control flow continues normally.

(β) Θεωρήστε το παρακάτω πρόγραμμα Java:

```
1 try {
2         <try-block> }
3 finally {
4         <finally-block> }
```

1. Ξαναγράψτε τον παραπάνω κώδικα χρησιμοποιώντας `try-catch` αλλά χωρίς `finally`, υποθέτοντας ότι μόνο η δημιουργία εξαιρέσεων μπορεί να προκαλέσει την απότομη διακοπή εκτέλεσης των εντολών του παραπάνω προγράμματος.

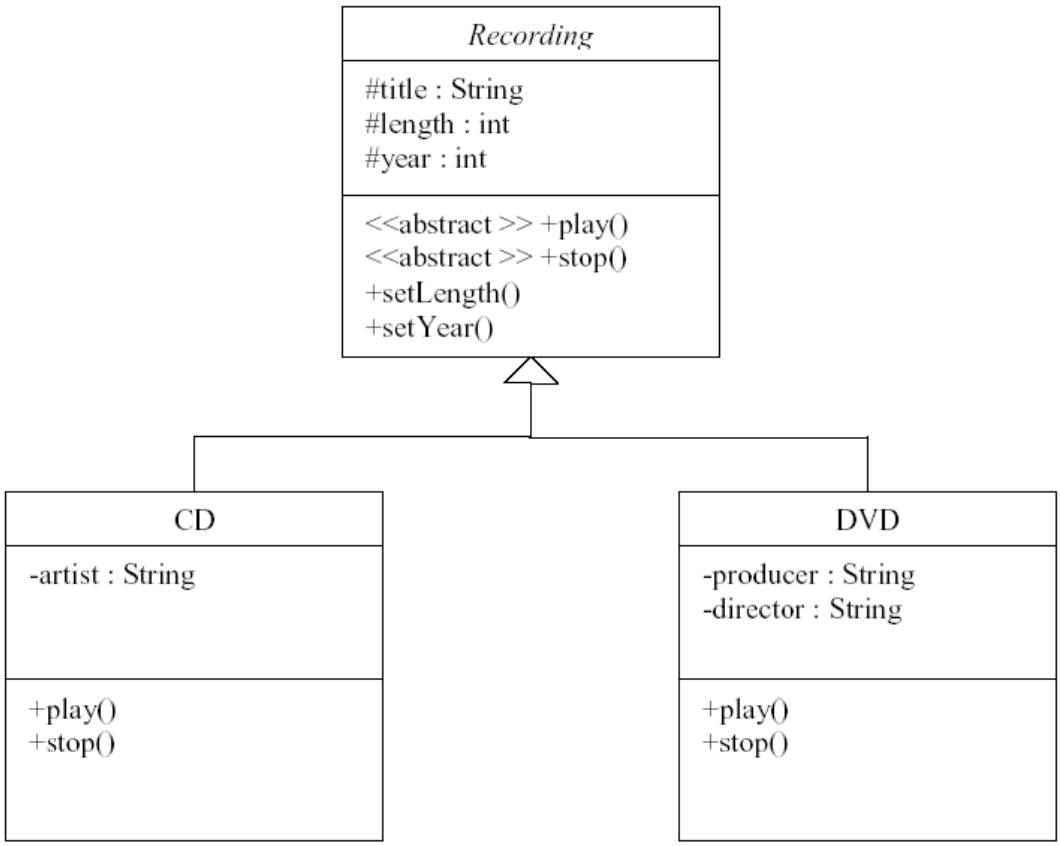
Λύση:
```
try {
   <try-block>}
catch(Throwable e) {
   <finally-block> throw e;}
<finally-block>
```

2. Είναι ο προηγούμενος μετασχηματισμός σωστός όταν επίσης θεωρήσουμε ότι εντολές επιστροφής (`return`) μπορούν επίσης να προκαλέσουν την απότομη διακοπή εκτέλεσης των εντολών του παραπάνω προγράμματος? Αν η απάντησή σας είναι καταφατική εξηγήστε γιατί. Αν η απάντησή σας είναι αρνητική εξηγήστε τους κυριότερους λόγους και προτείνετε έναν τρόπο που εξασφαλίζει την σωστή εκτέλεση του μετασχηματισμένου προγράμματος.

Λύση:
No, it does not, because in Java, a "return" cannot be caught as an exception. One way to make it work would be to rewrite all "return" statements by throws of a special kind of `ReturnException`, and putting a `try-catch` for `ReturnException`s around every call.

Άσκηση 2 (15 μονάδες)

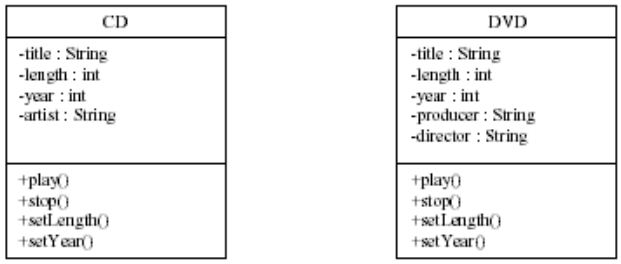
Απαντήστε στις παρακάτω ερωτήσεις χρησιμοποιώντας την ακόλουθη ιεραρχία κλάσεων, όπου η κλάση στην κορυφή της ιεραρχίας έχει τις αφαιρετικές (abstract) μεθόδους `play()` και `stop()`.



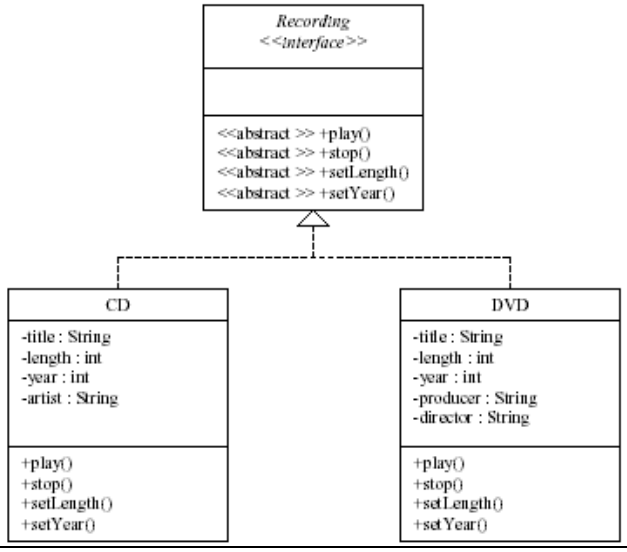
(α) Πώς μπορείτε να υλοποιήσετε την παραπάνω ιεραρχία κλάσεων σε μια γλώσσα προγραμματισμού η οποία δεν υποστηρίζει κληρονομικότητα κλάσεων? Ξανασχεδιάστε το παραπάνω διάγραμμα για την λύση που προτείνετε και εξηγήστε τις διαφορές μεταξύ του αρχικού και του τροποποιημένου διαγράμματος καθώς και τους κυριότερους λόγους για τους οποίους η χρήση της κληρονομικότητας καθιστά ευκολότερη την σχεδίαση και συντήρηση της παραπάνω ιεραρχίας κλάσεων.

Λύση:
There are two valid solutions to this question: 1) have two classes, CD and DVD, that implement everything from the initial system, plus all functionality from the Recording abstract class; 2) change the Recording abstract class to an interface, move all attributes from Recording to the subclasses, make methods setColour() and move() abstract, and implement setColour() and move() in both children classes. The downside of either solution, as compared with the initial solution using inheritance, is that code common to both CD and DVD will have to be duplicated. This is not so much a problem at development-time as it is during maintenance, where changing one of the methods should require changing it in both CD and DVD classes. Corollaries to this are that there is an increased amount of code to read and understand, and the chance of making a mistake/typo increases with the size of the code base. A further downside of the first solution is that dynamic binding will not be possible.

Solution 1:



Solution 2:



(β) Υλοποιήστε σε ένα πρόγραμμα Java την σχεδίαση που σας δόθηκε στην αρχή της άσκησης. Στην απάντησή σας να συμπεριλάβετε μόνο τις μεταβλητές στιγμιοτύπων και τις μεθόδους που αναγράφονται στο διάγραμμα ενώ η υλοποίηση των μεθόδων πρέπει να είναι κενή. Σημειώστε ότι τα σύμβολα πριν από τις μεταβλητές στιγμιοτύπων και τις μεθόδους της κάθε κλάσης υποδεικνύουν τα αντίστοιχα δικαιώματα εξουσιοδοτημένης πρόσβασης: συν (+) για **`public`**, πλην (-) για **`private`** και δίεση (#) για **`protected`**.

Λύση:

```
public abstract class Recording {
 protected String title;
 protected int length, year;
 public abstract void play();
 public abstract void stop();
 public void setLength() { … }
 public void setYear() { … }
}
public class CD extends Recording {
 private String artist;
 public void play() { … }
 public void stop() { … }
}
```

```
public class DVD extends Recording {
 private String producer, director;
 public void play() { … }
 public void stop() { … }
}
```

Άσκηση 3 (10 μονάδες)

Εξετάστε το παρακάτω πρόγραμμα Java:

```
public class Z {
 void show(Z z) {
 System.out.println("Z");
 }
}

public class Y extends Z {
 void show(Y y) {
 System.out.println("Y");
 }
}
```

```
public class X {
public static void
      main(String[] args) {
 Z c = new Z();
 Y b = new Y();
 Z a = new Y();
 c.show(b);
 b.show(c);
 b.show(a);
 a.show(a);
 b.show((Y)a);
 a.show(c);
 b.show(b);
 }
}
```

Τι τυπώνεται στην στάνταρ έξοδο του προγράμματος όταν εκτελέσουμε τον κώδικα της κλάσης **X** ?

Λύση:

The output is:
Z *(2 points)*
Z *(1 point)*
Z *(1 point)*
Z *(1 point)*
Y *(1 point)*
Z *(2 points)*
Y *(2 points)*

A common mistake was identifying the first line as an error by considering that c is of class Z and Z has no method that takes a parameter of class Y. However, since b is of class Y it is also of class Z and therefore the method inside Z fits.

Άσκηση 4 (20 μονάδες)

(α) Δώστε την υλοποίηση της παρακάτω διεπαφής Java η οποία υποστηρίζει (μη-αναδρομικά) κλειδώματα (locks) ανάγνωσης (read) και γραφής (write), με ένα μόνο συγγραφέα ή οποιοδήποτε αριθμό αναγνωστών, οι οποίοι μπορούν να αποκτούν το κλείδωμα (locks) οποιαδήποτε χρονική στιγμή:

```
interface ReadWriteMonitor {
  void acquireReadLock() throws InterruptedException;
  void releaseReadLock();
  void acquireWriteLock() throws InterruptedException;
  void releaseWriteLock();
}
```

Ακριβέστερα, όταν μια προγραμματιστική ίνα (thread) απαιτεί ένα κλείδωμα γραψίματος (write lock), μπορεί να αποκτήσει το κλείδωμα γραψίματος ακόμα και αν άλλες προγραμματιστικές ίνες (threads) απαιτούν συνεχώς κλειδώματα ανάγνωσης (read locks). Με άλλα λόγια, μια προγραμματιστική ίνα δεν μπορεί να περιμένει μέχρι την στιγμή όπου καμία άλλη δεν επιθυμεί να αποκτήσει ένα κλείδωμα ανάγνωσης γιατί αυτή η στιγμή μπορεί να μην έρθει ποτέ.

Λύση:

```
class MyLock implements ReadWriteMonitor {
 int readLocks;
 boolean writeLocked;
 int writeLocksDesired;
 void synchronized acquireReadLock() throws
                                  InterruptedException {
  while (writeLocked||writeLocksDesired != 0) wait();
  readLocks++;
 }
 void synchronized releaseReadLock() {
  readLocks--;
  notifyAll();
 }
 void synchronized acquireWriteLock() throws
                                  InterruptedException {
  writeLocksDesired++;
  try {
    while (writeLocked || readLocks != 0 ) wait();
    writeLocked = true;
    }
  finally {
    writeLocksDesired--;
    }
 }
 void synchronized releaseWriteLock() {
  writeLocked = false;
  notifyAll();
 }
}
```

(β) Δώστε την υλοποίηση της παρακάτω διεπαφής Java:

```
interface TodoList {
 void add(Runnable r);
}
```

Για την υλοποίησή σας θεωρήστε την παρακάτω κλάση **QueueImpl**, η οποία υλοποιεί την παρακάτω διεπαφή:

```
interface Queue {
 void enqueue(Object r);
 Object dequeue();
 int size();
}
```

Η μέθοδος **dequeue**() επιστρέφει **null** όταν δεν υπάρχει κανένα αντικείμενο για απομάκρυνση από την ουρά. Η κλάση **QueueImpl** δεν είναι ασφαλής ως προς την χρήση προγραμματιστικών ινών (not thread safe) δηλ. δεν χρησιμοποιεί τεχνικές συγχρονισμού (synchronization). Η υλοποίηση της ToDoList πρέπει να μπορεί να καλεί τις μεθόδους εκτέλεσης κάθε εργασίας (task) που προστίθεται στην ToDoList, μία μία κάθε χρονική στιγμή, και με την σειρά που έχουν προστεθεί. Εντούτοις, η μέθοδος **add**() πρέπει να μπορεί να τερματίζει την εκτέλεσή της αμέσως μόλις μια εργασία προστεθεί στην ToDoList. Σημειώστε ότι ανά πάσα χρονική στιγμή μόνο μία εργασία μπορεί να εκτελείται.

Λύση:

```
class MyTodoList implements TodoList, Runnable {
 QueueImpl q;
 synchronized public void add(Runnable r) {
   if (q == null) {
        q = new QueueImpl();
        Thread t = new Thread(this);
        t.setDaemon(true);
        t.start();
   }
   q.enqueue(r);
   notifyAll();
  }
 public void run() {
   while (true) {
        Runnable r;
        synchronized(this) {
            while (q.size() <= 0)
            try { wait(); }
            catch (InterruptedException e) {};
        r = q.dequeue();
       };
        r.run();
   }
  }
}
```

Άσκηση 5 (30 μονάδες)

(α) Σχολιάστε τα πλεονεκτήματα και μειονεκτήματα των παρακάτω υλοποιήσεων συλλογών δεδομένων Java.

1. Κάτω από ποιες προϋποθέσεις είναι προτιμότερο ένα διάνυσμα (Vector) από έναν πίνακα (array) για τον χειρισμό μια συλλογής δεδομένων που απαιτεί πρόσβαση με την χρήση δεικτών θέσης (index)?

Λύση:
A *Vector* is preferable to an *Array* under several different circumstances. For example, it is often preferable when the number of elements that will be needed is unknown. This is because it will do the reallocate-and-grow automatically. It is also a better choice if common operations such as `contains()`, `indexOf()`, or `clone()` are required. These could be written for an *Array* -- but they're there and ready to use in a *Vector*.

2. Πότε είναι προτιμότερο ένα διάνυσμα (Vector) από μία απλή ή διπλή συνδεδεμένη λίστα (singly or doubly linked list)? Με άλλα λόγια, ποιες λειτουργίες (operations) είναι πιο αποδοτικές σε ένα διάνυσμα από ότι σε μια συνδεδεμένη λίστα?

Λύση:
Vectors are preferable to *linked lists* any time random access is required. Internally, the elements of vectors are organized in a contiguous chunk of memory, which allows individual elements to be located by address after only a simple computation. Linked lists, by contrast, require step-by-step traversal. Additionally, if the size is known in advance, and memory utilization is a concern, Vectors are preferable to linked lists, even if memory access will be sequential. Vectors require one reference per object, instead of two references per object in a singly linked list (or three per in a doubly linked list). Each element in a doubly linked list contains not only one reference for the data and one reference for the next element -- but also an additional element for the predecessor.
From the perspective of operations, accessing (or mutating) by index is more efficient in an index, as is adding at the end, if the Vector is large enough (doesn't require growth).
On the other hand, doubly linked lists should only be used when the extra reference is worth the space. What does this reference buy us? It makes it possibly to (without recursion or painful loops-within-loops) traverse the list backwards. It also simplifies removing within the list or at the tail. Furthermore, each-and-every mutator that does not require this predecessor pointer is made more complicated, and as a consequence, slower, by its presence. For example, it needs to be updated each time a node is added or removed -- even if it wasn't used to find the node. So, singly linked lists should be used any time it isn't often necessary to traverse the list in reverse, or remove nodes other than at the beginning of the list.

(β) Για κάθε ένα από τα παρακάτω προβλήματα, περιγράψτε ποιος πιστεύετε ότι είναι ο καταλληλότερος ΑΤΔ (Αφαιρετικός Τύπος Δεδομένων) για να σχεδιάσετε (ως προς την λειτουργικότητά της διεπαφής που προσφέρει) και να υλοποιήσετε (ως προς τις επιδόσεις της συγκεκριμένης δομής δεδομένων που χρησιμοποιεί) τις λειτουργίες που σας ζητούνται στην εκφώνηση.

1. Υποθέστε ότι εργάζεστε σε μια εταιρία λογισμικού που σχεδιάζει μια εφαρμογή διαχείρισης του χαρτοφυλακίου μετοχών ενός επενδυτή. Καλείστε να διαλέξετε τον καταλληλότερο ΑΤΔ από το Java Collection Framework για να αναπαραστήσετε ένα χαρτοφυλάκιο δηλ μια συλλογή μετοχών που ανά πάσα στιγμή έχει στην κατοχή του ένας επενδυτής. Η εφαρμογή πρέπει να επιτρέπει την εισαγωγή μιας μετοχής στο χαρτοφυλάκιο όταν αυτή αγοράζεται από έναν επενδυτή, την αφαίρεση μιας μετοχής από το χαρτοφυλάκιο όταν αυτή πωλείται από τον επενδυτή, και την πρόσβαση σε οποιαδήποτε μετοχή του χαρτοφυλακίου όταν ο επενδυτής θέλει να τυπώσει σχετικές πληροφορίες για μια συγκεκριμένη μετοχή.

Λύση: There are a few basic ways to design and implement a solution to this problem: use a Vector or Sequence implemented using either a dynamic array or a linked list.
The List ADT is unacceptable, as it does not provide methods used to access an arbitrary stock in the portfolio. Both the Vector ADT and Sequence ADT contain such methods, and for that reason are acceptable. Either ADT could be implemented using a dynamic array or a linked list. Given this situation, each would provide a similar level of efficiency.
Using a dynamic array, the insert operation and the remove operation would both run in time that is $O(n)$ as the array would have to grow or shrink, respectively, in the worst case. Accessing an arbitrary stock in the portfolio would take $O(1)$ time, as the array can instantly access any of its elements.
Using a linked list, the insert operation would run in $O(1)$ time, as stocks could be inserted at either the first or last position in constant time; the problem does not describe the need to insert stocks at arbitrary locations in the structure. Both removing and accessing an arbitrary stock from the portfolio would take $O(n)$ time, as links must be followed from the head of the list to the stock that is to be removed or accessed.

2. Υποθέστε ότι εργάζεστε σε μια εταιρία λογισμικού που σχεδιάζει ένα διαλογικό σύστημα οδήγησης αυτοκινήτων σε πραγματικό χρόνο. Το σύστημα πρέπει να προσφέρει κατάλληλες βοήθειες οδήγησης στον οδηγό του αυτοκινήτου, και να παρακολουθεί την πορεία του στον δρόμο καθώς ο οδηγός κατευθύνεται προς τον προορισμό του ακολουθώντας πιστά τις βοήθειες που του δίνονται από το σύστημα. Καλείστε να διαλέξετε τον καταλληλότερο ΑΤΔ από το Java Collection Framework για να αναπαραστήσετε μία συλλογή από βοήθειες οδήγησης για να φτάσει ένας οδηγός από την αφετηρία στον προορισμό του. Το σύστημα πρέπει να επιτρέπει την αφαίρεση μιας βοήθειας οδήγησης από την αρχή της συλλογής

όταν αυτή ακολουθείται από τον οδηγό, να προσφέρει την επόμενη βοήθεια μετά από μία συγκεκριμένη βοήθεια οδήγησης, και να ελέγχει εάν υπολείπονται βοήθειες στην συλλογή δηλ εάν ο οδηγός έχει φτάσει τελικά στον προορισμό του.

Λύση:
The most appropriate way to solve this problem would be to use the List ADT. It is possible to use any of the other linear ADTs as well, but the List ADT best fits the description of the problem: remove from the front of the structure, access an element immediately after a given element, and test if the structure is empty.
If the chosen linear ADT is implemented using a linked list, each of the methods could easily run in time that is $O(1)$. For this reason, a linked list is the most appropriate choice. It is possible to implement the chosen linear ADT using a dynamic array and still have each of the methods run in time that is $O(1)$.
If the array is never shrunk as directions are removed from the structure, each of the methods will run in constant time, but memory will be wasted. It is more appropriate to assume that the remove operation would take $O(n)$ time if a dynamic array implementation were used. For that reason, a linked list is a more appropriate choice of implementation.

Ασκηση 6 (30 μονάδες)

Θεωρήστε την παρακάτω διεπαφή Java που καθορίζει το συμβόλαιο του ΑΤΔ **Ουρά με Προτεραιότητα** (Priority Queue ADT):

```
/**
 * Required methods for the Priority Queue ADT.
 */
public interface IPriorityQueue {

 /**
  * Inserts the given key and element into this.
  * @param aKey key for the item
  * @param anElement element for the item
  */
     public void insertItem(Comparable key,
                            Object element);

 /** Returns a highest priority element in this. */
     public Object highest();

 /** Returns a highest priority key in this. */
     public Comparable highestKey();

 /**
  * Removes the highest priority item from this,
  * and returns its element.
  */
     public Object removeHighest();
}
```

Υποθέστε επίσης μια κλάση **Item** της οποίας ο ορισμός περιλαμβάνει δύο μεταβλητές στιγμιοτύπων: μία `public Comparable` με το όνομα **key** και μία `public Object` με το όνομα **element**. Μπορείτε επίσης να υποθέσετε ότι η κλάση **Item** διαθέτει επίσης μία μέθοδο κατασκευής αντικειμένων που παίρνει σαν παραμέτρους εισόδου ένα `Comparable` και ένα `Object`, και αποθηκεύει αυτά τα αντικείμενα αντιστοίχως στις μεταβλητές στιγμιοτύπων **key** και **element** του αντικειμένου **Item** που δημιουργείται.

Δώστε μια κλάση **UnsortedPriorityQueue** που υλοποιεί την διεπαφή **IPriorityQueue** αποθηκεύοντας τα κλειδιά (keys) και τα στοιχεία (elements) της σε ένα μη ταξινομημένο **Vector**. Βεβαιωθείτε ότι η υλοποίησή σας λαμβάνει υπόψη της όλες τις συνθήκες λαθών που προβλέπονται στο συμβόλαιο του ΑΤΔ και δημιουργήστε μια εξαίρεση **RuntimeException** σε κάθε περίπτωση που ένα λάθος εμφανίζεται. Για την ακριβέστερη διόρθωση της άσκησης ενσωματώστε στον κώδικά σας κατάλληλα σχόλια Javadoc.

Λύση:

```
import java.util.Vector;

/**
 * IPriorityQueue implementation using a sorted vector
 * to hold its keys and elements.
 * @author Jeff Raab
 */
public class UnsortedPriorityQueue implements
                                        IPriorityQueue {

 /** Vector used to hold the keys and elements.*/
 protected Vector items = new Vector();

 /** Constructs a new unsorted priority queue. */
 public UnsortedPriorityQueue() {}

 /**
  * Inserts the given key and element into this.
  * @param aKey key for the item
  * @param anElement element for the item
  * @example insertItem(1, A) into {} : {(1, A)}
  * @example insertItem(0, B) into {(1, A)} :
  *                                      {(1, A), (0, B)}
  */
 public void insertItem(Comparable key,Object element){
   items.add(new Item(key, element));
  }

 /**
  * Returns a highest priority element in this.
  * @example highest() on {} throws an exception
  * @example highest() on {(1, A), (0, B)} returns B
  */
```

```
 public Object highest() {
   if (items.isEmpty()) {
         throw new RuntimeException("Queue is empty");
       }
   Object o = items.elementAt(highestIndex());
   Item i = (Item)o;
   return i.element;
 }

 /**
  * Returns a highest priority key in this.
  * @example highestKey() on {} throws an exception
  * @example highestKey() on {(1, A), (0, B)} returns 0
  */
 public Comparable highestKey() {
    if (items.isEmpty()) {
        throw new RuntimeException("Queue is empty");
       }
    Object o = items.elementAt(highestIndex());
    Item i = (Item)o;
    return i.key;
 }

 /**
  * Removes the highest priority item from this,
  * and returns its element.
  * @example removeHighest() on {} throws an exception
  * @example removeHighest() on {(1, A), (0, B)}
  *                                          returns B
  */
 public Object removeHighest() {
    if (items.isEmpty()) {
        throw new RuntimeException("Queue is empty");
       }
    Object o = items.remove(highestIndex());
    Item i = (Item)o;
    return i.element;
 }

 /** Returns the index of the highest key in this. */
  private int highestIndex() {
  int min = 0;
  Item minItem = (Item)items.elementAt(0);
  for (int i = 1; i < items.size(); i++) {
       Item anItem = (Item)items.elementAt(i);
       if (anItem.key.compareTo(minItem.key) < 0) {
           min = i;
           minItem = anItem;
          }
      }
  return min;
  }
}
```