P06 – STORAGE UNIT ORGANIZER

Posted on February 24, 2019 by Mouna AYARI BEN HADJ KACEM

- **SUBMIT your completed assignment by 9:59PM on Wednesday, March 13th 2019.** We will accept and grade submissions made through 9:59PM on Thursday, March 14th 2019 (HARD DEADLINE). But, NO CREDIT will be granted for any work that is submitted even one second after this hard deadline.
- **Pair Programming is NOT allowed for this assignment. You have to work on this assignment INDIVIDUALLY.**

CHANGELOG

IMPORTANT NOTE (03/08/2019)

- The LinkedBoxList can contain boxes having the same weight. The order of boxes having the same weight within the list is equivalent to the order of their insertion into the list. This means that a newBox should be placed after the boxes having the same weight, already inserted into the list.

OVERVIEW

This assignment involves the implementation of a storage unit as a sorted singly linked list. This storage unit list stores boxes of different colors and weights. It has also a capacity that can be easily expanded. These boxes must be sorted in the descending order with respect to their weight, so that the heaviest box should be at the head of the list. A graphic application has been developed to provide you with a good illustration of the different operations of this linked storage unit list.

OBJECTIVES AND GRADING CRITERIA

The goals of this assignment include:

- Implement a storage unit as dynamic sorted Linked List from scratch.
- Further developing your experience working with object oriented design code and graphic applications
- Gain more experience with developing unit tests.

20	Online Tests: these automated grading test results are visible upon uploading your submission. You are allowed multiple opportunities to correct the organization and functionality of your code (if necessary).
20	Offline Tests: these automated grading tests are run after the assignment's deadline has

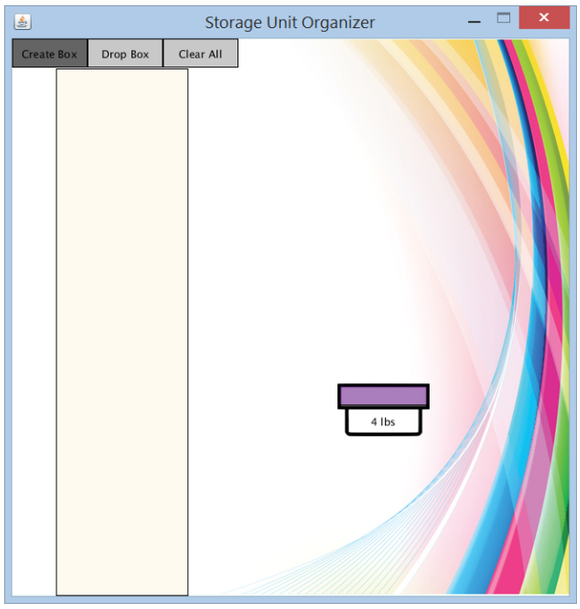
points	passed. They check for similar functionality and organizational correctness as the Online Tests. Since you will not have opportunities to make corrections after seeing the feedback from these tests, you should consider and test the correctness of your own code as thoroughly as possible.
10 points	Code Readability: human graders will review the commenting, style, and organization of your final submission. They will be checking whether it conforms to the requirements of the CS300 Course Style Guide. Since you will not have opportunities to make corrections after seeing the feedback from these graders, you should consider and review the readability of your own code as thoroughly as possible.

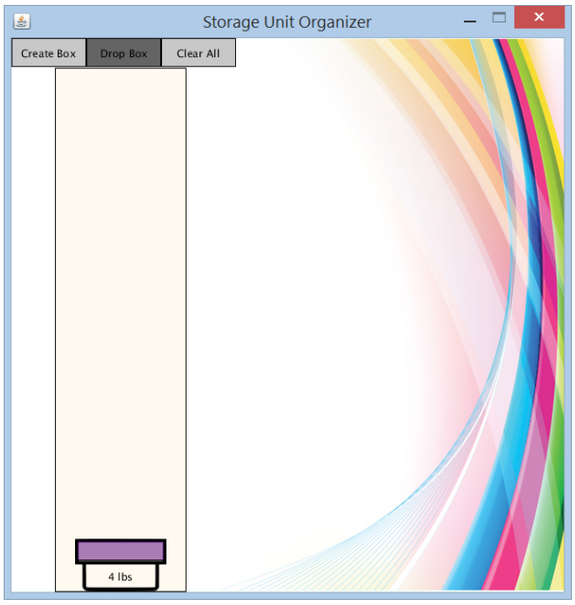
SUBMISSION

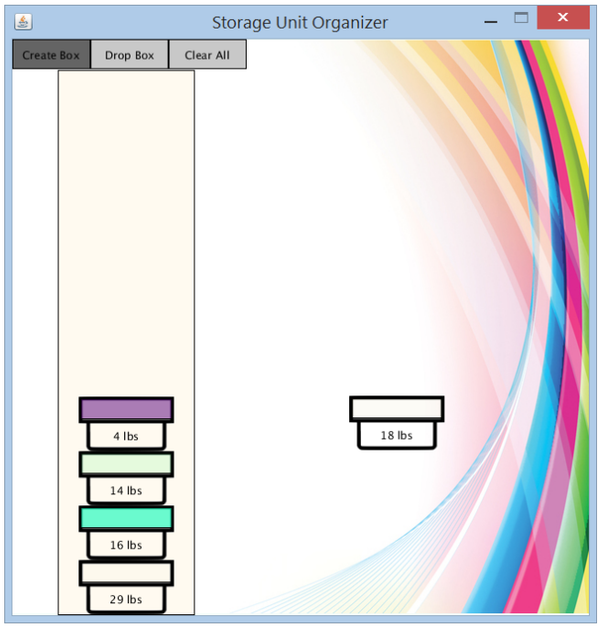
For this assignment, you will need to submit 4 files through zybooks: Box.java, LinkedBoxNode.java, LinkedBoxList.java, and StorageUnitTests.java. You can make as many submissions as you would like prior to the assignment deadline.

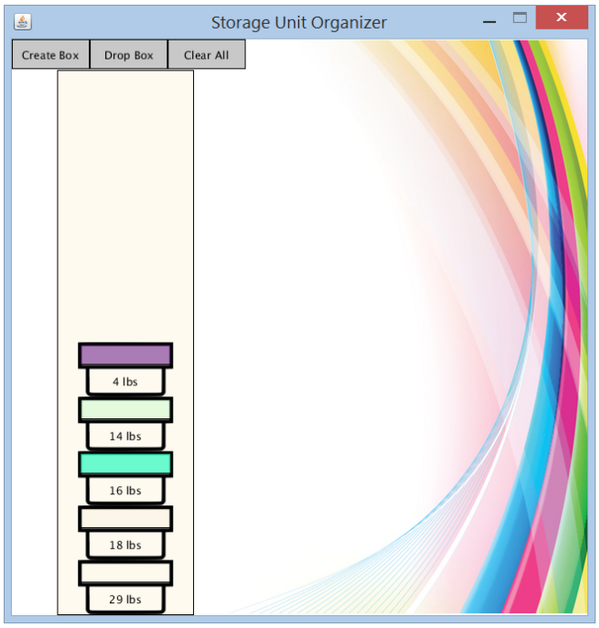
APPLICATION DEMO

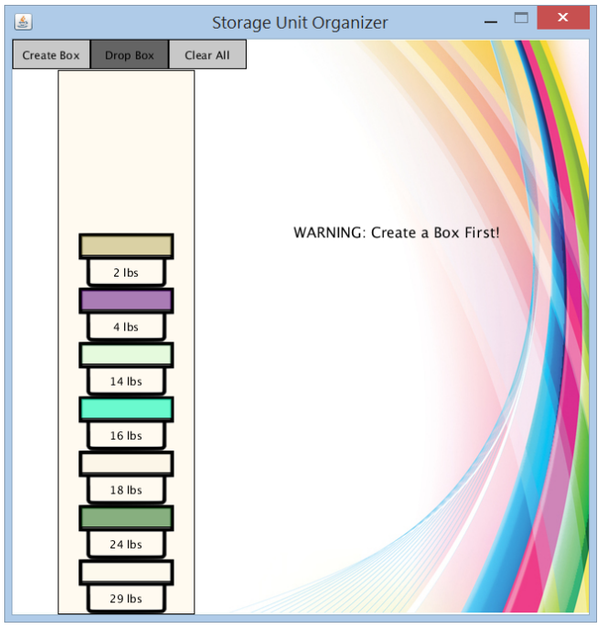
The following is a demo of your program once finalized.

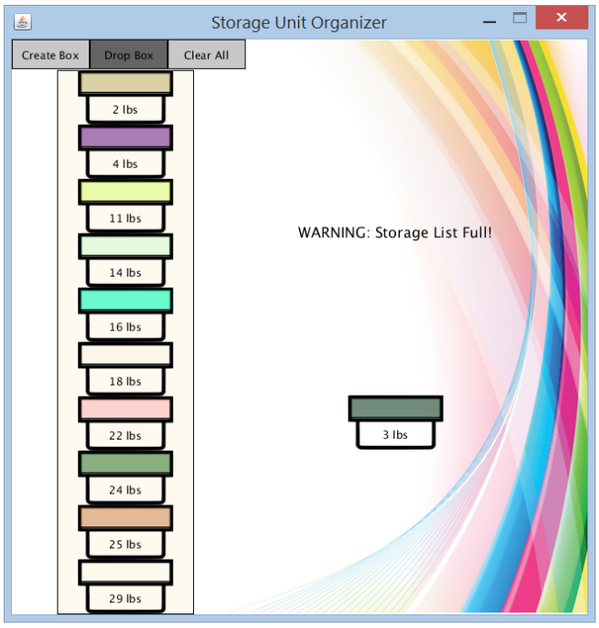












STEP1. CREATE BOX CLASS

Start by creating a new Java Project in eclipse. Recall that you have to ensure that your new project uses Java 8, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-1.8" within the new Java Project dialog box. Then, add a new class called Box that implements the interface Comparable<Box> to this project. This class models a box to be stored in a Storage Unit using our StorageUnitOrganizer application. The specific fields that your Box class MUST define should be exactly the following:

```
1 private static Random randGen = new Random(); // generator of random numbers
2 private int color; // color of this box
3 private int weight; // weight of this box in lbs between 1 inclusive and 31 exclusive
```

As mentioned above, your Box class should define a static field *randGen* that represents a generator of random numbers, and two instance fields that represent respectively the color of the box, and its weight. The weight of this box MUST be between 1 inclusive and 31 exclusive. Whereas, the instance field color can be any integer. It represents a code for a color that can be used, for instance, by PApplet graphic objects and interpreted as a specific color.

This class defines also the following two constructors.

```
// Creates a new Box and initializes its instance fields color and weight to
// random values
public Box() {}

// Creates a new Box and initializes its instance fields color and weight to the
// specified values
// Throws IllegalArgumentException if the provided weight value is out of the
// range of [1..30]
public Box(int color, int weight) {}
```

The no-argument **constructor** creates a new Box with a random color and a random weight. Recall that color can be any integer and the weight must be an integer in the range of 1 inclusive and 31 exclusive.

The overloaded **constructor** creates a new Box and sets its color and weight to the specific given values. It throws an IllegalArgumentException if the provided input parameter weight is out of the range of [1..30].

It is worth noting that you are responsible in this assignment for providing a significant error message to each exception that your code may throw.

Your Box class SHOULD also implement the following methods with exactly the following signatures:

```
@Override
public boolean equals(Object other) { } // equals method defined in Object class

@Override
public int compareTo(Box otherBox) { } // compareTo method defined in Comparable<Box>
                                       // interface

public int getColor() { } // Getter for the instance field color of this box
public int getWeight() { } // Getter for the instance field weight of this box
```

equals() and compareTo() method should be implemented according to the following description:

- equals(Object other) returns true if the specified other object is a Box and this box and other have the same color and same weight. Otherwise, it returns false.
- compareTo(Box otherBox) returns a negative integer, a positive integer, or zero as this box is lighter than, heavier than, or has the same weight as the specified otherBox.

At this stage, we recommend that you design and implement your own test methods to assess the good functioning of the constructor of your class Box and its public methods including the accessors. Create a class called StorageUnitTests and add it to your project. All your test methods must be static, do not take any input parameter, and return a boolean (true if your test method detects a correct behavior of your implemented method(s) and false otherwise). In particular, your StorageUnitTests class must include the following two test methods with exactly the following signatures:

```
// Checks whether the behavior of equals method is correct
public static boolean testBoxEquals(){}

// Checks whether the behavior of compareTo method is correctly implemented
public static boolean testBoxCompareTo(){}
```

STEP2. CREATE LINKEDBOXNODE CLASS

Now, create a new class called LinkedBoxNode. This class models a linked node in our application and should have ONLY the following instance fields, constructors and instance methods (getters and

setters).

```
private Box box; // box that represents the data for this Linked node
private LinkedBoxNode next; // reference to the next Linked Box Node

// constructors
public LinkedBoxNode(Box box) {} // creates a new LinkedBoxNode object with a given
                                 // box and without referring to any next LinkedBoxNode
public LinkedBoxNode(Box box, LinkedBoxNode next){} // creates a new LinkedBoxNode
            // object and sets its instance fields box and next to the specified ones

// getters and setters methods
public LinkedBoxNode getNext() {}
public void setNext(LinkedBoxNode next) {}
public Box getBox() {}
public void setBox(Box box) {}
```

You are responsible for adding adequate test methods to your StorageUnitTests class in order to assess the implementation of your LinkedBoxNode class.

STEP3. CREATE LINKEDBOXLIST CLASS

Now, create a new class called LinkedBoxList. This class models a dynamic list to store box objects sorted in a descendant order with respect to their weights. It is worth to note that you SHOULD NOT use any predefined linked list data structure provided by Java or any other library or source to implement your LinkedBoxList. For instance, you should not use LinkedList or ArrayList classes from the java.util package to implement your LibnkedBoxList. You MUST implement this sorted linked list from scratch.

Your LinkedBoxList class MUST declare ONLY the following instance fields:

```
private LinkedBoxNode head; // head of this LinkedBoxList (refers to the element
                            // stored at index 0 within this list)
private int size; // number of boxes already stored in this list
private int capacity; // capacity of this LinkedBoxList
                      // maximum number of box elements that this LinkedBoxList
                      // can store
```

Your LinkedBoxList class MUST define and implement the following constructor.

```
//Creates an empty LinkedBoxList with a given initial capacity
public LinkedBoxList(int capacity) {}
```

LinkedBoxList models a dynamic sorted singly linked list. We note that capacity is simply a restriction that we impose in this assignment. We'll use later this class to model and simulate a Storage Unit. This latter has generally a capacity (a maximum number of boxes that it can store).

In addition, your LinkedBoxList class MUST implement the following public methods with exactly the following signatures. **It is also worth to note** that you are NOT allowed in this assignment to add any additional instance or static field, or any additional public method, or additional constructor, NOT DEFINED in this write up. You can only add private helper methods that you can call from the pre-defined public methods.

Recall also that LinkedBoxList models a sorted linked list of boxes sorted in the descending order with respect to their weights, so that the head of this list refers to the heaviest box stored within the list. Make sure to consider this property while implementing add() method defined below.

```
// Returns the size of this list
public int size() {}

// Return the capacity of this list
```

```
public int getCapacity() {}

// Expands the capacity of this LinkedBoxList with the specified number a of
// additional elements
public void expandCapacity(int a) {}

// Checks whether this LinkedBoxList is empty
// returns true if this LinkedBoxList is empty, false otherwise
public boolean isEmpty() {}

// Checks whether this LinkedBoxList is full
// Returns true if this list is full, false otherwise
public boolean isFull() {}

// Adds a new box into this sorted list
// Throws IllegalArgumentException if newBox is null
// Throws IllegalStateException if this list is full
public void add(Box newBox) throws IllegalArgumentException, IllegalStateException {}

// Checks if this list contains a box that matches with (equals) a specific box object
// Returns true if this list contains findBox, false otherwise
public boolean contains(Box findBox) {}

// Returns a box stored in this list given its index
// Throws IndexOutOfBoundsException if index is out of the range 0..size-1
public Box get(int index) throws IndexOutOfBoundsException {}

// Removes a returns the box stored at index from this LinkedBoxList
// Throws IndexOutOfBoundsException if index is out of bounds. index should be in
// the range of [0.. size()-1]
public Box remove(int index) throws IndexOutOfBoundsException {}

// Removes all the boxes from this list
public void clear() {}

// Returns a String representation of the state and content of this LinkedBoxList
// An example of source code for this method is provided you in the next paragraph
@Override
public String toString() {}
```

In order to be able at this stage to display the content of a LinkedBoxList object in text mode, we provide you with an example of an implementation for the overridden toString() method. Feel free to use this code in your program. Feel free also to come up with any other String representation for your LinkedBoxList that makes sense.

```
  /**
   * Returns a String representation for this LinkedBoxList
   */
  @Override
  public String toString() {
    StringBuilder result = new StringBuilder(); // creates a StringBuilder object
    String newLine = System.getProperty("line.separator");
    result.append("-------------------------------------------------"+newLine);
    if (!isEmpty()) {
      LinkedBoxNode runner = head;
      int index = 0;
      // traverse the list and add a String representation for each box
      while (runner != null) {
        result.insert(0,
            "Box at index " + index + ": " + runner.getBox().getWeight() + " lbs" + newLine);
        runner = runner.getNext();
        index++;
      }
      result.insert(0, "Box List Content:"+ newLine);
    }
    result.insert(0,"List size: " + size + " box(es)." + newLine);
    result.insert(0,"Box List is empty: " + isEmpty() + newLine);
    result.insert(0,"-------------------------------------------------"+ newLine);
    return result.toString();
  }
}
```

Recall that your are responsible for checking the good functioning of the different methods that you have implemented in this assignment. In particular, your StorageUnitTests should include the following test method with exactly the following signature.

```
// Checks whether remove method defined in your LinkedBoxList works correctly
```

```
2 public static boolean testLinkedBoxListRemove() {}
```

All in all, your submitted StorageUnitTests class MUST contain at least FIVE test methods.

Note also that you should not add an instance getter method for the head instance field defined in your LinkedBoxList class. If you need to display the content of your sorted linked list from your test methods, you can call your toString() method overridden in your LinkedBoxList class.

STEP4. GRAPHIC USER APPLICATION SETUP – STORAGE UNIT ORGANIZER

Finally, after testing thoroughly the good functioning of your implemented classes and their defined behaviors, let's visualize the execution of your program using a graphic PApplet application called StorageUnitOrganizer.

Now, download this StorageUnitOrganizerGraphics.zip file, and extract the contents of this archive file directly into your P6 project folder. This should add a core.jar file, and a folder of images to your project folder. The core.jar file is part of the processing graphical library. Note that if this .jar file does not immediately appear in your eclipse Project Explorer, try right-clicking your project in the project folder and selecting "Refresh" to fix that. To make use of the code within this jar file, you'll need to right-click on it within the Project Explorer and choose "Add to Build Path" from the "Build Path" menu.

Next, download this file srcStorageUnitGraphic.zip and extract its contents into the source folder (src) of your P6 project folder. This archive includes the source java files for GraphicBox, Button, CreateBoxButton, DropBoxButton, ClearButton, and StorageUnitOrganizer classes that model and implement the graphic environment for this StorageUnitOrganizer application. StorageUnitOrganizer class represents the main class for your graphic application. It extends PApplet class defined in processing graphic library. GraphicBox extends your Box class so that a box can be drawn a visible graphic object in the provided StorageUnitOrganizer graphic application. Button class is the super class for any button that models and implements the common properties and behaviors of any button that can be defined in this application. CreateBoxButton, DropBoxButton, and ClearButton classes extend Button class, has different labels, and implement different behaviors when they are clicked (when mouse pressed and is over the button).

Make sure to read through all the provided code and understand it. We recall that NONE of the provided code in this section should be submitted to zybooks. You can use this code to have a graphic illustration of the use of your implementation for this sorted singly linked data structure LinkedBoxList.

Now, you can run your StorageUnitOrganizer graphic application that uses your implemented LinkedBoxList class. Running the StrorageUnitOrganizer main method should result in an interactive graphic user interface comparable to the sample shown at the top of the write-up. Here are a few tips on how to use this application.

A user has to create a new Box (by clicking on "Create Box" button), and then drop it into the Storage Unit (by clicking "Drop Box"). If the user tries to click on "Drop Box" without creating the new Box to be dropped first, the error message within the IllegalArgumentException thrown from the call LinkedBoxList.add() with a null argument will be displayed to the display window.

The Storage Unit (storageList of type LinkedBoxList) related to this graphic application has a capacity of 10. If the list reaches its capacity, and the user creates a new box and tries to drop it into the storageList, the error message within the thrown IllegalStateException will be displayed to the display window.

To remove a box from the list, move the mouse over it and press the R-key. If your LinkedBoxList.remove() method is correctly implemented, the box will be removed from the storage unit and a descriptive message will be displayed to the display window.

SUBMISSION

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the course style guide, you should submit your final work through zybooks. The most recent of your highest scoring submissions prior to the deadline of **9:59PM on Thursday March 14th (HARD DEADLINE)** will be recorded as your zybooks test score. NO CREDIT will be granted for any work that is submitted even one second after this hard deadline. The second portion of your grade for this assignment will be determined by running that same submission against additional automated grading tests after the submission deadline. Finally, the third portion of your grade for your P06 submission will be determined by humans looking for organization, clarity, commenting, and adherence to the course style guide.

EXTRA CHALLENGES

Here are some suggestions for interesting ways to extend this simulation, after you have completed, backed up, and submitted the graded portion of this assignment. **No extra credit will be awarded for implementing these features**, but they should provide you with some valuable practice and experience. DO NOT submit such extensions using zyBooks.

- You can try to extend the graphic application StorageUnitOrganizer to simulate your LinkedBoxList.get() method. For instance, you can implement the following behavior. When the user presses G-key and the mouse is over a graphic box, your LinkedBoxList.get() method will be called. The user can get that specific box without removing it from the list.
- You can add “save” button to save a description of the content of the storage list conforming to a given text format with your choice.
- You can add “load” button to load a text file that stores a representation of the content of the Storage Unit. The behavior of your load button when clicked, should appropriately read and load an already saved file.
- You can also extend the graphic simulation of remove (and get) method such that when the box is removed or gotten, a particle fountain with the same color of the box will be displayed.