# Assignment 2

COMP 250 - Winter 2018

Due date : March 2nd, 2018

## 1 General Instructions (read carefully)

- Office hours are displayed on the calendar on the webpage of the course.
- You are provided some starter code that you should fill in as requested. **Add your code only where you are instructed to do so**. You can add some helper methods. Do not modify the code in any other way and in particular : do not change the methods or constructors that are already given to you, do not import extra code and do not touch the method headers. **Any failure to comply with these rules will give you an automatic 0**.
- **Submission instructions**
  - Assignment is due on March 2nd 11:59 PM.
  - Late assignments will be accepted up to 3 days late and will be penalized by 20 points per day. Note that submitting one minute late is the same as submitting 23 hours late.
  - Don't worry if you realize that you made a mistake after you submitted : you can submit multiple times but only the latest submission will be evaluated. We encourage you to submit a first version a few days before the deadline (computer crashes do happen and myCourses may be overloaded during rush hours).
  - Submit a .zip containing only the files `Warehouse.java` and `Shelf.java` to the myCourses Assignment 1 folder.
- The starter code includes a tester class. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will test your code on a more challenging set of examples. We therefore highly encourage you to

modify that tester class, expand it and share it with other students on the myCourses discussion board. Do not include it in your submission. However, your code will only be tested on valid inputs, for instance, `10200011` is not a valid input if the method requires a number in base 2. The tester file is only here to help you, the reference is the subject of the assignment.

- You will automatically get 0 if your code does not compile.
- **Failure to comply with any of those rules will be penalized.** If anything is unclear, it is up to you to clarify it by asking either directly a TA during office hours, or on the discussion board on myCourses.

# 2 Amazon

Congratulations, you have just been hired by Amazon ! They would like to automate their warehouses and are asking you to code the basic outline they will be using.

**Automation in real-life** There was automation far before computers were invented. For instance, John Kay invented the flying shuttle (he received the patent for his invention in 1733) that enabled the automation of looms (used to weave cloths). This was one of the great contributors to the industrial revolution. Nowadays, automation is expanding very quickly. Some striking examples include an almost fully automated plant in the delivery industry in Tianjin (China), receptionists who are replaced with robots or screens or the treatment of information that is done by facebook, google etc (or the grading of this course).

We won't be discussing here the societal changes (both positive — automation of tedious or dangerous tasks — or negative — removal of jobs) that will come with these changes.

**Structure of the code that is provided to you** You have five files that are provided to you.

There are four classes.

First there are `Box` and its subclass `UrgentBox`. A `Box` has a unique identifier, a height and a length. They also have two fields `next` and `previous` that will be useful to construct linked lists.

A `Shelf` is where you store `Boxes`. A `Shelf` has a length and a height. You can only store a `Box` on a `Shelf` if it fits the remaining space (in height and in length). `Boxes` are stored in a two-way linked list.

Finally, a `Warehouse` contains an array of `Shelf` and two linked lists which correspond to `Boxes` and `Urgentboxes` that need to be shipped. We assume that a `Warehouse` contains at least one `Shelf`.

The class `Warehouse` contains two methods `print` and `printShipping` that are useful to test your code. We have only provided a very scarce testing framework in `Tester.java` and you should expand it.

**Sort storage** The `height` field is really important : you cannot put a box that is 10 high into a spot that is 5 high. But at the same time, you do not want to put a box that is 10 high onto a shelf that is 30 high if there is some space left on a shelf 15 high. To make it simpler to place a box on the optimal shelf, you are first going to organize (sort) the shelves by increasing order of height.

**Question 1.** Implement merge sort in `Warehouse` to sort all shelves by increasing order of height. You will have to code `mergeSort` (divide part of merge sort) and `merge` (merging part). You can refer yourself to the slides of lecture 9 for the detailed algorithm (be careful, the indices in an array start at 0 in java). You can assume that the maximal height of a `Shelf` is 1000.

For the other methods that you are going to implement, you can assume that `storage` is already sorted.

**Add Boxes** Your warehouse is now ready ! You are starting to get boxes. You need to put them on the shelves : each box on its optimal shelf. This is what you are going to code now : first how to add a box on a shelf and second decide which shelf from the warehouse to put the box onto (and do it).

**Question 2.** Implement `addBox` in `Shelf`. Here you can assume that the box given in argument fits both in height and length so you don't need to check it in this function. You will have to add it to the last position on the `Shelf`.

**Question 3.** Implement `addBox` in `Warehouse` : find the smallest shelf with enough space available for the box given in argument, add the box on this shelf and return the `noProblem` message. If there is no such shelf, do not do anything to the warehouse and return the `problem` message.

**Shipping Boxes** Ultimately, you want to send to each customer what he ordered from Amazon. You are going to implement this part now. First, you are going to implement the method that removes a box from the shelf it was on, then you are going to implement an auxiliary method that puts a box into the appropriate shipping list and finally you are going to implement the method that finds the box with a given id, removes it from its shelf and put it into the corresponding shipping list.

**Question 4.** Implement `removeBox` in `Shelf`. The argument is the id String of a box. If you can find a `Box` with that `identifier`, remove it from the `Shelf` and return the box you have found. Do not forget to update all the fields that need to be. If there is no `Box` on that `Shelf`, return `null`.

**Question 5.** Implement `addToShip` in `Warehouse`. The argument given is a `Box`. If that box is an `Urgentbox`, it is added to the beginning of the toShipUrgently list. Otherwise it is added to the beginning of the toShip list. Do not forget to update all the fields that need to be. The method returns noProblem if everything was done successfully, problem otherwise.

The method addToShip is an auxiliary method for the method shipBox that you will code next.

**Question 6.** Implement shipBox in Warehouse. The argument given is the identifier of a Box. If a Box with such identifier exists, remove it from its corresponding Shelf and add it to its corresponding shipping list, then return noProblem. If no such Box exists in the entire warehouse, return Problem.

**Moving Boxes around** Your warehouse is now fully functional : each day you receive and ship boxes. But the placement of the boxes may become non optimal. Let us see why on an example : there are two shelves with height 15 and 20 respectively and length 10 for both. You receive a first box of height 15 and length 10. You place it on the first shelf (of height 15). You now receive a second box of height 15 and length 10, so you place it on the second shelf. And then you ship the first box. So the first shelf is empty and the second shelf has the box of height 15. Now if you receive a third box of height 16, you have to send it back saying “sorry, no room left”. But if you had moved the second box on the first shelf, you could have stored the third box on the last shelf !

You are going to implement this reorganization of the warehouse, first for a single box, then for the entire warehouse.

**Question 7.** Implement moveOneBox in Warehouse. The arguments are a Box and a number of Shelf corresponding to the Shelf on which the Box sits. If there is a Shelf with a smaller height where the Box would fit, move the Box to the end of that new Shelf (and don’t forget to update all the fields that need to be)

**Question 8.** Implement reorganize in Warehouse. Start with the lower Shelf and the first Box on this Shelf and go on until the last Box on the last Shelf. You can optimize this procedure, but this is the order we are looking for.