Data Structures and Algorithms Spring '23 section1 Dr. Hristescu

Lab 5

Assigned: Thursday, February 16 Due: 10:00pm Monday, February 20

Problem0 (Prelab):
Using the generic StackInterface [and StackException], implement it as a **generic** class using a) **Resizable Array**-based implementation (**StackRA**- data fields **items** array and **top** integer**:** index of the top item in the Stack and initial size of the array to be 3) and b) **Simply Linked Structure**-based implementation (**StackSLS** – only data field **top:** reference to the node containing the top item in the Stack) using the generic Node class and test them both thoroughly using a menu driven test program with the options below on your own input data. **DO NOT add any additional functionality to the stack than the one specified in StackInterface.** Have **toString** collect the items starting with the top item and ending with the one at the bottom or the stack.

0. Exit
1. Push item onto the stack.
2. Pop item from the stack.
3. Display top item in the stack.
4. Display items in the stack.
5. Clear the stack.

Note: More on generics on page 291 of the textbook and **in Ch 9.3 (Java Generics) starting on pg 475**.

- When using an array, use the following (separately or combined) which will cause a compiler warning (not error!), which you can disregard.
  T [] arrayvariable; //declaration
  arrayvariable = (T[]) new Object[desiredlength]; //memory allocation for array and assignment to array variable
- When declaring and creating an instance of a generic class, the appropriate type parameter(s) has to be instantiated. Ex:
  StackRA<String> stack1= new StackRA<String>();
  StackSLS<Elephant> stack2= new StackSLS<Elephant>();

Problem1: Follow these instructions for the driver developed for Problem 0 to take the name of the class implementing the StackInterface as command line argument.

Problem 2: You are modelling a delivery business, picking up and delivering orders. You have a narrow delivery bag that can fit small packages piled on top of one another and an even narrower samples bag that can hold samples that you may receive as a thank you upon each delivery. Therefore, both bags work in last-in-first-out (LIFO) order. Packages can contain several items of the same type. The program should support the options below. Here are the descriptions for the options that are not self-explanatory.
Option 1: Pick up an order: User specifies package information in this order (name of item, weight of each item, number of items, whether this is a healthy item or not, sender and receiver). The package will be placed on top of the existing packages in the bag.
Option 2: Drop off an order: The last package that had been picked up will be delivered to the appropriate receiver. Upon delivery, **only if the package contains a healthy item**, the receiver will be asked if they allow you to keep one of the items as a tip. If affirmative, the item is placed on top of all other samples in the bag of samples.
Option 5: Enjoy an item from the bag of samples: The sample item that was placed in the bag of samples last will be removed and enjoyed.
Menu options:

0. Exit Program
1. Pick up an order
2. Drop off an order
3. Display the number of packages, the weight of the delivery bag, and its content
4. Display the number of samples, the weight of the samples bag, and its content
5. Enjoy an item from the samples bag
6. Enjoy all the samples in the samples bag

A sample run on this data is here.
Test your Problem 2 driver with both StackRA and StackSLS. If you did the Extra Credit, the decision about which implementation to use does not need to be hardcoded into the driver but is supplied at runtime as a command line argument. For Problem 2 you have to instantiate the type for the 2 stacks. Please make sure that they are not the same type. I suggest you use inheritance to ensure that there is no code duplication. I suggest to use **Sample** and **Package** as the type names and these types to instantiate the type of the item for the sample and the delivery bag respectively.
Set things up as follows:

- **Sample** class with *name* and *weight* d.f. as well as supporting functionality including a *getWeight* and *getUnitWeight* and *toString* method.
- **Package** class to extend **Sample**, with *amount*, *healthy*, *sender*, *receiver* d.f. as well as supporting functionality pertaining to these d.f.s and making sure that inherited *getWeight* method is overwritten to correctly compute the

package weight rather than the inherited sample weight.

- generic **Bag**<T> class with *collection*, *units* and *totalweight* d.f., where collection is of **StackInterface<T>** type, units keeps track of the number of units (samples or packages) in the bag and totalweight keeps track of the total weight of the bag; **Attach the appropriate bag functionality including a method named getTotalWeight**. When using the item functionality, use casting to **Sample**, ex: ((Sample)item).getWeight();
- Your driver should use for this application a **Bag<Package>** for the delivery bag and a **Bag<Sample>** for the samples bag.
- Make sure the driver handles the displaying of the bag content by using the Sample and Package toString methods

Note:

- **DO NOT enhance the functionality of the stack.** Everything that is specific to the application should be handled in the driver.
- Since the Stack is a generic class, make sure you **use the appropriate data type** for each collection that you are using. Do not use more memory than necessary (i.e. don't use types whose data fields are not used).

Problem 3:
Make one (or all) of the List ADT implementations from Labs 2,3,4 into generic classes.

Problem4:
Lab5Conclusions: Summarize in a few sentences what you have learned from working on this lab.

Submit: Follow the instructions to turn in the lab electronically using **Lab5** for the name of your directory.
P0: code for StackRA and StackSLS implementations of StackInterface, just **one** test driver and sample runs on you own input suite.
P1: all the code developed including driver program (just **one** driver), sample runs.
P2: Submit Driver using command line argument for stack implementation, sample runs (including on provided input).
P3: code for the generic List implementation(s).
P4: Conclusions