NAVIGATION

Home
- My home
- My profile
- Current course
  - CS 2114 Spr 2016
    - Participants
    - February 8 - February 14
      - Reading Quiz 4 2016 Spring
      - OpenDSA HW Assignment 2, Algorithm Analysis
      - Lab 04: Grocery Bags
      - Lab 04 Text File
      - CS2114_Lecture(
      - CS2114 Lecture05PartB Bags LinkedImplemen
      - CS2114 Lecture 06 Algorithm Efficiency File_2016S
      - Practice Problems 2/8
      - Project 2: Whack A Shape
- My courses

SETTINGS

Course administration

My profile settings

PROJECT 2: WHACK A SHAPE

Project 2: Whack A Shape

Due Date: Monday, Feb 22. 8am.

Introduction

In this project you will learn how to extend and implement the rudimentary methods of a `SimpleLinkedBag` and a `SimpleArrayBag`. Both of these classes will implement the `SimpleBagInterface<T>`. You will then use these methods to create a very familiar game, where you have a `Window` with Shapes which disappear when clicked. These Shapes will either be the `CircleShape` or `SquareShape` classes. These Shape instances will be stored in your `SimpleBagInterface<Shape>` and will be predetermined by input strings. These input strings will be one of the following:

- red circle
- blue circle
- red square
- blue square

Your job will be to convert these strings into shapes and add them to your bag. When it comes time to play the game in a `Window`, you will use the `pick()` method to randomly select one shape to display on the screen without removing it. Each Shape will have an `onClick()` method defined so you can listen when the user clicks on them, just like a `Button` in Project 1. When clicked, you will remove the shape from the `Window`, take it out of the Bag, and display another random shape from the bag until the bag is empty. Once the bag is empty, you will display a `TextShape` saying, “You Win!” The Window will have a Quit button on the North side of the window which will quit the program and end the game.

Remember: compile and run frequently (like project 1). Submit to Web-CAT only when you are stuck or finished in order to make everyone’s wait time shorter. These instructions will give you guidance but will not indicate every step, you will become an increasingly independent programmer.

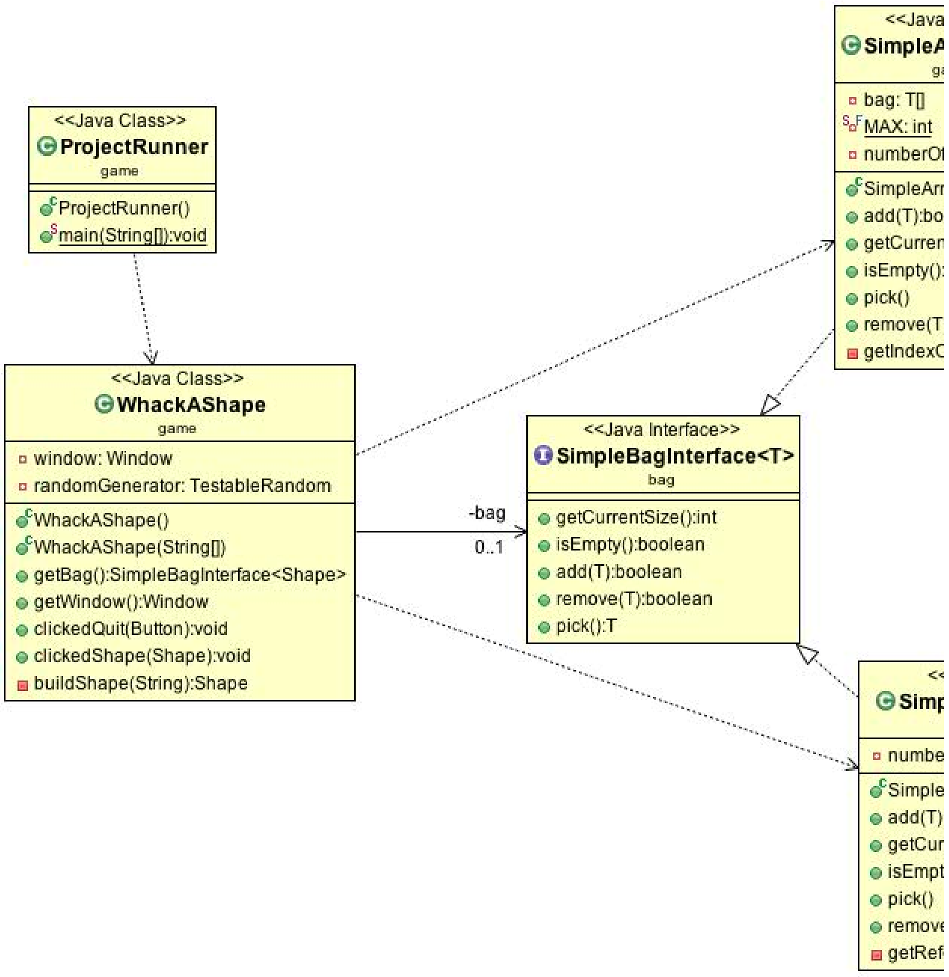
Unified Modeling Language Diagram (UML)

The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system. From here on out, we provide you with a UML of the project to get an idea of what methods go where, and how classes interact.

Here is a legend of symbols used in these UML Diagrams which appear on the left side of the fields or methods:

- red box icon = private visibility
- green circle icon = public visibility
- blue triangle icon = default (package) visibility
- superscript S icon = static
- superscript F icon = final
- superscript C icon = constructor

In `WhackAShape`, you have two configuration options: you can pick whether to use a `SimpleArrayBag<Shape>` or a `SimpleLinkedBag<Shape>` to store your Shapes. Whichever you chose won't influence the functionality of your program, since they implement the same interface!



Make a new project

Make a new project named `WhackAShape`, with the package `game`. Remember you are required to implement the public (or package) methods and fields shown in the UML diagram, the private ones are your design choice. Notice in the above UML diagram some of the fields are shown with the multiplicity of the aggregation(or has-a) relationship. For example, `WhackAShape` has bag field of type `SimpleBagInterface`.

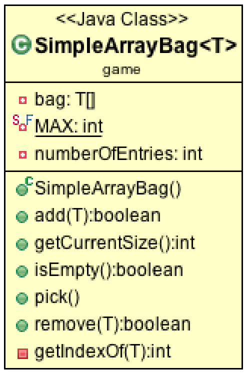
You will need to configure your BuildPath to include CarranoDataStructuresLib, GraphWindowLib and CS2114-Support Projects. If you have both of those projects downloaded and open, this can be done by right clicking on your `WhackaShape` project in the Package Explorer. "Select Build Path" > "Configure Build Path..." and then choose "Projects" and press the "Add" button to choose the two required projects.

The next task is to create 2 classes that implement an interface. However when you are creating these classes you can make use of Eclipse to generate the method stubs for these classes. For those of you who do not know what a method stub is, it is basically a skeleton.

Now create two new classes, `SimpleArrayBag<T>` and `SimpleLinkedBag<T>`. When you are adding the class name in the popup menu for new class creation in Eclipse,notice the interfaces block below the name. You should add the `SimpleBagInterface <T>` from here. Keep in mind that you are not creating a new Interface here but just adding an interface from Carrano Data Structures to the classes you are creating. Once you have added the interface and clicked the Finish button, Eclipse will generate an error for your class. Click on the error icon and Eclipse will provide you with options to fix the errors. Click on "Add unimplemented methods", to get rid of the errors and to auto-generate the method stubs for your class.

To start, we need to fully implement both `SimpleArrayBag` and `SimpleLinkedBag`. Reference your class notes for very similar implementations.

SimpleArrayBag



Refer to the UML diagram to declare instance variables and set the constant MAX = 25. These method implementations are similar to the Carrano examples in the text and class lectures. Remember the UML lists the return type for each method.

SimpleArrayBag()

Initialize the fields in the constructor. Your `bag` field needs to be initialized to a new array of T objects of size MAX, and the `numberOfEntries` field needs to be 0. Use the following code to initialize your bag:

```
@SuppressWarnings("unchecked")
    T[] tempbag = (T[]) new Object[MAX];
    bag = tempbag;
```

getCurrentSize() && isEmpty()

Implement these yourself.

add(T anEntry)

To see if we can add, check if `numberOfEntries` or the length of the bag is less than 25. – If it is not, return false – If it is, place anEntry in our `bag` at the index given by `numberOfEntries`. Increment `numberOfEntries` and return true. – Also think about the case if anEntry is null! In this case it should not add anything and return false.

pick()

In pick, we will use the `TestableRandom` class to generate random numbers. Import student.TestableRandom to use this class.

```
TestableRandom generator = new TestableRandom();
int index = generator.nextInt(10);
```

The `nextInt()` method will randomly generate a number ranging from zero to one less than the numer specified. The example above will randomly create 0-9, or 10 options. All have an equal chance of occuring.

If `isEmpty()` returns true, return null. This means there is nothing to draw from the bag.

If it's false, generate a Random number named index where `numberOfEntries` is the bound. Return the entry at that index in our bag.

getIndexOf(T anEntry)

Write this private method that is a helper method for `remove()`. For every entry in our bag, check if it matches anEntry using the `equals()` method. If it does, return the index at which you found it. Otherwise, if we reach the end of the bag and we still have not found it, return -1.

remove(T anEntry)

Call `getIndexOf()` with `anEntry` to check if this ArrayBag contains the entry to remove.

If `getIndexOf()` returns -1, return false. This means we failed to remove the entry. Otherwise, to remove anEntry, we need to set the entry at the index given to null. However, we don't want to have any holes in our array. We need to move all the data as far to the left as possible.

To fill in the hole, place the last entry in our bag at the index which `getIndexOf()` returned. The last entry in the bag is at the index of numberOfEntries – 1.

Don't forget to set the original last entry's location to null, then decrement `numberOfEntries`. Return true.

SimpleArrayBagTest

To ensure that you implemented your ArrayBag's methods as you expect, write a test method for each public method you've created. Make sure that these tests also exercise the code in any private methods. When all of your tests pass, you can move on to your `SimpleLinkedBag` implementation or make your `WhackaShape` game. The order is up to you.

Hint: Testing TestableRandom

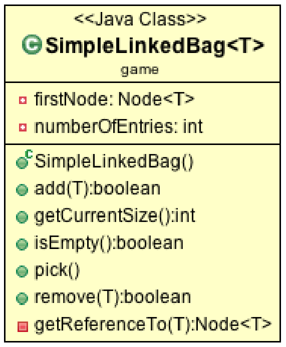
We have you use the provided `TestableRandom` class because you can specify its randomly generated numbers. Having the number specified allows you to repeat the test case. Here's how you can specify the numbers...

```
TestableRandom.setNextInts(0, 42, 7);
```

That line of code predetermines any randomly generated numbers will now be 0, 42, 7, then resume normal behavior. Be sure to set these values before you try to test code which uses the `TestableRandom` class.

Reference Student API for Javadocs on TestableRandom as needed.

SimpleLinkedBag



We need to implement and test all of SimpleLinkedBag's methods before we can use them in our game. These method implementations are similar to the Carrano examples in the text and class lectures. When using the `Node` class, import it from the bag package. Refer to your text, class notes and examples to see examples of how to use a Node that holds a generic type. Remember the UML lists the return type for each method.

SimpleLinkedBag()

In `SimpleLinkedBag`, initialize the instance variables in the constructor. firstNode needs to be null and numberOfEntries needs to be 0.

getCurrentSize() && isEmpty()

Implement these yourself.

add(T anEntry)

Build a new `Node` containing the T entry as a parameter. To check if the list is empty, we call `isEmpty()`.

If it was empty, set your new `Node` to be the `firstNode` field. Otherwise, prepend your node to the `SimpleLinkedBag` by setting the new Node's next to be the `firstNode`. Always reset the `firstNode` to be your new `Node` and increment the `numberOfEntries` field by one. Finally return true, since adding is always successful. Here too think about the case if anEntry is null! In this case it should not add anything and return false.

pick()

If `isEmpty()` returns true, return null. This means there is nothing to draw from the bag.

Otherwise, we need to randomly select from the list. To do this, generate a random number named index using numberOfEntries as the bound. Refer back to the `pick()` section under `SimpleArrayBag` to see how the `TestableRandom` class works.

Starting with the `firstNode`, call `next()` repeatedly for index's number, to arrive at your randomly selected `Node`. Return its `data()`.

getReferenceTo(T anEntry)

We need to search for the entry specified. To start, make a local boolean named `found` and set it to false. Also, make a local Node named `currentNode` and set it equal to `firstNode`.

Next, create a while loop which continues while the entry is not found and there is still more of the bag to search.

Check to see if every `Node` contains the entry by checking if `anEntry` equals that node's contents. Use anEntry's equals() method and currentNode's data() method.

If found, set your boolean to true. Otherwise, move your local Node by setting it to its next() value.

Once out of the while loop, return currentNode. This will be null if it was not found, or not null if we successfully found the entry specified.

remove(T anEntry)

Call getReferenceTo with your entry parameter to check if this LinkedBag contains the entry to remove.

If `getReferenceTo()` returns null, return false. This means we failed to remove the entry.

Otherwise, we need to remove the `Node` given. However, we can't simply take it out. In a singly linked list, the only easily removed `Node` is the first one.

Instead, we will swap the values in the firstNode and our localNode. That way the data we want to remove will be contained in the first node, then we can simply update the firstNode pointer to be firstNode's next().

Decrement the `numberOfEntries`, and return true since we successfully found and removed our entry.

SimpleLinkedBagTest

To ensure that you implemented your LinkedBag's methods as you expect, write a test method for each method you've created. When all your tests pass, you can move on to making your `WhackaShape` game.

Hint: Testing TestableRandom

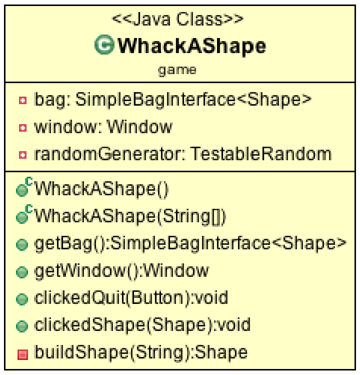
We have you use the provided `TestableRandom` class because you can specify its randomly generated numbers. Having the number specified allows you to repeat the test case. Here's how you can specify the numbers...

```
TestableRandom.setNextInts(0, 42, 7);
```

That line of code predetermines any randomly generated numbers will now be 0, 42, 7, then resume normal behavior. Be sure to set these values before you try to test code which uses the `TestableRandom` class.

Reference Student API for Javadocs on TestableRandom as needed.

WhackAShape



In `WhackAShape`, we implement the methods bottom up. If method A depends on methods B and C, then methods B and C will be implemented before method A.

buildShape(String input)

In the buildShape() method, you will parse the provided `input` to determine whether to create a `CircleShape` or `SquareShape`. Before you create the shape, you need to determine it's size, location and color.

Use TestableRandom to randomly generate an int size ranging from 100-200.

Randomly generate an x and y index for its location. X will range over the window's width, and y over the window's height. Use window's `getGraphPanelWidth()` and `getGraphPanelHeight()` methods. Be sure to subtract the size you generated, since the object will be drawn from its upper left hand corner. We don't want part of the shape hanging off the edge of the screen.

Using the String's `contains()` method and if statements, determine if it says "red" or "blue". If neither of these colors are specified, throw a new `IllegalArgumentException`. This tells whoever gave the string to this method that their input was wrong. Allow the @throws tag to be generated.

Also check to see if the string contains "circle" or "square". If it doesn't, throw a new `IllegalArgumentException`.

If the string does contain "circle" or "square", use x, y, size, and color as parameters to build the appropriate `CircleShape` or `SquareShape`. See the GraphWindow API to look at their methods and constructors. Name your shape `currentShape`.

Tie `currentShape` to the `clickedShape()` method by calling its onClick method with the parameters this and "clickedShape." Be sure to specify the method's name. Return currentShape.

clickedShape(Shape shape)

If the user has clicked the displayed shape, this should call the `clickedShape()` method. Here, we know the user has selected the `Shape` given as the Shape parameter. We now need to remove this `Shape` from the window, by calling the window's `removeShape(shape)` method, and from the bag, by calling the bag's `remove(shape)` method.

Once the clicked shape is removed, the next `Shape` needs to be displayed. Pick a new one from the bag using bag's `pick()` method, and name it `nextShape`. Check to see if `nextShape` is null.

If it is, it means that the bag is empty, and the game is over. Display a new `TextShape` saying "You Win!" in the center of the screen. To get the x and y location of the center of the screen, use `getGraphPanelWidth()` and `getGraphPanelHeight()` on your `Window`.

Otherwise, add `nextShape` to the window using the window's `addShape(nextShape)` method.

clickedQuit(Button button)

Same as project 1, call `System.exit()` with a zero for its parameter. This means there was no errors.

getWindow() && getBag()

Implement these yourself.

WhackAShape()

In the default `WhackAShape` constructor, you will initialize your bag with `SimpleArrayBag` or `SimpleLinkedBag`, your choice. (Feel free to try both! It won't affect your `WhackAShape` code.) Initialize your window, and add a Quit `Button` to your Window's north side, much like in project 1.

To put a few things in your bag, start out by calling `buildShape()` for "red circle", "blue circle", "red square", and "blue square." Be sure to put each `Shape` into your bag.

Add the first `Shape` to the `Window` by calling `pick()`.

WhackAShape(String[] inputs)

Initialize your bag and window as well as add a a Quit `Button` as you did in the default constructor.

For each string in your input parameter, use the `buildShape()` method to convert them into Shapes. Surround this call with a try catch statement. If an exception is caught, you've tried to build a shape from a string which was not one of the four choices. Here it is possible that the string was wrong, since you're using a variable which you did not define.

Inside the catch clause, print out your exception's error message by calling its printStackTrace() method. This is typically autogenerated.

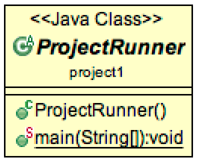
Afterwards, add the shape returned from the buildShape method into your bag.

Once we've finished adding every shape from the input, add the first `Shape` to the window by calling pick().

Build the ProjectRunner Class

Create the `ProjectRunner` class the same way you created the other two classes.

Fill out the fields and methods to match the class diagram. (You don't need to provide the default constructor for this class)



Implement ProjectRunner's main method

main(String[] args)

The main method is called when you run the program. Here, your job is to instantiate the WhackAShape class by calling one of the two constructors you just made. To tell which one to call, check your args parameter. If its length is greater than 0, you can pass the args String array as the input parameter to your constructor. If it's empty, call the default constructor instead.

WhackAShapeTest

We don't require you to write a test class for WhackAShape.

Running your code

You can try running your program as an Application once you've written the main method. Adjust its Run Configurations -> Arguments -> Program Arguments to be one of the example inputs provided here. See the How to Run a program video on the CS2114 youtube site for more details.

example1: "red circle" "blue circle" "red square" "blue square" "red circle" "blue circle" "red square" "blue square"

This input should give you a working game which you can play with.

example2: "beemo was here!"

The second one should make your program throw your `IllegalArgumentException`, and crash.

Submit to Web-Cat. When you see pink lines for code coverage in TestClasses, no points are deducted. If you have any questions or have gotten stuck, go to the CS Lounge (MCB 106) and find a TA during office hours.

Submission status

Submission status	This assignment does not require you to submit anything online
Grading status	Not graded
Due date	Monday, February 22, 2016, 8:00 AM
Time remaining	The due date for this assignment has now passed