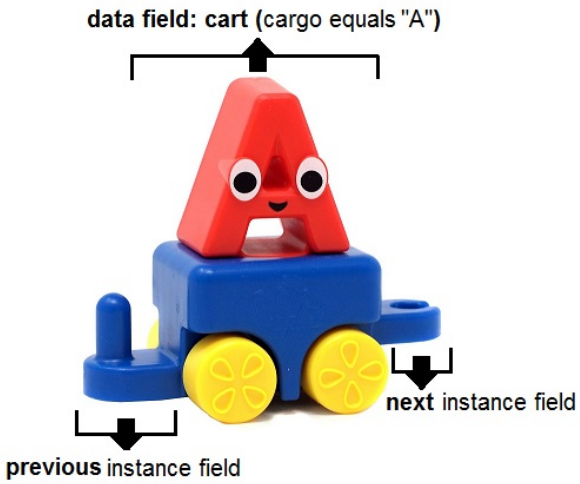
P06 Alphabet Train

Overview

This assignment is inspired by Bob the Train's Alphabet Adventure. It involves the implementation of a chain of alphabet linked carts as a doubly-linked list from scratch. Figure 1 presents a graphic illustration of the objects that we are going to use in this program (a cart, a linked cart, and an alphabet list). Each node in our doubly-linked list will be an object of the LinkedCart class. As shown in Figure 1(a), each LinkedCart contains a link to the previous node, a link o the next node in the list, and a data field called cart. Figures 1(c) and (d) illustrate samples of our alphabet doubly-linked list. Note that the cargo carried by a cart in our alphabet list must be one upper-case alphabet letter. Note also that our list is sorted and does not contain duplicate letters.



(a) One linked cart

(b) A set of linked carts

(c) Alphabet List (sample#1)

(d) Alphabet List (sample#2)

Figure 1: Graphic Illustration of LinkedCart and AlphabetList Objects

Learning Objectives

The goals of this assignment include:

- Implement a doubly linked list of alphabet carts from scratch.
- Further developing your experience working with object oriented design code and exception handling.
- Gain more experience with developing unit tests.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P6 pre-assignment quiz is accessible through Canvas before having access to this specification by 9:59PM on Sunday 10/20/2019. Access to the pre-assignment quiz will be unavailable passing its deadline.
15 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident in this, you should run additional tests of your own.
30 points	**Manual Grading and Supplemental Automated Tests:** When your final grade feedback appears on Gradescope, it will include the feedback from these additional automated grading tests, as well as feedback from human graders who review the code in your submission by hand.

Assignment Requirements and Reminders

- EVERYONE must complete the pair programming registration form by **Sunday, Mar. 8^{th} @ 9:59PM**.
- DO NOT submit the provided `SortedListADT.java` and `Cart.java` source files on Gradescope.
- You HAVE TO submit ONLY the three source files that you created on Gradescope.
- You ARE NOT allowed to use a dummy node at the head of the AlphabetList doubly-linked list.
- You ARE NOT allowed to add any fields either instance or static, and any public methods either static or instance to your `LinkedCart`, and `AlphabetList` classes, other than those defined in this write-up and these javadocs.

- You CAN define local variables that you may need to implement the methods defined in this program.
- You CAN define private methods to help implement the different public methods defined in these javadocs, if needed.
- ALL your test methods MUST be implemented in your `AlphabetListTester` class.
- In addition to the required test methods, we HIGHLY recommend (not require) that you develop your additional own unit tests (**public static methods that return a boolean**).
- Ensure that your code for every assignment is styled in conformance to CS300 Course Style Guide.
- Feel free to reuse the javadoc method headers provided in these javadocs in your class and method javadoc headers.

1 Getting Started

Start by creating a new Java Project in eclipse. You may call it P06 Alphabet Train, for instance. Then, download and add these provided SortedListADT.java, and Cart.java source files to your project. The provided `SortedListADT` interface includes all the abstract method which must be implemented by our sorted list. The `Cart` class represents the data objects carried in our list. Read carefully through the provided code and the details in the javadoc method headers. You can notice that the class Cart implements the Comparable< *Cart* > interface.

Now, download this AlphabetListTester.java file and add it your project. This file represents a skeleton for the `AlphabetListTester` class. You HAVE TO implement all your P06's test methods in this file (the required ones whose signatures and javadocs method headers are already added to this provided AlphabetListTester.java source file, and the ones that you may define in your own). Note also that the `AlphabetListTester` MUST be the only class which may include a main method in your submission. In the next steps, you are going to create and implement the two other classes that you have to submit on gradescope `LinkedCart` and `AlphabetList`, and add them to this project.

2 Create the LinkedCart class

Create a new class called `LinkedCart`. Each instance of this class represents a linked cart (as graphically illustrated in 1.(a) and (b). Every LinkedCart object should have ONLY the THREE following instance fields.

```
private final Cart CART; // data field of this linked Cart
private LinkedCart previous; // reference to the previous linked cart in
                             // a list of carts
private LinkedCart next; // reference to the next linked cart in a list of carts
```

Notice that the date field CART is **final**, meaning that once assigned to a LinkedCart, you cannot change it. Note also that the class `Cart` contains only a getter for the cargo instance field. No setter is defined to change the letter of a cart once created.

Now, implement the constructors and the public methods defined for the `LinkedCart` class according to their detailed javadocs description provided within these javadocs. At meantime, implement the test method *testLinkedCart()* in your `AlphabetListTester` class with accordance to the details provided in its javadoc style method header.

3 Create the AlphabetList class

Now, create a new class called `AlphabetList` and add it to your P06 project source folder. This class MUST implement the provided interface SortedListADT< *Cart* >. This class models the doubly-linked list data structure which stores elements of type `Cart` in ascending sorted order. This class MUST define the following instance and static fields ONLY.

```
private static final Cart MIN_CART = new Cart("A"); // The smallest cart that
                                    //  can be added to this sorted list
private static final Cart MAX_CART = new Cart("Z"); // The largest cart that
                                    // can be added to this sorted list
private LinkedCart head; // head of this doubly linked list
private LinkedCart tail; // tail of this doubly linked list
private int size; // size of this list
private int capacity; // maximum number of carts which can be stored in this list
```

You are not allowed to add any additional instance or static field to this class. Recall also that you ARE NOT allowed to use a dummy node at the head of the AlphabetList.

Now, make sure to implement the two constructors and all the methods defined for this class with respect to the specification provided in these javadocs. Descriptive error messages related to each exception which may be thrown by your methods are provided in the details of the javadocs style method header comments of the class `AlphabetList`. In addition, we provide you in the following paragraph with the implementation of the AlphabetList.toString() method. We note that our list of carts will be implemented using linked carts. Our alphabet list can store only carts in the range of MIN_CART . . . MAX_CART. Duplicate items are not allowed in this list. Further details are provided in these javadocs.

Provided implementation of `AlphabetList.toString()` method:

```
String string = "This list contains " + size + " cart(s)";
if (size == 0) {
  return string;
}
string += ": [ ";
LinkedCart currentCart = head;
while (currentCart != null) {
  string += currentCart.getCart().toString() + " ";
  currentCart = currentCart.getNext();
}
string += "]";
return string;
```

Make sure to design and implement all the test methods defined in these javadocs. If a test method contains multiple test scenarios, we recommend that you break it down into smaller private test methods. You are also HIGHLY encouraged to add additional test methods to your `AlphabetListTester` class to gain confidence that are the methods defined in your `AlphabetList` are working correctly with respect to these javadocs. Recall that all your test methods MUST be static, do not take any input parameter, and return a boolean (true if the expected behavior is satisfied and false otherwise).

Here are some tips for the implementation of your test methods. First, you have to read the method's javadoc style header and determine how each method is supposed to operate with different input parameters. For instance, you can ask yourself the following questions:

1. Is a newly created train empty? (it should be empty)
2. Is a train with a single cart empty? (it should not be empty)
3. If you add a cart with "Z" and then a cart with "A", does your train place the "A" cart before the "Z" cart? (it should place the "A" cart first)
4. If you add two carts with "A", does it throw an exception? (it should throw an exception with the right description)

Illustrative Example

In order to provide you with a better understanding on how to use the implemented classes, we provide in the following an example of source code and its expected output, when the methods are called with correct input arguments.

```
AlphabetList letters = new AlphabetList();
System.out.println(letters);
letters.add(new Cart("D"));
System.out.println(letters);
letters.add(new Cart("B"));
System.out.println(letters);
letters.add(new Cart("C"));
System.out.println(letters);
letters.add(new Cart("F"));
System.out.println(letters);
letters.add(new Cart("A"));
System.out.println(letters);
letters.add(new Cart("Z"));
System.out.println(letters);
letters.add(new Cart("E"));
System.out.println(letters);
System.out.println("Read Forward: " + letters.readForward());
System.out.println("Read Backward: " + letters.readBackward());
letters.remove(0);
System.out.println(letters);
letters.remove(letters.size() -1);
System.out.println(letters);
letters.remove(3);
System.out.println(letters);
System.out.println("Read Forward: " + letters.readForward());
System.out.println("Read Backward: " + letters.readBackward());
letters.clear();
System.out.println(letters);
System.out.println("Read Forward: " + letters.readForward());
System.out.println("Read Backward: " + letters.readBackward());
```

Expected output:

```
This list contains 0 cart(s)
This list contains 1 cart(s):  [ D ]
This list contains 2 cart(s):  [ B D ]
This list contains 3 cart(s):  [ B C D ]
This list contains 4 cart(s):  [ B C D F ]
This list contains 5 cart(s):  [ A B C D F ]
This list contains 6 cart(s):  [ A B C D F Z ]
This list contains 7 cart(s):  [ A B C D E F Z ]
Read Forward:  ABCDEFZ
Read Backward:  ZFEDCBA
This list contains 6 cart(s):  [ B C D E F Z ]
This list contains 5 cart(s):  [ B C D E F ]
This list contains 4 cart(s):  [ B C D F ]
Read Forward:  BCDF
Read Backward:  FDCB
This list contains 0 cart(s)
Read Forward:
Read Backward:
```

4 Assignment 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 CS300 Course Style Guide, you should submit your final work through gradescope.com. The only 3 files that you must submit include: `LinkedCart.java`, `AlphabetList.java`, and `AlphabetListTester.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline of Due: **9:59PM on March 11**th. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the CS300 Course Style Guide.