

CICS-160 Fall 2023

Assignment #5

Due on Sunday December 10

Learning Goals:

This assignment is designed for you to demonstrate engagement with a broad set of topics related to our course. Among these: knowledge of both implementing and using queues, your ability to abstract a class definition by having objects such as lists and queues inside other objects, testing via Java JUnit tests, and to find and use online documentation about the built-in Java ArrayList and LinkedList classes. In addition, this assignment will be good practice on the process of dividing bigger tasks into smaller components, and then implementing those smaller components, which is an important skill in computer programming.

Your tasks

- Implement class Queue, with the following methods:
 - enqueue(e)
 - dequeue()
 - isEmpty()
 - peek()

Your queue needs to be able to store a not-defined number of elements, expanding as needed. This will require you implement your Queue via lists as opposed to arrays. Because of the way objects of type Queue will be used in the rest of the assignment, your implementation of class Queue must allow its user to store any type of object inside of it, but only one type at a time, depending on how the Queue is declared, via Generics. For example, you should be able to declare a Queue of Integer objects, and a Queue of Car objects, and a Queue of Person objects, but a Queue of Car objects should NOT be able to store an Integer object, or a Person object.

- Implement a class called Car. **Please notice that this Car class is different from the one we have seen and used earlier during this semester.** This class will have three attributes: id, powerSource and pricePerDay. id is of type int. powerSource is of type int, and will take values of 1, 2, or 3. 1 will be used to signify that the Car is a gasoline car. 2 will be used to signify that the car is a hybrid car. 3 will be used to signify that the car is an electric car. pricePerDay will store a real (i.e. floating point) value. To store a number as a float, append the letter 'f' at the end (i.e 2.5f). This class will have get and set methods for each of its attributes, and whatever constructor(s) you choose to implement. The name for the get and set methods will be getId, setId, getPowerSource, setPowerSource, getPricePerDay, and setPricePerDay.

- Implement a class called CarRequests. This class has as an attribute a Queue of Integer. The values inside this queue will indicate what type of vehicle customers are requesting. For example, a queue with values 1,2,3,2,2 means that customers are requesting a gasoline car, a hybrid car, an electric car, a hybrid car, and a hybrid car, in that order.
 - Implement CarRequests.addRequest(int x), which adds a request for the type of Car indicated by argument x to the Queue CarRequests stores internally. addRequest does not return a value.
 - Implement CarRequests.getRequest(), which returns the int at the front of the Queue CarRequests stores internally.
 - Implement a class called CarRequests.hasPendingRequests(). hasPendingRequests() returns false if the Queue maintained inside CarRequests is empty, and true otherwise.
- Implement a class called CarLot. This class will internally have three objects of type Queue, each of them storing Car objects. While these three queues all store Car objects, we will use them to store different types of cars. One queue will have only gasoline cars, one queue will have only hybrid cars, and one queue will have only electric cars.
 - Class CarLot has a method called addCar. addCar takes as input an object of type Car, and places it in the appropriate queue inside carLot based on the type of car that is being added (gasoline, hybrid, or electric). If the powerSource value does not correspond to any one of the carLot queues, the method simply ignores the car and does not add it to any queue. addCar returns true when the car is added to one of the queues, and false otherwise.
 - Class CarLot will have a method called processRequests. processRequests will receive as input an object of type CarRequests. The output of processRequests will be a list (not an array, not a CarLot) of objects of type Car, satisfying the requests being placed by customers. These objects will come from the queues that class CarLot is keeping internally. For example, with aCarRequests object storing a Queue with values 1,2,3,2,2 processRequests will return a list that has one car from the gasoline cars queue, one car from the hybrid cars queue, one car from the electric cars queue, and two cars from the hybrid cars queue, in that order. At any point that a car is placed in the list that processRequests will return, it is taken out of the internal queue it was in. Your code must only use valid queue operations while working with the queues that CarLot keeps internally. If at any point the Queue inside CarRequests has a request that cannot be satisfied (for example, a customer is requesting a hybrid car, but the queue for hybrid cars is empty) a Car with an id of zero will be placed in the output list.
- Write Junit tests for all methods in class Queue, class Car, and class CarLot.

Submit your work in seven files: Car.java, CarTests.java, CarRequests.java, CarLot.java, CarLotTests.java, Queue.java, and QueueTests.java.

Points distribution: see Gradescope for autograded parts of the assignment.
The following items will be graded manually (30 points total):

- Declaration of class Queue uses Generics 5 points
- JUnit tests for class Queue
 - Constructor Queue() 1 point
 - enqueue(e) 2 points
 - dequeue() → e 2 points
 - peek() → e 2 points
 - isEmpty() → boolean 2 points
- JUnit tests for class Car
 - Constructor Car(int id, int powerSource, float pricePerDay) 1 point
 - getId() → int 1 point
 - setId(int) 1 point
 - getPowerSource() → int 1 point
 - setPowerSource(int) 1 point
 - getPricePerDay() → float 1 point
 - setPricePerDay(float) 1 point
- JUnit tests for CarLot
 - Constructor CarLot() 1 point
 - addCar(Car) → boolean 3 points
 - processRequests(CarRequests) → List 5 points