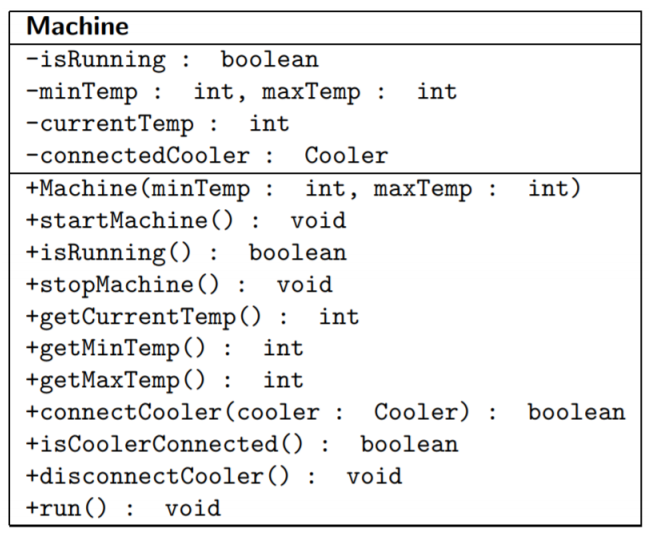
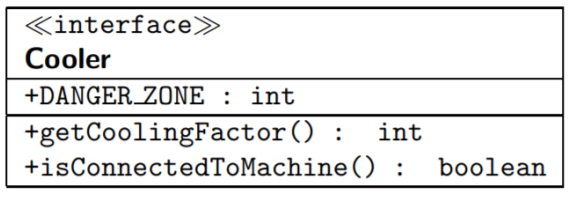
**Question 1)**

Using the UML diagram below create a class called Machine that represents a factory machine.



While not running, the machines temperature is just at room temperature, however if the machine is running its temperature gradually rises by some random value between 0−5 degrees every 200mS. In order to keep its temperature within a required range (between minTemp and maxTemp), it needs to be occasionally cooled by a Cooler object which can connect and disconnect to this machine. This is represented by the connectedCooler field which can be null if there is no Cooler currently connected. When a Cooler is connected the current temperature will instead drop by a value determined by the connected cooler’s getCoolingFactor method every 200ms.

A Cooler is described by the interface below and is designed to only connect to one machine at a time. Create this interface as shown in the UML and set its DANGER\_ZONE constant to 50.



If machine overheats or overcools (above and below the max and min temperature), it should stop running and throw a custom RuntimeException called MachineTemperatureException. The startMachine method should start its own instance machine running as a new thread and stopMachine should terminate itself as a running thread.

**Question 2)**

Prepare a class called MonitoringCooler that implements both the Cooler interface and the Runnable interface, representing a cooler for monitoring a collection of machines. It should have a constructor which accepts as parameters a collection of machines to monitor and a *cooling factor* which a *Machine* might use to cool itself down.

The class should contain a startCooler method that can run this instance object as a separate thread. The run method should continually monitor the machines until its requestStop method is called. If the cooler is not connected to a machine at some instant, then the monitoring consists of continually checking whether any currently running machine gets dangerously close to overheating (determined by the machines maximum temperature minus the DANGER\_ZONE). In this case the cooler should connect to that machine in order to cool it.

While the cooler is connected to a machine, its monitoring consists of checking whether the connected machine is dangerously close to overcooling (machine minimum temperature plus the DANGER\_ZONE), in which case it should disconnect itself from the machine and then continue monitoring its collection of machines.

Although a Cooler monitors a collection of machines, it can only connect to one machine at a time in order to cool it. Be careful not to connect to machines that have already over cooled or over heated and so not running anymore. Also be careful not to attach to any machines that already have another Cooler instance attached.

**Question 3)**

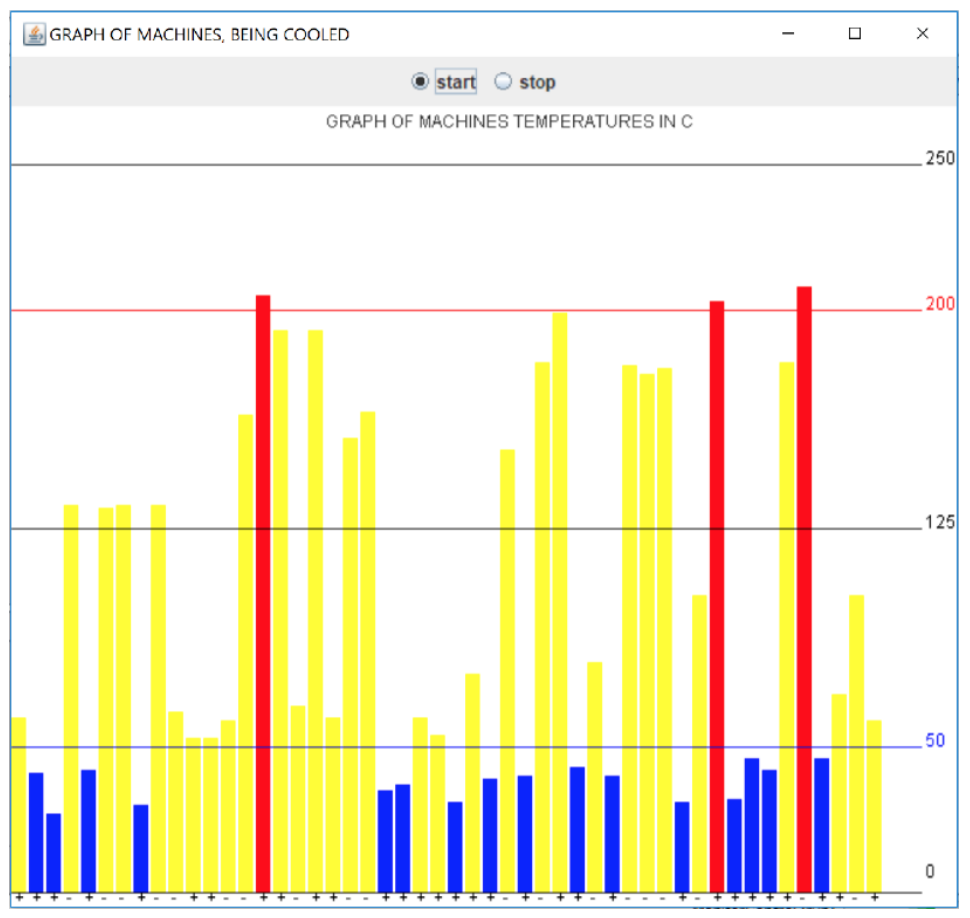
Create a class called FactoryControlGUI, which shows a graph of a running collection of Machine objects along the x-axis and temperature along the y-axis. It should show guide lines showing safe operating temperatures between 50 - 200 degrees, overheated lines at 250 degrees and overcooled at 0 degrees Celsius.).

As machines run, the bar graph dynamically increases in value determined by the temperature of each instance of Machine. The colour of the bar graph should be set blue if machine is below within the danger zone of being overcooled, yellow if machine in the safe operating temperature, and red if the machine is in the danger zone of being overheated.

All machines are also monitored by a collection of several MonitoringCooler objects that all have a cooling factor of 25 degrees Celsius. Each cooler should monitor the entire collection of machines attaching and detaching as needed. Add “+” and “-” strings to the bottom of each machine bar to indicate whether the cooler is connected or not.

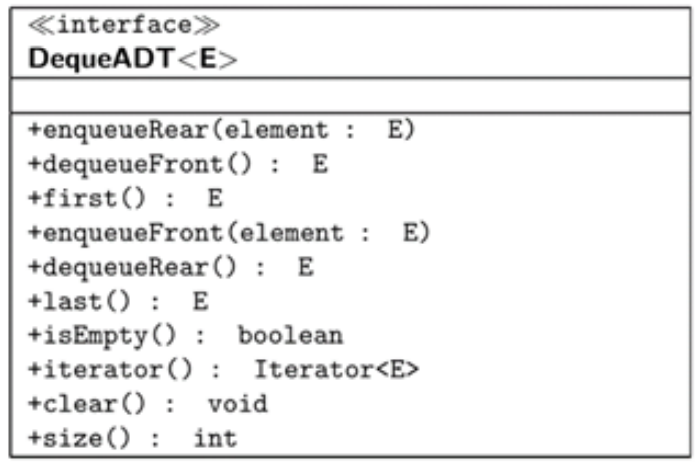
Add two radio buttons used to start and stop all running threads. When the startButton radio button is pressed it should start up all the machine and cooler threads as well as a separate javax.swing.Timer to perform the repainting on the panel. Once the stopButton radio button is pressed it should stop all the running threads and the timer. Use constants to determine the number of machines and coolers within your program.

Because MonitoringCooler and Machine are multi-threaded, many of the threads will be accessing and changing the state of objects. To avoid race conditions, make any necessary changes to Question 1 and/or Question 2 to make this program thread safe. Feel free to experiment with number of coolers and machines as well as DANGER\_ZONE and cooling factor values to fully test the program.



**Question 4)**

A deque (pronounced deck) or double-ended queue, is a linear collection that allows insertion and deletion at both ends. The interface above called DequeADT available from Blackboard, is an abstract data type which describes operations that a deque should have.



* It has *enqueue* and *dequeue* methods for adding or removing an element from the front and rear of the deque.
* *first* and *last* for obtaining the elements (but not removing) at the front and rear of the deque.
* *isEmpty* for checking whether any elements are in the deque.
* *clear* for clearing out all elements in deque.
* *size* for returning the number of elements currently in the deque.
* *iterator* which can be used to contiguously iterate over the elements in the deque, going from the front of the deque to the rear.

Implement the Deque abstract data type using **two** different underlying data structures. One called LinkedDeque which uses a suitable linking structure, and one called ArrayDeque which uses a suitable underlying array structure. Implement the data structure yourself, do not use extra Java Collection classes.

For efficiency *first, last, enqueueRear, enqueueFront, dequeueRear, dequeueFront, clear, size,* and *isEmpty* should all be O(1) operations. Add a suitable toString which returns a string representation of each deque implementation.

Further enhance your classes to make sure all methods are safe from unexpected events using appropriate exception handling (example, if user code tries to remove something from the deque when empty, it should throw an appropriate exception).

Create a test class with a suitable main method which effectively tests all operations on one or both Deque implementations.

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package deque;

import java.util.Iterator;

/\*\*

\*

\*/

interface DequeADT<E> {

public void enqueueRear(E element);

public E dequeueFront();

public E first();

public E last();

public void enqueueFront(E element);

public E dequeueRear();

public boolean isEmpty();

public int size();

public Iterator<E> iterator();

}

**Question 5)**

Design and create a class that uses an algorithm and an appropriate data structure used to efficiently evaluate whether opening and closing bracket and brace pairs match up inside any given string in O(n) time, where n is the length of the string. All other content in the string can be ignored.

These pairs are: () {} <> []

Example:

String: “{((2 x 5)+(3\*-2 + 5))}” - will evaluate successfully

String: “{ (2 x 5)+(3\*-2 + 5))}” - will not as one of the opening brackets ( missing

Example:

String: “List<List>” - will evaluate successfully

String: “List<List ” - will not as missing > closing angle brace

Example:

String: “{(<>){}{…}(e(e)e){hello}}” - will evaluate

String: “{(< eeeek>>){}{…} e(e)e){hello} ” - will not evaluate for multiple reasons