A fuel depot has a number of fuel tanks arranged in a line and a robot that moves a filling mechanism back and forth along the line so that the tanks can be filled. A fuel tank is specified by the `FuelTank` interface below.

```
public interface FuelTank
{
  /** @return an integer value that ranges from 0 (empty) to 100 (full) */
  int getFuelLevel();
}
```

A fuel depot keeps track of the fuel tanks and the robot. The following figure represents the tanks and the robot in a fuel depot. The robot, indicated by the arrow, is currently at index 2 and is facing to the right.

Tank index	0	1	2	3	4	5
Fuel level in tank	80	70	20	45	50	25
Robot			➔			

The state of the robot includes the index of its location and the direction in which it is facing (to the right or to the left). This information is specified in the `FuelRobot` interface as shown in the following declaration.

+

The state of the robot includes the index of its location and the direction in which it is facing (to the right or to the left). This information is specified in the `FuelRobot` interface as shown in the following declaration.

```
public interface FuelRobot
{
  /** @return the index of the current location of the robot */
  int getCurrentIndex();


  /** Determine whether the robot is currently facing to the right
   *  @return true  if the robot is facing to the right (toward tanks with larger indexes)
   *          false if the robot is facing to the left (toward tanks with smaller indexes)
   */
  boolean isFacingRight();


  /** Changes the current direction of the robot */
  void changeDirection();


  /** Moves the robot in its current direction by the number of locations specified.
   *  @param numLocs the number of locations to move. A value of 1 moves
   *                 the robot to the next location in the current direction.
   *          Precondition: numLocs > 0
   */
  void moveForward(int numLocs);
}
```

A fuel depot is represented by the `FuelDepot` class as shown in the following class declaration.

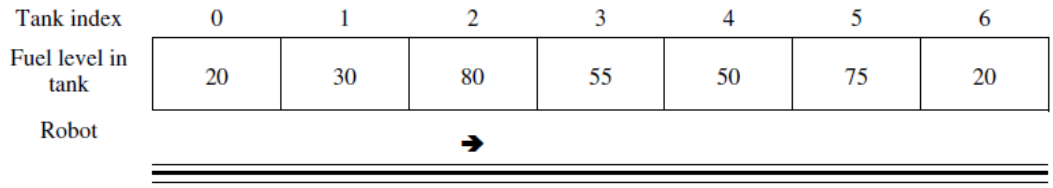
```
public class FuelDepot
{
  /** The robot used to move the filling mechanism */
  private FuelRobot filler;

  /** The list of fuel tanks */
  private List<FuelTank> tanks;

  /** Determines and returns the index of the next tank to be filled.
   *  @param threshold fuel tanks with a fuel level ≤ threshold may be filled
   *  @return index of the location of the next tank to be filled
   *  Postcondition: the state of the robot has not changed
   */
  public int nextTankToFill(int threshold)
  {  /* to be implemented in part (a) */  }


  /** Moves the robot to location locIndex.
   *  @param locIndex the index of the location of the tank to move to
   *         Precondition: 0 ≤ locIndex < tanks.size()
   *  Postcondition: the current location of the robot is locIndex
   */
  public void moveToLocation(int locIndex)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the `FuelDepot` method `nextTankToFill` that returns the index of the next tank to be filled.

The index for the next tank to be filled is determined according to the following rules:

- Return the index of a tank with the lowest fuel level that is less than or equal to a given threshold.
  If there is more than one fuel tank with the same lowest fuel level, any of their indexes can be returned.
- If there are no tanks with a fuel level less than or equal to the threshold, return the robot's current index.

For example, suppose the tanks contain the fuel levels shown in the following figure.



The following table shows the results of several independent calls to `nextTankToFill`.

`threshold`	Return Value	Rationale
50	0 or 6	20 is the lowest fuel level, so either 0 or 6 can be returned.
15	2	There are no tanks with a fuel level ≤ `threshold`, so the robot's current index is returned.

Complete method `nextTankToFill` below.

```
/** Determines and returns the index of the next tank to be filled.
 *  @param threshold fuel tanks with a fuel level ≤ threshold may be filled
 *  @return index of the location of the next tank to be filled
 *  Postcondition: the state of the robot has not changed
 */
public int nextTankToFill(int threshold)
```

(b) Write the `FuelDepot` method `moveToLocation` that will move the robot to the given tank locati
Because the robot can only move forward, it may be necessary to change the direction of the robot before having it move. Do not move the robot past the end of the line of fuel tanks.

Complete method `moveToLocation` below.

```
/** Moves the robot to location locIndex.
 *  @param locIndex the index of the location of the tank to move to
 *         Precondition: 0 ≤ locIndex < tanks.size()
 *  Postcondition: the current location of the robot is locIndex
 */
public void moveToLocation(int locIndex)
```

2011 CANONICAL SOLUTIONS

Question 3: Fuel Depot

Part (a):

```
public int nextTankToFill(int threshold) {
  int minLevel = this.tanks.get(0).getFuelLevel();
  int minTankIndex = 0;
  for (int i = 1; i < this.tanks.size(); i++) {
    if (this.tanks.get(i).getFuelLevel() < minLevel) {
      minLevel = this.tanks.get(i).getFuelLevel();
      minTankIndex = i;
    }
  }
  if (minLevel <= threshold) {
    return minTankIndex;
  } else {
    return this.filler.getCurrentIndex();
  }
}
```

// Alternative solution

```
public int nextTankToFillA(int threshold) {
  int minTankIndex = this.filler.getCurrentIndex();
  for (int i = 0; i < this.tanks.size(); i++) {
    if (this.tanks.get(i).getFuelLevel() <= threshold &&
        this.tanks.get(i).getFuelLevel() <
          this.tanks.get(minTankIndex).getFuelLevel()) {
      minTankIndex = i;
    }
  }
  return minTankIndex;
}
```

Part (a)	`nextTankToFill`	**5 points**

Intent: *Return index of tank with minimum level* (`<= threshold`)

+4 Determine minimum element of `tanks` that is `<= threshold`, if any

- **+1½** Consider fuel levels of elements of `tanks`
  - **+½** Accesses fuel level of an element of `tanks`
  - **+½** Accesses at least one element of `tanks` in context of repetition (iteration/recursion)
  - **+½** Accesses every element of `tanks` at least once
- **+2½** Identify minimum element of `tanks` that is `<= threshold`
  - **+½** Compares fuel levels from at least two elements of `tanks`
  - **+½** Implements algorithm to find minimum
  - **+½** Identifies tank (*object or index*) holding identified minimum
  - **+½** Compares `threshold` with fuel level from at least one element of `tanks`
  - **+½** Determines element identified as minimum fuel level that is also `<= threshold`

+1 Return the index of the element satisfying the conditions, or the current index if no element does so

- **+½** Returns index of element identified as satisfying threshold & minimum conditions*
- **+½** Returns `filler.getCurrentIndex()` when no element satisfies conditions*

**Note: Point is not awarded if wrong data type is returned.*

Part (b):

```
public void moveToLocation(int locIndex) {
  if (this.filler.getCurrentIndex() > locIndex) {
    if (this.filler.isFacingRight()) {
      this.filler.changeDirection();
    }
    this.filler.moveForward(this.filler.getCurrentIndex() - locIndex);
  }
  if (this.filler.getCurrentIndex() < locIndex) {
    if (!this.filler.isFacingRight()) {
      this.filler.changeDirection();
    }
    this.filler.moveForward(locIndex - this.filler.getCurrentIndex());
  }
}
```

Part (b)	`moveToLocation`	**4 points**

Intent: Move robot to given tank location

+2 Ensure robot is pointing in direction of tank to be filled
- **+½** Determines direction `filler` is currently facing
- **+½** Changes `filler`'s direction for some condition
- **+1** Establishes `filler`'s direction as appropriate for all conditions

+2 Place robot at specified location
- **+½** Invokes `moveForward` method with a parameter
- **+½** Invokes `moveForward` method with a verified non-zero parameter
- **+1** Invokes `filler.moveForward` method with a correctly computed parameter