FRQ2

1. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

This question involves the implementation of the `AddtionPattern` class, which generates a number pattern. The `AdditionPattern` object is constructed with two positive integer parameters, as described below.

The first positive integer parameter indicates the starting number in the pattern.
The second positive integer parameter indicates the value that is to be added to obtain each subsequent number in the pattern.

The `AdditionPattern` class also supports the following methods.

`currentNumber`, which returns the current number in the pattern
`next`, which moves to the next number in the pattern
`prev`, which moves to the previous number in the pattern or takes no action if there is no previous number

The following table illustrates the behavior of an `AdditionPattern` object that is instantiated by the following statement.

```
AdditionPattern plus3 = new AdditionPattern(2, 3);
```

FRQ2

Method Call	Value Returned (blank if no value)	Explanation
`plus3.currentNumber();`	`2`	The current number is initially the starting number in the pattern.
`plus3.next();`		The pattern adds `3` each time, so move to the next number in the pattern (`5`).
`plus3.currentNumber();`	`5`	The current number is `5`.
`plus3.next();`		The pattern adds `3` each time, so move to the next number in the pattern (`8`).
`plus3.next();`		The pattern adds `3` each time, so move to the next number in the pattern (`11`).
`plus3.currentNumber();`	`11`	The current number is `11`.
`plus3.prev();`		Move to the previous number in the pattern (`8`).
`plus3.prev();`		Move to the previous number in the pattern (`5`).
`plus3.prev();`		Move to the previous number in the pattern (`2`).
`plus3.currentNumber();`	`2`	The current number is `2`.
`plus3.prev();`		There is no previous number in the pattern prior to `2`, so no action is taken.
`plus3.currentNumber();`	`2`	The current number is `2`.

Write the complete `AditonPattern` class. Your implementation must meet all specifications and conform to all examples.

FRQ2

2. **Directions**: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

An APLine is a line defined by the equation $ax + by + c = 0$, where a is not equal to zero, b is not equal to zero, and a, b, and c are all integers. The slope of an APLine is defined to be the double value $-a\ b/$. A point (represented by integers x and y) is on an APLine if the equation of the APLine is satisfied when those x and y values are substituted into the equation. That is, a point represented by x and y is on the line if $ax + by + c$ is equal to 0. Examples of two APLine equations are shown in the following table.

Equation	Slope ($-a$ / b)	Is point (5, -2) on the line?
$5x + 4y - 17 = 0$	-5 / 4 = -1.25	Yes, because $5(5) + 4(-2) + (-17) = 0$
$-25x + 40y + 30 = 0$	25 / 40 = 0.625	No, because $-25(5) + 40(-2) + 30 \neq 0$

Assume that the following code segment appears in a class other than APLine. The code segment shows an example of using the APLine class to represent the two equations shown in the table.

```
APLine line1 = new APLine(5, 4, -17);
double slope1 = line1.getSlope();         // slope1 is assigned -1.25
boolean onLine1 = line1.isOnLine(5, -2); // true because 5(5) + 4(-2) + (-17) = 0

APLine line2 = new APLine(-25, 40, 30);
double slope2 = line2.getSlope();         // slope2 is assigned 0.625
boolean onLine2 = line2.isOnLine(5, -2); // false because -25(5) + 40(-2) + 30 ≠ 0
```

Write the APLine class. Your implementation must include a constructor that has three integer parameters that represent a, b, and c, in that order. You may assume that the values of the parameters representing a and b are not zero. It must also include a method getSlope that calculates and returns the slope of the line, and a method isOnLine that returns true if the point represented by its two parameters (x and y, in that order) is on the APLine and returns false otherwise. Your class must produce the indicated results when invoked by the code segment given above. You may ignore any issues related to integer overflow.

FRQ2

3. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

The `Bus` class simulates the activity of a bus. A bus moves back and forth along a single route, making stops along the way. The stops on the route are numbered consecutively starting from 1 up to and including a number that is provided when the `Bus` object is created. You may assume that the number of stops will always be greater than 1.

The bus starts at the first stop and is initially heading toward the last stop. At each step of the simulation, the bus is at a particular stop and is heading toward either the first or last stop. When the bus reaches the first or last stop, it reverses direction. The following table contains a sample code execution sequence and the corresponding results.

FRQ2

Statement or Expression	Value returned (blank if no value)	Comment
`Bus bus1 = new Bus(3);`		The route for `bus1` has three stops numbered 1–3.
`bus1.getCurrentStop();`	1	`bus1` is at stop 1 (first stop on the route).
`bus1.move();`		`bus1` moves to the next stop (2).
`bus1.getCurrentStop();`	2	`bus1` is at stop 2.
`bus1.move();`		`bus1` moves to the next stop (3).
`bus1.getCurrentStop();`	3	`bus1` is at stop 3.
`bus1.move();`		`bus1` moves to the next stop (2).
`bus1.getCurrentStop();`	2	`bus1` is at stop 2.
`bus1.move();`		`bus1` moves to the next stop (1).
`bus1.move();`		`bus1` moves to the next stop (2).
`bus1.getCurrentStop();`	2	`bus1` is at stop 2.
`bus1.getCurrentStop();`	2	`bus1` is still at stop 2.
`Bus bus2 = new Bus(5);`		The route for `bus2` has five stops numbered 1–5.
`bus1.getCurrentStop();`	2	`bus1` is still at stop 2.
`bus2.getCurrentStop();`	1	`bus2` is at stop 1 (first stop on the route).

Write the complete `Bus` class, including the constructor and any required instance variables and methods. Your implementation must meet all specifications and conform to the example.

FRQ2

4. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

This question involves the implementation of a fitness tracking system that is represented by the `StepTracker` class. A `StepTracker` object is created with a parameter that defines the minimum number of steps that must be taken for a day to be considered *active*.

The `StepTracker` class provides a constructor and the following methods.

`addDailySteps`, which accumulates information about steps, in readings taken once per day
`activeDays`, which returns the number of active days
`averageSteps`, which returns the average number of steps per day, calculated by dividing the total number of steps taken by the number of days tracked

The following table contains a sample code execution sequence and the corresponding results.

FRQ2

Statements and Expressions	Value Returned (blank if no value)	Comment
`StepTracker tr = new StepTracker(10000);`		Days with at least 10,000 steps are considered active. Assume that the parameter is positive.
`tr.activeDays();`	`0`	No data have been recorded yet.
`tr.averageSteps();`	`0.0`	When no step data have been recorded, the `averageSteps` method returns `0.0`.
`tr.addDailySteps(9000);`		This is too few steps for the day to be considered active.
`tr.addDailySteps(5000);`		This is too few steps for the day to be considered active.
`tr.activeDays();`	`0`	No day had at least 10,000 steps.
`tr.averageSteps();`	`7000.0`	The average number of steps per day is $(14000 / 2)$.
`tr.addDailySteps(13000);`		This represents an active day.
`tr.activeDays();`	`1`	Of the three days for which step data were entered, one day had at least 10,000 steps.
`tr.averageSteps();`	`9000.0`	The average number of steps per day is $(27000 / 3)$.
`tr.addDailySteps(23000);`		This represents an active day.
`tr.addDailySteps(1111);`		This is too few steps for the day to be considered active.
`tr.activeDays();`	`2`	Of the five days for which step data were entered, two days had at least 10,000 steps.
`tr.averageSteps();`	`10222.2`	The average number of steps per day is $(51111 / 5)$.

Write the complete `StepTracker` class, including the constructor and any required instance variables and methods. Your implementation must meet all specifications and conform to the example.