1. A travel agency maintains a list of information about airline flights. Flight information includes a departure time and an arrival time. You may assume that the two times occur on the same day. These times are represented by objects of the `Time` class.

   The declaration for the `Time` class is shown below. It includes a method `minutesUntil` that returns the difference (in minutes) between the current `Time` object and another `Time` object.

```
public class Time
{
  /** @return difference, in minutes, between this time and other;
   *          difference is negative if other is earlier than this time
   */
  public int minutesUntil(Time other)
  {  /* implementation not shown */  }

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

   For example, assume that `t1` and `t2` are `Time` objects where `t1` represents 1:00 P.M. and `t2` represents 2:15 P.M. The call `t1.minutesUntil(t2)` will return `75` and the call `t2.minutesUntil(t1)` will return `-75`.

The declaration for the `Flight` class is shown below. It has methods to access the departure time and the arrival time of a flight. You may assume that the departure time of a flight is earlier than its arrival time.

```
public class Flight
{
    /** @return time at which the flight departs
     */
    public Time getDepartureTime()
    { /* implementation not shown */ }

    /** @return time at which the flight arrives
     */
    public Time getArrivalTime()
    { /* implementation not shown */ }

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

A trip consists of a sequence of flights and is represented by the `Trip` class. The `Trip` class contains an `ArrayList` of `Flight` objects that are stored in chronological order. You may assume that for each flight after the first flight in the list, the departure time of the flight is later than the arrival time of the preceding flight in the list. A partial declaration of the `Trip` class is shown below. You will write two methods for the `Trip` class.

```
public class Trip
{
  private ArrayList<Flight> flights;
    // stores the flights (if any) in chronological order

  /** @return the number of minutes from the departure of the first flight to the arrival
   *          of the last flight if there are one or more flights in the trip;
   *          0, if there are no flights in the trip
   */
  public int getDuration()
  {  /* to be implemented in part (a) */  }

  /** Precondition: the departure time for each flight is later than the arrival time of its
   *                preceding flight
   *  @return the smallest number of minutes between the arrival of a flight and the departure
   *          of the flight immediately after it, if there are two or more flights in the trip;
   *          -1, if there are fewer than two flights in the trip
   */
  public int getShortestLayover()
  {  /* to be implemented in part (b) */  }

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

(a) Complete method `getDuration` below.

```
/** @return the number of minutes from the departure of the first flight to the arrival
 *          of the last flight if there are one or more flights in the trip;
 *          0, if there are no flights in the trip
 */
public int getDuration()
```

(b) Write the `Trip` method `getShortestLayover`. A layover is the number of minutes from the arrival of one flight in a trip to the departure of the flight immediately after it. If there are two or more flights in the trip, the method should return the shortest layover of the trip; otherwise, it should return -1.

For example, assume that the instance variable `flights` of a `Trip` object `vacation` contains the following flight information.

	Departure Time	Arrival Time	Layover (minutes)
Flight 0	11:30 a.m.	12:15 p.m.	} 60
Flight 1	1:15 p.m.	3:45 p.m.	} 15
Flight 2	4:00 p.m.	6:45 p.m.	} 210
Flight 3	10:15 p.m.	11:00 p.m.	

The call `vacation.getShortestLayover()` should return 15.

Complete method `getShortestLayover` below.

```
/** Precondition: the departure time for each flight is later than the arrival time of its
 *                preceding flight
 *  @return the smallest number of minutes between the arrival of a flight and the departure
 *          of the flight immediately after it, if there are two or more flights in the trip;
 *          -1, if there are fewer than two flights in the trip
 */
public int getShortestLayover()
```

Part (a)

```
  /** @return the number of minutes from the departure of the first flight to
the arrival
   *          of the last flight if there are one or more flights in the
trip;
   *          0, if there are no flights in the trip
   */
  public int getDuration()
  {
    if (flights.size() > 0)
    {
      Time start = flights.get(0).getDepartureTime();
      Time end =  flights.get(flights.size() - 1).getArrivalTime();
      return start.minutesUntil(end);   // No need to loop through all
flights
    }
    else
      return 0;
  }
```

Part (b)

```
  /** Precondition: the departure time for each flight is later than the
arrival time of its
   *                preceding flight
   *  @return the smallest number of minutes between the arrival of a flight
and the departure
   *          of the flight immediately after it, if there are two or more
flights in the trip;
   *          -1, if there are fewer than two flights in the trip
   */
  public int getShortestLayover()
  {
    if (flights.size() < 2)
      return -1;
    int shortest = 24 * 60; [1]
    for (int i = 1; i < flights.size(); i++)
    {
      Time arrival = flights.get(i - 1).getArrivalTime();
      Time departure = flights.get(i).getDepartureTime();
      int mins = arrival.minutesUntil(departure);
      if (mins < shortest)
        shortest = mins;
    }
    return shortest;
  }
```

Notes:

1. minutes in a day - the greatest possible value.

Question 1: Flight List

Part A:	getDuration	4 points

+1 handle empty case
- +1/2 check if `flights` is empty
- +1/2 return 0 if empty

+1 access start time
- +1/2 access `flights.get(0)`
- +1/2 correctly call `getDepartureTime` on a flight

+1 access end time
- +1/2 access `flights.get(flights.size()-1)`
- +1/2 correctly call `getArrivalTime` on a flight

+1 calculate and return duration
- +1/2 call `minutesUntil` using `Time` objects
- +1/2 return correct duration (using `minutesUntil`)

Part B:	getShortestLayover	5 points

+1 handle case with 0 or 1 flight
- +1/2 check if `flights.size() < 2`
- +1/2 return -1 in that case

+1 traverse `flights`
- +1/2 correctly access an element of `flights` (in context of loop)
- +1/2 access all elements of `flights` (lose this if index out-of-bounds)

+2 1/2 find shortest layover (in context of loop)
- +1 get layover time between successive flights (using `minutesUntil`)
- +1/2 compare layover time with some previous layover
- +1 correctly identify shortest layover

+1/2 return shortest layover