Question 3: Battery Charger

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

+1 1/2 accesses array elements
- **+1/2** accesses any element of `rateTable`
- **+1/2** accesses an element of `rateTable` using an index derived from `startHour`
- **+1/2** accesses multiple elements of `rateTable` with no out-of-bounds access potential

+2 1/2 accumulates values
- **+1/2** declares and initializes an accumulator
- **+1/2** accumulates values from elements of `rateTable`
- **+1/2** selects values from `rateTable` using an index derived from `startHour` and `chargeTime`
- **+1** determines correct sum of values from `rateTable` based on `startHour` and `chargeTime`

+1 value returned
- **+1/2** returns any nonconstant (derived) value
- **+1/2** returns accumulated value

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

+1/2 invokes `getChargingCost` or replicates functionality with no errors

+1 determines charging cost
- **+1/2** considers **all** potential start times; must include at least 0 … 23
- **+1/2** determines charging cost for potential start times

Note: No penalty here for parameter passed to `getChargingCost` *that violates its preconditions (e.g., 24)*

+1 compares charging costs for two different start times

+1 determines minimum charging cost based on potential start times

Note: Penalty here for using result of call to `getChargingCost` *that violates its preconditions (e.g., 24)*

+1/2 returns start time for minimum charging cost

A3a[1]

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *          Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime  the number of hours the battery needs to be charged
 *          Precondition: chargeTime > 0
 *  @return  the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
    int cost =0;
    for (int x = startHour; x < startHour + chargetime ; x ++)
    {
        cost += rateTable[x % 24];
    }
    return cost;
}
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

A3a2

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return  an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int cheapestHour = 0;

    for (int x = 1; x <= 23; x++)
    {
        if ( getChargingCost(cheapestHour, chargeTime) > getChargingCost(x, chargeTime))
            cheapestHour = x;
    }
    return cheapestHour;
}
```

GO ON TO THE NEXT PAGE.

A3b1

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *          Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *          Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
    int price=0;

    for(int i=0; i<chargeTime; i++)
    {
        price += rateTable[startHour + i];
    }

    return price;

}
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a). A3 b2

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *           Precondition: chargeTime > 0
 *  @return  an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
```

```
{
   int startHour = 0;
   int lowPrice = 500000;
   for(int x = 0; x < 23; x++)
   {
      if( lowPrice > getChargingCost(x, chargeTime)
      {
         lowPrice = getChargingCost(x, chargeTime);
         startHour = x;
      }

   }
   return startHour;
}
```

GO ON TO THE NEXT PAGE.

A3c,

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return  the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
   int totalHours = startHour + chargeTime;
   int i = 1;
   while (totalHours > 24)
   {
      i++;
      totalHours -= 24;
   }

   return i * rateTable[startHour + totalHours];
}
```

Part (b) begins on page 12.

A3 C_2

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
  for(int i = 0; i ≤ 23; i++)
  {
      if (i.getChargingCost < (i - 1).getChargingCost)
            return   i.getChargingCost;
  }
```

Question 3

Overview

This question focused on array traversal, abstraction, and algorithms for accumulation and finding a minimum. Students were provided with the framework of the `BatteryCharger` class that included a private array instance variable with exactly 24 `int` elements, and they were asked to implement two instance methods. The first method, `getChargingCost`, required calculation of a total charging cost given a start time (`startHour`) and a number of hours (`chargeTime`). This could be accomplished by accessing elements of the instance array, beginning with the element at index `startHour`, and traversing in a circular manner (for example, by using the modulus operator), accumulating the values from the array, and returning the sum. The second method, `getChargeStartTime`, required students to return the start time that would allow the battery to be charged at minimal cost. This was best accomplished by invoking the `getChargingCost` method from part (a) for each of the 24 potential start times, comparing the results to determine which achieve the minimum charging cost, and returning that start time.

Sample: A3a
Score: 9

In part (a) the student correctly accesses an element from `rateTable` using an index derived from `startHour`. The loop body demonstrates accessing multiple elements of `rateTable` with no out-of-bounds potential. The student declares and initializes an accumulator and accumulates values from `rateTable` using an index derived from `startHour` and `chargeTime`. A sum of the values from `rateTable` is correctly determined and returned. Part (a) earned all 5 points.

In part (b) the student correctly invokes `getChargingCost`. For each potential start time (0 through 23), the charging cost is determined. The student compares two charging costs and correctly determines the minimum charging cost. The student correctly initializes and returns the start time for the minimum charging cost. Part (b) earned all 4 points.

Sample: A3b
Score: 6

In part (a) the student correctly accesses an element from `rateTable` using an index derived from `startHour`, thus earning the first two ½ points. Since there is an out-of-bounds potential if a wrap around from `rateTable[23]` to `rateTable[0]` is necessary, the student did not earn this ½ point. The student declares and initializes an accumulator and accumulates values selected from `rateTable` based on `startHour` and `chargeTime`, thus earning three ½ points. Since the correct sum will not be determined in the case where a wrap around is necessary, the student did not earn 1 point. A derived accumulated value is returned, thus earning the student the last two ½ points. Part (a) earned 3½ points.

In part (b) the student correctly invokes `getChargingCost`, thus earning the first ½ point. The only start times considered are 0 through 22; consequently, the student did not earn the second ½ point. For each start time considered, the charging cost is determined, thus earning the student ½ point. The student compares two charging costs and earned 1 point. Since the initial value of `lowPrice` is set to a constant, which may be less than all charging costs, the 1 point for determining the minimum charging cost was not earned. The start time for the assumed minimum charging cost is returned, so the last ½ point was earned. Part (b) earned 2½ points.

Question 3 (continued)

Sample: A3c
Score: 3

In part (a) the student accesses an element from `rateTable` using an index derived from `startHour` and earned the first two ½ points. Since the access occurs outside the loop, only one element of `rateTable` is accessed, so the student did not earn the next ½ point. There is no accumulator, so the student failed to earn the 2½ "accumulates value" points. The student earned ½ point because a derived value is returned. But since the return value is not an accumulated value, the student did not earn the final ½ point. Part (a) earned 1½ points.

In part (b) the student did not earn the first ½ point because `getChargingCost` is not invoked correctly. The student does not consider all potential start times (0 through 23) because the `return` statement inside the loop may cause a premature exit. Consequently, the student did not earn the second ½ point. For each start time considered, the student determines a charging cost, thus earning the next ½ point. The student performs a comparison of charging costs and earned 1 point. The student did not earn 1 point for determining a minimum charge time. The student does not return a start time so did not earn the final ½ point. Part (b) earned 1½ points.