## Introduction.

You are doing a summer internship at the townhall of a village next to the sea. They would like to build a small parking lot next to the beach. Before making a final decision, they would like to simulate how the parking will be used and validate the investment. So in this assignment you are helping them by writing a parking simulation program.

## Overall description.

The parking will be represented by an array of a certain size, the cells represent the slots (occupied or empty). Cars can enter the parking lot (if it is not full) and occupy one slot. Vehicles stay for some time and pay $3 per hour with a maximum of $20 per 24 hour. For example a car staying 5 hours will pay $35 = $15. A car staying 10 hours will pay only $20 (because $310 = $30 > $20). And a car staying 30 hours will pay: 20$ for the first 24 hours and $3\*6 = $18 for the remaining 6 hours.

You need to know how many cars will want to park and how long they will stay. Fortunately, the police department keeps some statistics about traffic and tells you that, on any day:

1. No cars will arrive between midnight and 5:59 AM
2. Between 6 AM and 5:59 PM, there will be between 5 and 9 cars arriving every hour. Each possible total count has equal probability (20% that 5 cars arrive, 20% that 6 cars arrive... 20% that 9 cars arrive.
3. Between 6 PM and 8:59 PM, there will be between 1 and 2 cars arriving every hour. Each possible total count has equal probability (50% that 1 car arrives, 50% that 2 cars arrive).
4. No car will arrive between 9 PM and midnight
5. Each car that parks will stay anywhere between 1 and 72 hours with equal probability (1/72 that a car stays 1 hour, 1/72 that a car stays 2 hours,... 1/72 that a car stays 72 hours).

When a car arrives, it parks (if there is a spot available) and stays there between 1 and 72 hours (the time is determined randomly when the car enters the parking). When their time is over, you collect the money and the slot becomes vacant.

## Simulation

### **What we want to know**

The town can build a parking lot of either 50, 100, 150, 200, 250 slots. They would like to know for each possible parking size:

* What is the total money paid by visitors.
* What is the total number of cars that entered (e.g. there was a vacant slot for them).
* What is the total number of cars that were refused (e.g. there was no vacant slot for them).
* What is the total number of hours that cars have been staying on the parking lot.
* What is the average time a car occupied a slot

### **Other consideration**

The simulation should simulate 31 days of parking.  
At the end of the simulation, leave any remaining cars in the parking lot without collecting their money.

### **Time passing by.**

We need to simulate a clock that will tick for the duration of the simulation. The tick will represent one hour. At every tick of the clock:

1. Cars who are supposed to leave at that time do so. You collect their money and their slot becomes free.
2. A number of cars (as defined above) arrive.
3. Each car which can park indicates the time it plans to stay (in actuality your simulation is to choose the length of stay based on the probabilities above)

Time passing will be represented by a **for loop**, the ticks (e.g. hours) being the index of the loop.  
The simulation starts at time 0 (midnight).

### **Parking**

The parking will be represented with a parrallel array structure: two arrays related to each other. The size of the array will be the size of the parking. One array will keep track ot the slots occupancy and the other array of the number of hours the vehicule stayed on the slot (for payement purposes). Let's look at an example with a parking of 3 slots.

The array P represents the parking the array T represents the time.

Suppose a car arrives and needs to stay for x hours. Suppose we assign that car to the slot #y. Then we assign the values P[x] = y and T[x] = y. Then every hour, we decrement T[x]. When T[x] equals 0, the car needs to leave and has to pay P[x].

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Event | P [0] | P [1] | P [2] | T [0] | T [1] | T [2] |
| Initial situation | 0 | 0 | 0 | 0 | 0 | 0 |
| Tick 1. | No car leave (parking empty) | | | | | |
| car #1 arrives (3 hours) car #2 arrives (2 hours) | 3 | 2 | 0 | 3 | 2 | 0 |
| End of hour | 3 | 2 | 0 | 2 | 1 | 0 |
| Tick 2. | No car leave | | | | | |
| car #3 arrives (5 hours) car #4 arrives but is refused (no slot) | 3 | 2 | 5 | 2 | 1 | 5 |
| End of hour | 3 | 2 | 5 | 1 | 0 | 4 |
| Tick 3. | Spot #1 should leave (because T[1] = 0) and pay for P[1] hours. | | | | | |
| After car #2 left | 3 | 0 | 5 | 1 | 0 | 4 |

## Program Specifications

### **Generating a random number:**

Ensuring that random numbers are actually random is a big part of programming (and simulation). Java has some random generators built in, which is very nice. However the downside is that when we use them in programming assignments, we can't really test your program to see if it reliably works since it never produces the same results due to randomness. Instead, we will give you the random generator that you must use (see below) that will allow us to properly test your programs for functionality.

Your program will start by asking the value for the seed (to create the first random number).

##### method: int getRandom (int rnd) must use this code below:

public static int getRandom (int rnd){

long multiplier = 16807;

long modulus = 2147483647;

return (int)((multiplier \* rnd) % modulus);

}

You obtain a random number between 0 (included) and modulus (excluded)  
If you place the seed into a variable rndNb, then the first random number is obtained by doing rndNb = getRandom (rndNb). Then the same call rndNb = getRandom (rndNb) will give you the second number….etc. We have already see that. Remember you use this each time you need a random number - and you don't reset in the middle of the simulation.

### **Constants**

Your program will have the following constant:

|  |  |
| --- | --- |
| final int[] PKG\_SIZE = {25, 30, 35, 40, 45, 50}; | //Parking size to be simulated |
| final int HOURLY\_COST = 3; | //Hourly cost for cars |
| final int DAILY\_COST = 20; | //Maximum a car pays per day |
| final int HOUR\_MAX = 72; | //Max number of hours a car can stay |
| final int HOURS\_SIMULATION = 31 \* 24; | //Number of hours for one simulation |
| final long MODULUS = 2147483647; | //Used to create a random number |
| final long SEED = 314159; | //Used to create a random number |

### **Output**

At the end of the simulation, your program will have to produce a summary of the simulation (see examples)

### **Accumulating data for output**

You will need to accumulate values for being printed at the end. You can use the following variables:

int [ ] totPaid = new int [ ]; // Total paid by cars for the simulation.

int [ ] totIn = new int [ ]; // Total number of cars accepted to enter the parking.

int [ ] totRefused = new int [ ]; // Total number of cars not accepted to enter the parking.

int [ ] totHours = new int [ ]; // Total number of hours that all cars parked.

Each array will have the same size as the array PKG\_SIZE. The index 0 corresponds to a parking size of 50, the index 1 corresponds to a parking size of 100…etc.

## Program Decomposition

We only ask you to implement (and use) the following methods:

* getRandom
* void simulateHour (int ipk, int iHour, int[ ] parking, int[ ] time, int[ ] totPaid, int[ ] totIn, int[ ] totRefused, int[ ] totHours)<\li>

The method simulateHour simulates cars entering and leaving for a parking of size PKG\_SIZE[ipk], at the hour iHour (during the simulation). It should do the following in this order:

1. Find out the cars which are departing at this hour of the day (their time is zero), manage their leaving (free up their spots) and money collection.
2. Find out how many cars are showing up at the gate (random number depending on the hour of the day).
3. Determine how many can be accomodated (based on free spots in the parking.
4. For the ones that can stay, determine the time they stay (randomly) and assign them a spot (the lowest parking spot available).

**Warning:**we are giving you a lot of flexibility in the design of your program. So you can expect that we will pay close attention to the overall decomposition that you implement - passing zyBooks unit test is not a guarantee of a perfect score.

## Sample Run

Result of simulation.

Duration for each parking size: 744 hours (31 Days and 0 hours)

(1) (2) (3) (4) (5) (6)

50 34,507 998 1,578 35,360 35.43

100 65,365 1,875 690 67,920 36.22

150 89,669 2,533 37 92,832 36.65

200 89,029 2,538 0 92,342 36.38

250 89,888 2,556 0 93,299 36.50

(1) : parking size (number of slots)

(2) : total ($) paid during simulation.

(3) : number of cars accepted to park during simulation.

(4) : number of cars refused to park during simulation.

(5) : total number of hours for all cars which parked.

(6) : average number of hours that a car parked.

Process finished with exit code 0

### **Interpretation:**

Of course, one simulation is not enough. But the previous results show that a 50 slot parking is no good: more cars refused than accepted. Likewise, a 250 slot parking is too large - we can accomodate all cars with a 200 slot so why building a larger one? Finally it seems that a 150 slot is adequate. It brings a lot of money and there is little refusal. Simulation is great - but remember the validity of the simulation relies on the validity of the assumptions made!

### **Tips:**

* + Don't rush coding. Think how you are going to decompose your program.
  + Be sure to call getRandom() each time you need a random number.
  + **JavaDoc** - Remember that JavaDoc is required on all programs from PA07 onward.