Assignment 2 is due 11:59 p.m. the night before session 9

**Key Rules:**

* The project must compile and run to be marked otherwise it will receive a grade of 0
* The Assignment is **due 11:59 PM the night before session #9**. Late submissions are not permitted
* Class ParkingTicketTest.java and PoliceOfficerTest.java will be used to mark the majority of your assignment
* This assignment is meant to be done individually, group work will be considered plagiarism.
* All classes must have full Java documentation comments for all public elements( public methods and constructors) using @author , @version, @param, @return tags when applicable and a brief description of the class**. 1 mark will be deducted for each incomplete of missing Java** **documentation tag**
* All classes **must not have magic numbers**, **1 mark will be deducted for each use of a magic number.** Any number other than 0 is considered a magic number

Assignment 2 builds on assignment 1. Fix any errors or bugs in assignment 1 and add the following classes:

**Requirements**

Create a class called **ParkingTicket**. The class has the following:

|  |
| --- |
| Attributes:  ticket number e.g. “VAN1015”  Note that the ticket number must be unique, sequential and created by a private method.  police officer name e.g. “Bob Smith”  police officer's badge number e.g.“RCMP2251”  • The car's license plate number e.g. “1A2B3C”  • Amount of the fine e.g. 150.0 |
| Provide a constructor, the constructor will take parameters in the following order (inputOfficerName, inputOfficerBadgeNumber,carLicenseNumber,fine)  The constructor will call a private applicable method to generate the ticket number.  The following restrictions will be implemented in the constructor **using appropriate method calls.**   * officer name, officer badge number and car license plate number should not be null or an empty String. If the provided value was null, the value will be rejected and the error message “officer name must not be null “will be produced. If the provided value was an empty String the value will be rejected and the error message “officer name must not be an empty String” will be produced. Here are the required error message that should be used when applicable:   “officer name must not be null “  “officer name must not be an empty String"  "badge number must not be null"  "badge number must not be empty String"  "car license plate number must not be null"  "car license plate number must not be empty String"   * fine amount should not be 0 or negative number, if the provided value was 0 the value will be rejected and the error message “fine amount must not be 0” will be produced. If the provided value was negative the value will be rejected and the error message "fine amount must not be negative" will be produced |
| Provide appropriately named accessors for the fields  Police officer name  Police officer badge number  Car license plate number  Ticket number  Fine amount |
| Provide appropriately named mutators for the fields  Police officer name  Police officer badge number  Car license plate number  Fine amount  The mutators will apply the same validation mentioned in the constructor section above |
| Provide Private method to create unique sequential ticket numbers. The method is called from the constructor. The ticket number must be set in the constructor  When the ticket number is created it will not change. For example if the first ticket number was  “VAN1001”, the second ticket will be “VAN1002” etc.  (Hint: use a static variable to create and hold a counter to generate part of the ticket number, increment the static variable and assign the static variable combined with a String prefix to the field ticket number) |
| Provide a method displayDetails(), the method will display the details of the ParkedCar object in the following format:  Ticket Number: VAN1001  Officer Name: Bob Smith  Officer Badge number: RCMP2251  Car License Plate Number: null  Fine amount: 150.0 |
| Provide a static method to set the counter to the value 1000  This method will be used for testing purposes. |

Create a class called PoliceOfficer. The class has the following

|  |
| --- |
| Attributes  officer name e.g. “Bob Smith”  officer's badge number e.g.“RCMP2251”  **These should be the only instance variables in the class** |
| Provide a constructor, the constructor will take parameters in the following order (name, badgeNumber). The following restrictions will be implemented in the constructor **using the appropriate method calls**   * officer name and badge number must not be null or an empty String. If the provided value was null the value will be rejected and an error message “officer name must not be null” or “badge number must not be null” will be produced. If the provided value was an empty String the value will be rejected and an error message “ officer name must not be an empty String” or “badge number must not be an empty String” will be produced |
| Provide appropriately named accessor methods for the fields  Officer name  Badge number |
| Provide appropriately named mutator methods for the fields  Officer name  Badge number  The mutators will apply the same validation mentioned in the constructor section above |
| Provide a private method to check and return if the parkedCar time has expired. the method signature is:    private boolean isParkingTimeExpired(ParkedCar car, ParkingMeter meter) |
| Provide a private method to calculate the fine. The method will check if the parking time was expired and if so it will calculate the fine which is $20 for every hour or part of an hour of the parking time that was not payed for. For example if the car was parked for 61 minutes and that time was not paid for, the fine would be $40. The method signature is  private double calculateFine(ParkedCar car, ParkingMeter meter) |
| Provide a method to issue a parking Ticket. If the car’s parking time was over the time that was paid for create a new ParkingTicket object and then display the tickets details. Finally return the Parking Ticket object. If no ticket was issued return null. The method signature is :  public ParkingTicket issueParkingTicket(ParkedCar car, ParkingMeter meter) |

Create a class called Driver, the class has the following

|  |
| --- |
| Provide a main method, in this method implement the following   * Create a ParkedCar object with valid values. Pass the value 125 to set numberOfMinutesParked field. * Display the details of that object * Create a ParkingMeter object with valid values. Pass the value 60 to set numberOfPurchasedMinures field. * Display the details of that object * Create a PoliceOfficer object with valid values. Invoke method issueParkingTicket on the ParkedCar and ParkingMeter objects that were created previously |

**Instance variable, constructor and method names should match the following names in for the test classes to work properly. Here is a list of the used names**

|  |  |
| --- | --- |
| Class ParkingTicket | |
| Symbolic Constants  Static variable | TICKET\_PREFIX = “VAN”  counter = 1000 |
| Fields | officerName  officerBadgeNumber  ticketNumber  carLicensePlateNumber  fineAmountInCAD |
| Constructor | ParkingTicket(officer name, badge number, car License, fine) |
| Methods | generatTicketNumber  getOfficerName  setOfficerName  getOfficerBadgeNumber  getCarLicensePlateNumber  setCarLicensePlateNumber  getFineAmountInCAD  setFineAmountInCAD  getTicketnumber  displayDetails  resetCounter |
| Class PoliceOfficer | |
| Symbolic constants | ONE\_HOUR\_FINE\_AMOUNT  MINUTES\_IN\_HOUR |
| Fields | officerName  officerBadgeNumber |
| Constructor | PoliceOfficer(name, badgeNumber) |
| Methods | getOfficerName  setOfficerName  getOfficerBadgeNumber  setOfficerBadgeNumber  isParkingTimeExpired  calculateFine  issueParkingTicket |

**Grading**

The classes will be marked as follows:

|  |  |
| --- | --- |
| Class ParkingTicket | 23 marks |
| Class PoliceOfficer | 17 marks |
| Style and Driver class implementation | 15 marks |
| Using methods when applicable | 5 marks |
| Total | 60 marks |

Marks will be given for:

* Comments – appropriate and complete, including Javadoc tags @author, @version, @return and @param.
* Style – In addition to comments style includes following the Java naming convention for classes, variables and methods. It also includes correct indentation and use of symbolic constants instead of magic numbers whenever needed
* Correctness and completeness – code meets the requirements listed above

Create a .zip file containing your entire BlueJ project (zip the folder, not the individual files). Name the .zip file with the assignment number, e.g. “Assign2.zip”. Upload the file to D2L before the cut-off time.