Due Date – Monday, October 13, by 11:59 pm.

Submission

(1) Zip the project folder and submit the zipped file to Canvas.
(2) The zip file must include the following grading items.
- The **source folder "src"** containing the Java files (*.java) [**110 points**]
  (a) Include at least 2 Java packages, details in Project Requirement #12(c), or **lose 3 points**
  (b) Use all lowercase letters for the package names or **lose 2 points**
  (c) Add the **@author** tag in the comment block on top of EVERY Java class and put the name(s) who implemented the Java classes, or **lose 2 points**
- **JUnit test classes**, including the test cases implemented to test the methods below.
  (a) The `isValid()` method of the `Date` class. [**12 points**]
  (b) The `compareTo()` method of the `Vehicle` class. [**9 points**]
  (c) The `surcharge()` method of the `Truck, Sedan,` and `Utility` class. [**12 points**]
- **Class Diagram**, drawing with Lucid; select File/Export/PDF and include the PDF file. [**12 points**]
- The **Javadoc folder** "**doc"** that includes all the .html files generated by Javadoc. [**5 points**]

(3) The submission button on Canvas will disappear after **October 13, 11:59 pm**. Do not wait until the last minute to submit the project. You are responsible for ensuring the project is well-received in Canvas by the due date and time. **You get 0 points** if you do not have a submission on Canvas. **Projects sent through emails or other methods will not be accepted.**

Project Description

The RU operation department is expanding its services to other campuses, now including five campuses: Busch, Livingston, Cook in New Brunswick, Newark in Newark, and Camden in Camden. In addition, more vehicles have been added to the fleet. The RU operation department shall provide the list of vehicles in a text file with the campus information. Employees can book a vehicle parked at different campuses and return it to any campus. For accounting purposes, the department using the vehicles needs to share the cost of running the vehicles. There are additional charges (surcharges) if the campus locations are different for pickup and return. The cost schedule is listed in the table below according to the vehicle types.

Vehicle Type	Charge per mile	Surcharge
truck	$2.99	$39.99 flat
utility	$1.99	$0.30 per mile, maximum of $35.99
sedan	$1.79	$0.25 per mile, maximum of $32.99

The operation department is using the license plate to uniquely identify a vehicle. The license plate contains a fixed length of 6 characters, with the first five characters being digit numbers, and the last character being S, D, or X, representing the vehicle type of Sedan, Utility, and Truck, respectively.

Your team is tasked to extend the functionality of the software developed in Project 1 to include the new campus locations, calculate the charges, and print the cost report for the departments. The software shall remain an interactive system. In addition to the 15 requirements in Project 1, new requirements have been added and listed below.

1. The system shall be able to load a list of vehicles from a text file.
2. The system shall keep track of the vehicle locations for pickup and return.

3. The system shall be able to uniquely identify a vehicle with the 6-character license plate number, in which the first 5 characters are digit numbers, and the last character is S, D, or X, representing sedan, utility, and truck.
4. For each booking, the system shall add the drop-off campus name in addition to the beginning and ending dates of the trip, the vehicle being booked, and the employee who booked it.
5. When a trip is completed and the vehicle is being returned, the system shall update with the new mileage, the vehicle's drop-off location if it is different from the pickup.
6. The system shall calculate the cost of running the vehicles based on the completed trips, including the charge and surcharge based on the mileage run.
7. The system shall generate and display the cost report ordered by the departments, including the charges and surcharges for each trip, and the department total for all charges.
8. The system shall display the list of vehicles ordered by the city location of the campus, then the car make, and the date obtained.
9. The system shall display the list of bookings ordered by the city location of the campus, then the license plate number, and the beginning date.

Your software must support the new commands listed below and modify the commands from Project 1 if necessary.

(a) **L** command to load the vehicles from the text file "vehicles.txt" and add the vehicles to the fleet. Put the text file under your top-level project folder. Loading the file more than once should not add the same vehicle more than once. You can assume all the vehicle information in the text file is valid.

(b) **A** command to **add a Sedan, Utility,** or **Truck** object to the fleet, with the campus where the vehicle is currently parked. After the A command, the user shall enter the vehicle information in the following sequence: the license plate, date obtained, make, current reading on the odometer, and the campus name. Below is an example of an activity for adding a new vehicle.

```
A 58719D 1/31/2017 TOYOTA 87120 Livingston
```

In addition to the conditions in Project 1, you must also handle the following conditions.

1. Ensure the license plate is a 6-character number, five digits + one uppercase letter, either S, D, or X.
2. Check if the given campus name is valid.
3. Handle exceptions of missing data tokens or incorrect number format of the mileage.

(c) **D** command to **remove a vehicle** from the fleet; same as Project 1, however, must handle exceptions.

(d) **B** command to **book a vehicle** for a business trip with a drop-off campus name. For example,

```
B 10/25/2025 10/25/2025 58718D KAUR Busch.
```

You must check if the campus name is valid and handle exceptions.

(e) **C** command to **cancel an existing booking** is the same as Project 1.

(f) **R** command to **return the vehicle** when a trip is completed. In addition to the conditions in Project 1, you must update the vehicle with the new mileage and drop-off campus location if it is different from the pickup, which is the location where the trip begins.

(g) **P** commands**,** to display the information, sorted with different keys. If the keys for sorting are the same, the order of output for those does not matter.

- ✓ **PF** command to display the list of vehicles, ordered by the city location of the campus, then by make, then by date obtained.
- ✓ **PR** command to display the list of bookings, ordered by the city location of the campus, then by license plate, then by beginning dates.
- ✓ **PD** and **PT** commands are the same as Project 1.

- ✓ **PC** command to display the cost report, ordered by department, including the charge and surcharge for each trip, and the department total for all charges.

(h) **Q** command is the same as Project 1 to stop running the software system.

Project Requirement

1. You MUST follow the Coding Standard posted on Canvas under Modules/Week #1. **You will lose points** if you violate the rules listed in the coding standard.
2. You are required to follow the Academic Integrity Policy. See the **Additional Note #14** in the syllabus posted on Canvas. If your team uses a repository hosted on a public website, you MUST set the repository to private. Setting it to public is considered a violation of the academic integrity policy. The consequences of violation of the Academic Integrity Policy are: **(i) all parties involved receive 0 (zero) on the project, (ii) the violation is reported, and (iii) a record on your file of this violation.**
3. Test cases for grading are included in the file **Project2TestCases.txt.** The data for the test cases are fictional for testing purposes. DO NOT change the spacing or sequence in the file; use it as is. The graders will run your project by copying all the test cases in **Project2TestCases.txt** and pasting them to the terminal. Enter the test cases from the file in the same order to test your project.
4. The associated output generated from the test cases is in **Project2Output.txt**. You must match the output provided. Your project should be able to ignore the empty lines between the test cases in **Project2TestCases.txt**. You MUST use the **Scanner class** to read the command lines from the terminal (**System.in**); DO NOT read it as an external file, or you will **lose 5 points.** The only exception is reading the "**vehicles.txt**" for loading the vehicles.
5. The graders will compare your output with the expected output in **Project2Output.txt**. You will **lose 2 points** for each output not matching the expected output, OR for each exception not caught, which causes your software to terminate abnormally.
6. Each source file (.java file) can only include one public Java class; the file name is the same as the Java class name, or **lose 2 points**.
7. Your program MUST handle bad commands; **or lose 2 points** for each bad command not handled.
8. You CAN import the **Scanner**, **StringTokenizer, Calendar, DecimalFormat, File, Iterator**, **Exception** classes, and **your own packages**. Importing other Java library classes not listed above will result in a **loss of 5 points** for each Java class/package imported, with a **maximum loss of 10 points**.
9. You CANNOT use the Java library class **ArrayList** anywhere in the project, OR use any Java library classes from the Java Collections, or **you will get 0 points for this project**!
10. Be specific when importing Java library classes and DO NOT import unnecessary classes or the whole package. For example, `import java.util.*;` this will import all classes in the `java.util` package. You will **lose 2 points** for using the asterisk "*" to include all the Java classes in the `java.util` package, or other Java packages, with a **maximum of losing 4 points.**
11. You must define the **inheritance relationships** below and reuse your code as much as possible. You will **lose 5 points** for each class missing, or an incorrect inheritance relationship, or having duplicate code segments, or not following the requirements listed below.

    (a) Change the **Vehicle class** to an **abstract** class and add an instance variable and two abstract methods listed below. Change the modifiers to `protected` for all instance variables.

```
protected Campus campus; //Campus is an enum class defining the campuses.

public abstract double charge(int mileageUsed); //charge per mile used
public abstract double surcharge(int mileageUsed, boolean surcharge);
```

    The output for a vehicle should be updated to the format below.

```
80671S[HONDA:utility] 9/9/2022 [mileage:33220] [Livingston:New Brunswick]
```

(b) Create **Sedan, Utility,** and **Truck classes** to **extend the Vehicle class**. Define specific constants and override the `charge()` and `surcharge()` methods to calculate the charges correctly. **Polymorphism** is required, i.e., dynamic binding for the `equals()`, `compareTo()`, `toString()`, `charge()`, and `surcharge()` methods. If you use if/else or switch/case statements to check the vehicle types to determine the charges, you will **lose 10 points.**

12. **Refactoring**.

(a) The software developed in Project 1 includes three collection classes: **Fleet**, **Reservation**, and **TripList**, where the code is similar. Generalize the code and reuse the code by defining a generic class **List<E>.** This class works very similarly to the Java ArrayList class. You must implement the API below, or **lose 10 points** for not defining this class or not using this class. Using the ArrayList from the `java.util` will **result in 0 points for this project**.

```
public class List<E> implements Iterable<E> {
    private E[] objects; //E is the name for the generic type
    private int size;

    public List() { } //new an array type-casted to E with a capacity of 4.
    private int  find(E e) {} //return -1 if not found
    private void grow() {}    //grow the size of the array by 4
    public  boolean contains(E e) {}
    public  void add(E e) {}
    public  void remove(E e) {}
    public  boolean isEmpty() {}
    public  int     size() {}
    public  Iterator<E> iterator() {} //traversing the list using for each
    public  E    get(int index) {}    //return the object at the index
    public  void set(int index, E e) {} //put object e at the index
    public  int  indexOf(E e) {} //return index of object e, or return -1
    //private inner class for the iterator to work properly
    private class ListIterator<E> implements Iterator<E> {
        int current = 0; //current index when traversing the list (array)
        public boolean hasNext(){} //if it's empty or at the end of the array
        public E next(){} //return the next object in the list
    }
}
```

- You cannot change or add the instance variables, cannot change the method signatures listed above, or **lose 2 points** for each violation.
- DO NOT add any print methods or sorting methods, or you will **lose 5 points**
- You cannot add other constructors or additional methods, or **lose 2 points** for each violation.
- **Fleet, Reservation,** and **TripList** must extend this class**,** or **lose 5 points** for each violation**,** a maximum of **losing 10 points.**

(b) Extract the code for sorting in Project 1 to create a utility class, **Sort class**, to handle the sorting on different types of lists. You should define only static methods for sorting. You will **lose 5 points** if this class is missing.

(c) Create a utility package, "`util`," under the `src` folder to include `Date`, `List<E>`, and `Sort` classes, or **lose 3 points.**

13. You CANNOT use **System.in** or **System.out** statements in ALL classes, EXCEPT the user interface class `Frontend.java`, the testbed `main()`, and the `print()` methods in the collection classes, or **lose 2 points** for each violation, with a **maximum of losing 10 points.** You must always add the `@Override` tag for

overriding methods or **lose 2 points** for each violation. You should define necessary constant names in uppercase letters and **NOT** use MAGIC numbers, or **you will lose points** listed in the Coding Standard.

14. You MUST add or change the Java classes listed below, or **lose 5 points** for each class missing or NOT used.
    (a) Create an **enum class Campus** to define the five campuses with an additional attribute identifying the city names of the campuses. `private final String city;`
    (b) Modify the **Trip** class to include an additional instance variable: `private boolean surcharge;` to flag the surcharge for each completed trip. The output for a completed trip should follow the format below, where the "**" marks a surcharge.

    ```
    Trip completed: 58718D 10/27/2025 ~ 10/27/2025 mileage(old): 64390 mileage(new): 64500 mileage(used): 110 [dropped off: Cook**] [picked up: Busch]
    ```

    (c) Modify the **Booking** class to include an additional instance variable: `private Campus dropoff;` to keep track of the drop-off campus for each booking. The output for a booking should follow the format below.

    ```
    71707X:TOYOTA [Cook:New Brunswick] 11/3/2025 ~ 11/3/2025 [drop off:Cook] [RAMESH]
    ```

    (d) Add a `public void printCharges()` method to the **TripList** class to print the cost report by department.
    (e) Add a `public int load(Scanner scanner)` method to the **Fleet** class to load the vehicles from the text file "vehicles.txt" and return the number of vehicles loaded to the list.
    (f) Modify the **Frontend class** to include the new commands and update the existing commands. You must define a **run()** method with a while loop to continuously read the transactions until a "Q" command is entered. You will **lose 5 points** if the **run()** method is missing. You MUST keep this method **under 50 lines** for readability, or you will **lose 3 points**. You can define necessary instance variables and private helper methods to handle each command.
    (g) Add the **RunProject2** class to run your software. The graders will run this class to grade your project.

```
public class RunProject2 {
    public static void main(String [] args) {
        new Frontend().run();
    }
}
```

15. **JUnit test.** Create three JUnit test classes and write code to test the following methods.
    (a) `isValid()` in the `Date` class. Reuse the code in the testbed main() from Project 1. (12 points)
    - **four** invalid test cases that return false
    - **two** valid test cases that return true

    (b) `compareTo()` method in the `Vehicle` class. Subclasses inherit this method. Instantiate three different types of objects: **Sedan, Utility,** and **Truck** for testing the method. (9 points)
    - **three** test cases, each comparing two Sedan, two Utility, and two Truck objects, that return -1,
    - **three** test cases, each comparing two Sedan, two Utility, and two Truck objects, that return 1, and
    - **three** test cases, each comparing two Sedan, two Utility, and two Truck objects, that return 0

    (c) `surcharge()` methods in **Sedan**, **Utility**, and **Truck** classes. (12 points)
    - **three** test cases for no surcharges for Sedan, Utility, and Truck.
    - **two** test cases for the surcharges less than the maximum surcharges for Sedan and Utility.
    - **two** test cases for the surcharges greater than the maximum surcharges for Sedan and Utility.
    - **one** test case for imposing a surcharge for a Truck object.

16. **Class Diagram.** Create a class diagram to document your software structure for Project 2. **Hand-drawing the diagram or using a plug-in in IntelliJ to generate the diagram is unacceptable, and you will lose 12 points**. You must use **Lucid** to draw the class diagram and include the Java classes you developed for this project. Export the diagram as a PDF and include it in the zip file to be submitted.

(a) For each Java class, include ONLY the instance variables and public methods; DO NOT include the constants and private methods. DO NOT include the JUnit test classes and Java classes imported.
(b) Show the relationships between ALL the Java classes you developed; DO NOT show the packages.

17. You must **generate the Javadoc** after you properly comment your code. Set the scope to 'private" so your Javadoc will include the documentation for the private data members, constructors, and private and public methods of all Java classes. Generate the Javadoc in a single folder and include it in the zip file to be submitted to Canvas. Look for the index.html and open it to ensure the Javadoc is generated properly. You will lose points if any description in the Javadoc is empty. You will **lose 5 points** for not including the Javadoc.