Design Document and Test Plan

Name of team members who collaborated on the design and test plan:

1. Name (*first last*): Brianna Cherry

Name of programming for which you submit this document: **Parking Permits**

# UML Class Diagrams

|  |
| --- |
| Employee |
| - NAME: string - EMAIL: string  - ADDRESS: string  - DEPT: string  -IF\_VET: string |
| + Employee() + Employee(dept: string, if\_vet: bool)  + setName(): void  + setEmail(): void  + setAddress(): void  + setDept(dept: string): void  + setVeteran(if\_vet: bool): void  + getName(): const string  + getEmail(): const string  + getAddress(): const string  + getDept(): const string  + getVeteran(): const string |

|  |
| --- |
| Student |
| - NAME: string - EMAIL: string  - ADDRESS: string  - MAJOR: string  - STATE: string  -EMPLOYEE: bool |
| + Student() + student(major: string, state: string, if\_employee: bool)  + setName(): void  + setEmail(): void  + setAddress(): void  + setMajor(major: string): void  + setEmployee(if\_employee: bool): void  + setState(state: string): void  + getName(): const string  + getEmail(): const string  + getAddress(): const string  + getMajor(): const string  + GetState(): const string  + getEmployee(): const string |

|  |
| --- |
| Visitor |
| - NAME: string - EMAIL: string  - ADDRESS: string  - HS: bool  - SPEAKER: bool |
| + Visitor() + Visitor(speaker: bool, hs: bool)  + setName(): void  + setEmail(): void  + setAddress(): void  + setHS(hs: bool): void  + setSpeaker(speaker: bool): void  + getName(): const string  + getEmail(): const string  + getAddress(): const string  + getSpeaker(): const string  + getHS(): const string |

|  |
| --- |
| Construction |
| - NAME: string - EMAIL: string  - ADDRESS: string  - COMPANY: string  - IF\_VET: string |
| + Construction() + Construction(company: string, if\_vet: bool)  + setName(): void  + setEmail(): void  + setAddress(): void  + setCompany(company: string): void  + setVeteran(if\_vet: bool): void  + getName(): const string  + getEmail(): const string  + getAddress(): const string  + getCompany(): const string  + getVeteran(): const string |

|  |
| --- |
| Car |
| - MAKE: string - MODEL: string  - YEAR: int  - PLATE: string  - CLASSIC: bool  - LOW\_EMISSION: bool |
| + Car() + Car(plate\_no: string, classic: bool, low\_emission: bool)  + setMake(): void  + setModel(): void  + setYear(): void  + setPlate(plate: string): void  + setClassic(classic: bool): void  + setEmission(low\_emission: bool) : void  + getMake(): const string  + getModel(): const string  + getYear(): const int  + getPlate(): const string  + getClassic(): const string  + getEmission(): const string |

|  |
| --- |
| Motorcycle |
| - MAKE: string - MODEL: string  - YEAR: int  - COLOR: string  - IF\_MOPED: bool |
| + Motorcycle() + Motorcycle(color: string, if\_moped: bool)  + setMake(): void  + setModel(): void  + setYear(): void  + setColor(color: string): void  + setMoped(if\_moped: bool): void  + getMake(): const string  + getModel(): const string  + getYear(): const int  + getColor(): const string  + getMoped(): const string |

|  |
| --- |
| Scooter |
| - MAKE: string - MODEL: string  - YEAR: int  - COLOR: string  - WHEELS: int |
| + Scooter() + Scooter(color: string, wheels: int)  + setMake(): void  + setModel(): void  + setYear(): void  + setColor(color: string): void  + setWheels(wheels: int): void  + getMake(): const string  + getModel(): const string  + getYear(): const int  + getColor(): const string  + getWheels(): const int |

|  |
| --- |
| Truck |
| - MAKE: string - MODEL: string  - YEAR: int  - PLATE: string  - CLASSIC: bool  - HITCH: bool |
| + Truck() + Truck(plate\_no: string, classic: bool, hitch: bool)  + setMake(): void  + setModel(): void  + setYear(): void  + setPlate(plate: string): void  + setClassic(classic: bool): void  + setHitch(hitch: bool) : void  + getMake(): const string  + getModel(): const string  + getYear(): const int  + getPlate(): const string  + getClassic(): const string  + getHitch(): const string |

|  |
| --- |
| Invoice |
| - PRICE: double - PERSON\_TYPE: int  - VEHICLE\_TYPE: int  - generalPersonPrinter(): void  - studentPrinter(student: Student& const) : void  - visitorPrinter(visitor: Visitor& const) : void  - constructionPrinter(construction: Construction& const) : void  - employeePrinter(employee: Employee& const) : void  - generalVehiclePrinter(): void  - carPrinter(car: Car& const) : void  - motorcyclePrinter(motorcycle: Motorcycle& const) : void  - truckPrinter(truck: Truck& const) : void  - scooterPrinter(scooter: Scooter& const) : void |
| + Invoice() + Invoice(person: int, vehicle: int)  + invoicePrinter(car: Car& const, motorcycle: Motorcycle& const, truck: Truck& const, scooter: Scooter& const, student: Student& const, visitor: Visitor& const, employee: Employee& const, construction: Construction& const): void  + priceFinder(car: Car& const, motorcycle: Motorcycle& const, truck: Truck& const, scooter: Scooter& const, student: Student& const, visitor: Visitor& const, employee: Employee& const, construction: Construction& const): double |

# Pseudocode

*(See Ch. 1.6 in our textbook for an example of how to write detailed pseudocode)*

Initialize variables and class objects that will be used by the program regardless of user input:

* person to store the type of user, vehicle to store type of vehicle, and a class object for each person and vehicle class: student, visitor, employee, construction, truck, car, and motorcycle.

Display a menu of choices to the user and tell them to enter one of four “person” options. If the user doesn’t enter a valid input, prompt them to enter again.

Prompt the user to enter the information common to all “people” classes: name, address, and email; then take input for each one. These are stored in global variables, so setters don’t need to be used.

If user choses student:

* Create variables to store attributes UNIQUE to student class – major (string), state (string), if\_employee (bool). Prompt user to enter each attribute, then call a setter for the student class to set the variables in the class object initialized at the beginning.

If user choses employee:

* Create variables to store attributes UNIQUE to employee class—department (string) and if\_veteran (bool). Prompt user to enter each attribute, then call a setter for the employee class to set the variable in the class object initialized at the beginning.

If user choses construction:

* Create variables to store attributes UNIQUE to employee class—company (string) and if\_veteran (bool). Prompt user to enter each attribute, then call each setter for the construction class to set the variables in the class object initialized at the beginning.

If user choses visitor:

* Create variables to store attributes UNIQUE to visitor class—if\_hs (bool) and if\_speaker (bool). Prompt user to enter each attribute, then call a setter for the visitor class to set the variabless in the class object initialized at the beginning.

Now, do the same for the vehicle class objects and information:

Display a menu of choices to the user and tell them to enter one of four “vehicle” options. If the user doesn’t enter a valid input, prompt them to enter again.

Prompt the user to enter the information common to all “vehicle” classes: make, model, and year, then take input for each one. These are stored in global variables, so setters don’t need to be used.

If user choses car:

* Create variables to store attributes UNIQUE to car class – plate (string), if\_classic (string), low\_emission (bool). Prompt user to enter each attribute, then call a setter for the car class to set the variables in the class object initialized at the beginning.

If user choses motorcycle:

* Create variables to store attributes UNIQUE to motorcycle class—color (string) and if\_moped (bool). Prompt user to enter each attribute, then call a setter for the motorcycle class to set the variables in the class object initialized at the beginning.

If user choses truck:

* Create variables to store attributes UNIQUE to truck class—plate (string), if\_classic (bool), and if\_hitch (bool). Prompt user to enter each attribute, then call each setter for the truck class to set the variables in the class object initialized at the beginning.

If user choses scooter:

* Create variables to store attributes UNIQUE to scooter class—color (string) and wheels (int). Prompt user to enter each attribute, then call a setter for the scooter class to set the variables in the class object initialized at the beginning.

After all input has been entered, create an Invoice class object and call the invoicePrinter() function for that object to print the user’s input. To display the final price of the parking pass, call the priceFinder() function for your Invoice object, which will return the final price after considering person and vehicle type, special charges, and discounts.

# Test Plan

*(See Ch. 5.13 in our textbook for an example of how to write a test plan)*

|  |  |  |  |
| --- | --- | --- | --- |
| **Test #** | **Purpose** | **Input** | **Expected Output** |
| 1 | Student, car input | inputTest1.txt | outputTest1.txt |
| 2 | Speaker, truck input | inputTest2.txt | outputTest2.txt |
| 3 | Visitor, motorcycle input | inputTest3.txt | outputTest3.txt |
| 4 |  |  |  |
| 5 |  |  |  |
| … | *(Feel free to add more test cases)* |  |  |