Parking Ticket Simulator

Stephen Gerkin
November 16, 2019
CIST 2362 CRN 22282

Program Description

Demonstrate class interoperability by creating four classes that communicate. These classes will be a **ParkedCar**, **ParkingMeter**, **PoliceOfficer**, and **ParkingMeter**. Each class is responsible for knowing information relevant to what the class represents. **PoliceOfficer** is responsible for knowing if a **ParkedCar** has parked longer than the minutes purchased with the **ParkingMeter**. It is also responsible for creating **ParkingTicket** objects that must know the information contained within the **ParkedCar**, **ParkingMeter**, and the issuing **PoliceOfficer**. This operability should be demonstrated.

Programming Strategy:

Each class will be created with fields to know all relevant information. The **PoliceOfficer** will have a function that receives **ParkedCar** and **ParkingMeter** objects to evaluate. It will have another function that will construct a **ParkingTicket** object and return a pointer to the created object. It will be the responsibility of the object or function that calls this method to free any memory associated with creation of this object. A **Simulator** class will be created for demonstrating these classes by constructing a **PoliceOfficer** from user input and then allow the user to inspect **ParkedCar** and **ParkingMeter** objects that will be created dynamically. This ability to inspect will loop and create new objects with different values for time parked and the amount of time purchased for parking. The user will be able to attempt to issue **ParkingTicket** objects for any violation. If a **ParkingTicket** cannot be created, an exception will be caught and handled.

Class Documentation

ParkingMeter

ParkingMeter represents a parking meter that would be on a street.

Minutes for parking are purchased at the parking meter for allowing cars to park on the street. This is essentially a wrapper class for an integer value. It can only be instantiated with an existing value for the minutes parked

Member Variables

Data Type	Name	Description
int	minutesPurchased	The number of minutes purchased for this **ParkingMeter**.

Constructors and Destructors

ParkingMeter::ParkingMeter () `[delete]`

ParkingMeter::ParkingMeter (int *minutesPurchased*)

Default constructor.

Initializes the amount of time purchased for the meter. Minutes purchased must be an integer value greater than 0.

Parameters

minutesPurchased	The number of minutes purchased.

Exceptions

std::runtime_error	If minutes purchased is less than 0.

Member Function Documentation

Only getter for minutesPurchased.

ParkedCar

ParkedCar represents a car parked on the street.

It stores information about the car and provides getters and setters for this information. It additionally stores the amount of time the car has been parked.

Member Variables

Data Type	Name	Description
int	minutesParked	The amount of time a car has been parked.
std::string	make	The make of the car (i.e. “Chevy”).
std::string	model	The model of the car (i.e. “Sonic”).
std::string	color	The color of the car (i.e. “Grey”).
std::string	licenseNumber	The license plate number of the car.

Constructors and Destructors

ParkedCar::ParkedCar () `[delete]`

ParkedCar::ParkedCar (int *minutesParked*) `[explicit]`

Default constructor.

Requires the amount of time that a car has been parked.

Parameters

minutesParked	The amount of time a car has been parked.

Exceptions

std::runtime_error	if minutes parked is less than 0.

ParkedCar::ParkedCar (int *minutesParked*, const std::string & *make*, const std::string & *model*, const std::string & *color*, const std::string & *licenseNumber*)

Full argument constructor.

Requires all fields of ParkedCar to be set. Used mainly for debugging user interface, allowing for bypassing entry of car information during program run.

Parameters

minutesParked	The amount of time a car has been parked.
make	The make of the car (i.e. Chevy).
model	The model of the car (i.e. Sonic).
color	The color of the car (i.e. Grey).
licenseNumber	The license plate number of the car.

Exceptions

std::runtime_error	if minutes parked is less than 0.

ParkedCar::~ParkedCar () `[default]`

Member Function Documentation

Only getters and setters.

PoliceOfficer

PoliceOfficer encapsulates information about an officer issuing parking tickets.

The class is responsible for knowing the name of the officer, their badge number, and creating ParkingTicket objects.

Member Variables

Data Type	Name	Description
std::string	name	The name of the **PoliceOfficer**.
std::string	badgeNumber	The badge number for the **PoliceOfficer**.

Constructors and Destructors

PoliceOfficer::PoliceOfficer () `[delete]`

PoliceOfficer::PoliceOfficer (const std::string & *name*, const std::string & *badgeNumber*)

Default constructor requires information about a police officer to construct.

Parameters

name	The name of the officer.
badgeNumber	The badge number of the officer.

PoliceOfficer::~PoliceOfficer () `[default]`

Member Function Documentation

**Getters and setters omitted*

bool PoliceOfficer::carTimeExpired (const ParkedCar & *car*, const ParkingMeter & *meter*)

Determines if a car has been parked longer than the time purchased for the meter.

Parameters

car	The car we are inspecting.
meter	The meter the car is parked next to.

Returns
TRUE if a car is parked beyond the amount of time purchased.

ParkingTicket * PoliceOfficer::issueTicket (ParkedCar * *car*, ParkingMeter * *meter*)

Creates a new **ParkingTicket** object and returns a pointer to the object using the information provided.

Parameters

car	The car we are to ticket.
meter	The meter the car is parked next to.

Returns
A pointer to a new **ParkingTicket** object.

ParkingTicket

ParkingTicket encapsulates information about a **ParkingTicket**.

It stores information about a car that is being ticketed and the meter that the car is parked near. It provides methods for determining the amount of a fine that is to be given. Each time a ticket is created, a new ID is given to the ticket, pulled from a file that increments the value to keep track of the tickets issued. This provides some more realism for a ticket as each one should have a unique identifier. Methods are additionally provided to stream the information to an ostream operator, allowing ticket information to be saved to file or displayed to the screen. Lastly, each time a ticket is created, the current date and time is recorded and stored with the ticket. Only getter methods are provided as the information required by a ticket should only be supplied during construction. If an "error" is made entering information for a ticket, a new ticket should instead be created and the old one should be voided but not deleted to maintain proper record keeping.

Constant Attributes

Data Type	Name	Description
double	FINE_FIRST_HOUR	The base amount of fine for the first hour of a violation.
double	FINE_ADDITIONAL	The additional amount of fine for any subsequent hours after the first.

Member Variables

Data Type	Name	Description
unsigned int	ticketId	The ID number for the current **ParkingTicket** being issued.
int	timeInfraction	The number of minutes of a parking violation.
ParkedCar	car	The **ParkedCar** to be ticketed.
ParkingMeter	meter	The **ParkingMeter** the car is parked beside.
PoliceOfficer	officer	The issuing **PoliceOfficer**.
CurrentTimeStamp	timeStamp	A **CurrentTimeStamp** of the date and time the **ParkingTicket** is being issued.

Constructors and Destructors

ParkingTicket::ParkingTicket () `[delete]`

ParkingTicket::ParkingTicket (const ParkedCar & *car*, const ParkingMeter & *meter*, const PoliceOfficer & *officer*)

Default constructor requires information about a car, the information from a parking meter, and the information for the issuing officer in order to be instantiated.

Parameters

car	The car that is in violation.
meter	The parking meter the car is parked near.
officer	The officer issuing the ticket.

ParkingTicket::~ParkingTicket ()

Frees memory we use for storing the time stamp.

Member Function Documentation

**Some getters and setters omitted*

double ParkingTicket::getFine () const

Calculates and returns the amount of fine for the ticket.

Returns
The amount of the fine.

void ParkingTicket::setTicketId () [private]

Sets the ticketId number we use for this ticket.

Attempts to set it from a file that is incremented each time a **ParkingTicket** object is created, but will set the ticketId to -1 as a flag if there is a problem processing the file.

unsigned int ParkingTicket::getTicketNumberFromFile () [private]

Gets the current ticketId from our file that maintains this information and then updates the file with the new number to use at the next call.

Returns
The number we receive from the file.

void ParkingTicket::overrideTicketId (long *idNumber*)

Method to override the ticket number that was set during creation.

Should ideally only be used if there was an issue getting the ticketId from file.

Parameters

idNumber	The ID number we wish to set for this particular ticket.

unsigned int ParkingTicket::readIdFromFile (std::ifstream & *stream*) [private]

Reads the next ticketId from file.

Parameters

stream	The stream for the file we are reading from.

Returns
The next ticketId number.

void ParkingTicket::setNextTicketIdInFile (unsigned int *currentId*, const std::string & *fileLoc*) [private]

Saves the next ID number to the file used for maintaining ticket IDs.

Parameters

currentId	The ID we used for the current ticket.
fileLoc	The file name where information is to be stored.

void ParkingTicket::writeNewIdToFile (unsigned int *id*, std::ofstream & *stream*) `[private]`

Writes a new ID number to file.

void ParkingTicket::printTicket (std::ostream & *stream*)

Streams the ticket to an ostream object (i.e.

cout or file output).

Parameters

stream	The stream we are to put the ticket into.

CurrentTimeStamp

CurrentTimeStamp encapsulates a ctime struct with the date and time of the (local) system time at the time of instantiation.

It provides methods for getting both the date and time of the time stamp created. This class is used for notating the time and date a **ParkingTicket** is issued.

Member Variables

Data Type	Name	Description
int	year	The current year.
int	month	The current month.
int	day	The current day.
int	hour	The current hour.
int	minute	The current minute.
int	second	The current second.

Constructors and Destructors

CurrentTimeStamp::CurrentTimeStamp ()

Constructor gets the current local system time and stores it in the tm_now struct.

The individual values from the struct are then stored in their respective member variables.

CurrentTimeStamp::~CurrentTimeStamp () `[default]`

Member Function Documentation

std::string CurrentTimeStamp::getDateString ()

Gets a formatted string of the **CurrentTimeStamp** date in the form MM/DD/YYYY.

Returns

A string containing the date in form MM/DD/YYYY.

void CurrentTimeStamp::getTimeNow () `[private]`

Gets the current local system time and puts it in the struct that holds this information.

std::string CurrentTimeStamp::getTimeString ()

Gets a formatted string of the **CurrentTimeStamp** time in the form HH:MM:SS.

Returns

A string containing the time in form HH:MM:SS.

Simulator

Simulation class for demonstrating program functionality.

It displays information to the user and gets input for issuing tickets. As the user moves from car to car, a new **ParkedCar** and **ParkingMeter** is instantiated with a random integer value for the amount of time parked and the amount of time purchased for the meter. The user can then inspect both the car and the meter to determine if a ticket should be written. The process loops until the user enters input indicating the desire to quit, at which time an exception is thrown to signal the end of the simulation. Additionally, each time a ticket is written by the user, it is appended to a file that contains all the tickets written across any runs of the program.

Constant Attributes

Data Type	Name	Description
std::string	QUIT	String to signal user wants to exit.
std::string	FILE_NAME	The file we use for the ticket book.
int	MAX_MINUTES	The maximum minutes we use for time for **ParkedCar** or **ParkingMeter**.

Member Variables

Data Type	Name	Description
PoliceOfficer *	officer	A pointer to the **PoliceOfficer** object we use for simulation
ParkedCar *	car	A pointer to a **ParkedCar** object created dynamically through the simulation.
ParkingMeter *	meter	A pointer to a **ParkingMeter** object created dynamically through the simulation.

Member Class

QuitException (*inherits std::exception)*

QuitException is for signaling the user's desire to exit the simulation.

Constructors and Destructors

Simulator::Simulator () `[default]`

Simulator::~Simulator ()

Destructor frees the memory we use for the objects in the simulation if not already freed.

Member Function Documentation

void Simulator::constructOfficer () `[private]`

Builds the **PoliceOfficer** we are going to use for the simulation from user input.

void Simulator::displayWelcomeMsg () `[private]`

Displays information about how the program will work to the user.

int Simulator::getIntegerInput () `[private]`

For getting an integer value without any additional information.

Returns

User input as an integer.

int Simulator::getIntegerInput (const std::string & *inputType*) `[private]`

Gets an string input from the user and returns it as an integer (if possible).

Parameters

inputType	Information about the input we want to receive.

Returns

User input as an integer.

int Simulator::getMenuSelection (int *maxMenu*) `[private]`

Gets a menu selection from the user.

This method will recur if bad input is received.

Parameters

maxMenu	The max menu number.

Returns

The integer menu selection of the user.

int Simulator::getRandomInt () `[private]`

Gets a random integer value from 0 to MAX_MINUTES (not inclusive).

Returns

A random integer value.

std::string Simulator::getStringInput (const std::string & *inputMsg*) [private]

Gets string input from a user and sets display formatting.

Parameters

inputMsg	Information about what value we want to receive from the user.

Returns

the user input as a string.

void Simulator::inspectCar () [private]

Inspects our ParkedCar object to determine how long it has been parked and asks what we want to do with this information.

void Simulator::inspectMeter () [private]

Inspects a **ParkingMeter** object to see how much time has been purchased for this meter.

void Simulator::moveAlong () [private]

Displays a message that we move along in our patrol.

void Simulator::patrol () [private]

Method that is called infinitely (until quit) that allows us to interact with **ParkedCar** and **ParkingMeter** objects.

It instantiates a new **ParkedCar** and **ParkingMeter** each time the method is called, allowing the user to have different values for moving through the street. The objects are deleted once after all methods are done and the stack unwinds.

void Simulator::recordCarInformation () [private]

Records the information about a car and asks how we want to proceed.

void Simulator::start ()

For starting the simulation.

Initializes the seed we will use for random values when creating **ParkedCar** and **ParkingMeter** objects.

void Simulator::successfulTicket (ParkingTicket * *ticket*) [private]

Displays the ticket that we have written to the user and saves it to the ticket book.

Parameters

ticket	The ticket we have created.

void Simulator::unsuccessfulTicketAttempt () [private]

Displays a message to user that a ticket cannot be created and we should move on.

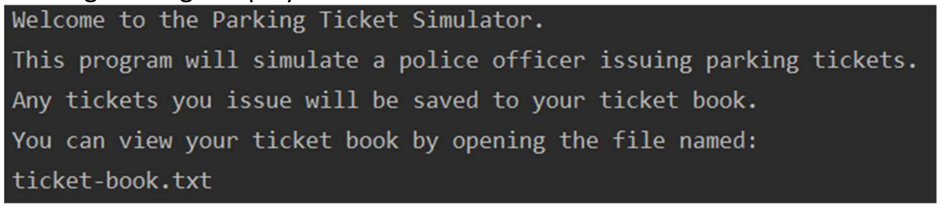
void Simulator::writeTicket () [private]

Attempts to write a ticket for our **ParkedCar**.

If we do not have the information about the car recorded, we will be required to enter the information about the car before we can write a ticket. If the car is not illegally parked, we will be forced to move along to the next car.

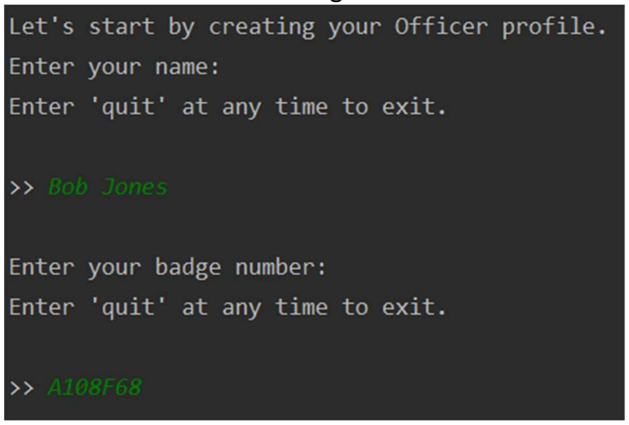
Input and Output Demonstration

Starting message display



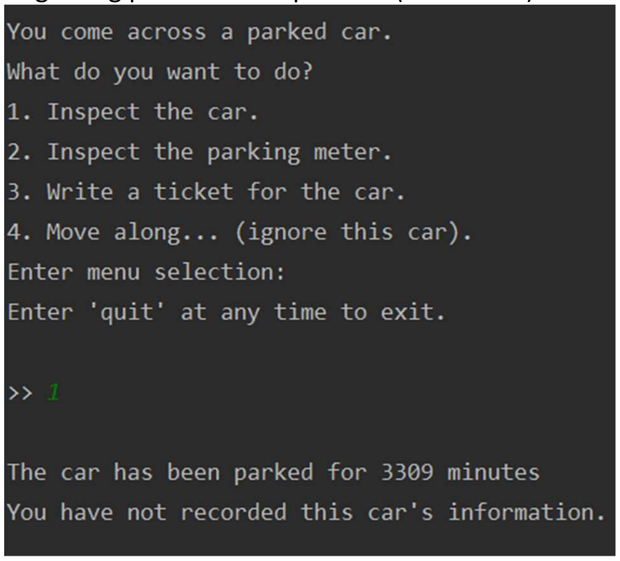
```
Welcome to the Parking Ticket Simulator.
This program will simulate a police officer issuing parking tickets.
Any tickets you issue will be saved to your ticket book.
You can view your ticket book by opening the file named:
ticket-book.txt
```

Enter officer name and badge number



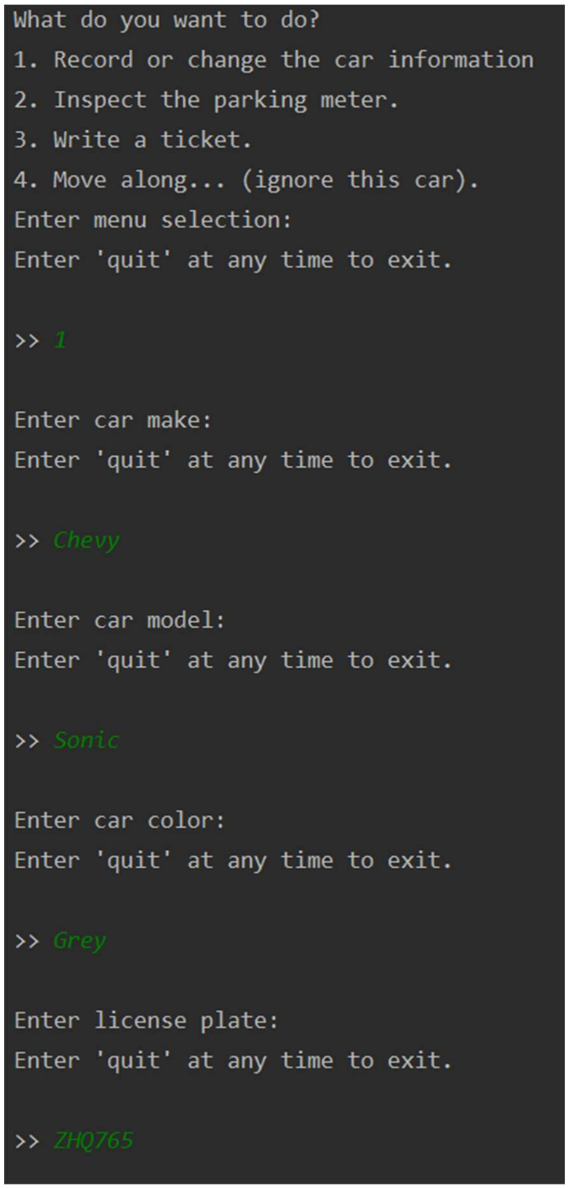
```
Let's start by creating your Officer profile.
Enter your name:
Enter 'quit' at any time to exit.

>> Bob Jones

Enter your badge number:
Enter 'quit' at any time to exit.

>> A108F68
```

Beginning patrol and inspect car (first menu)



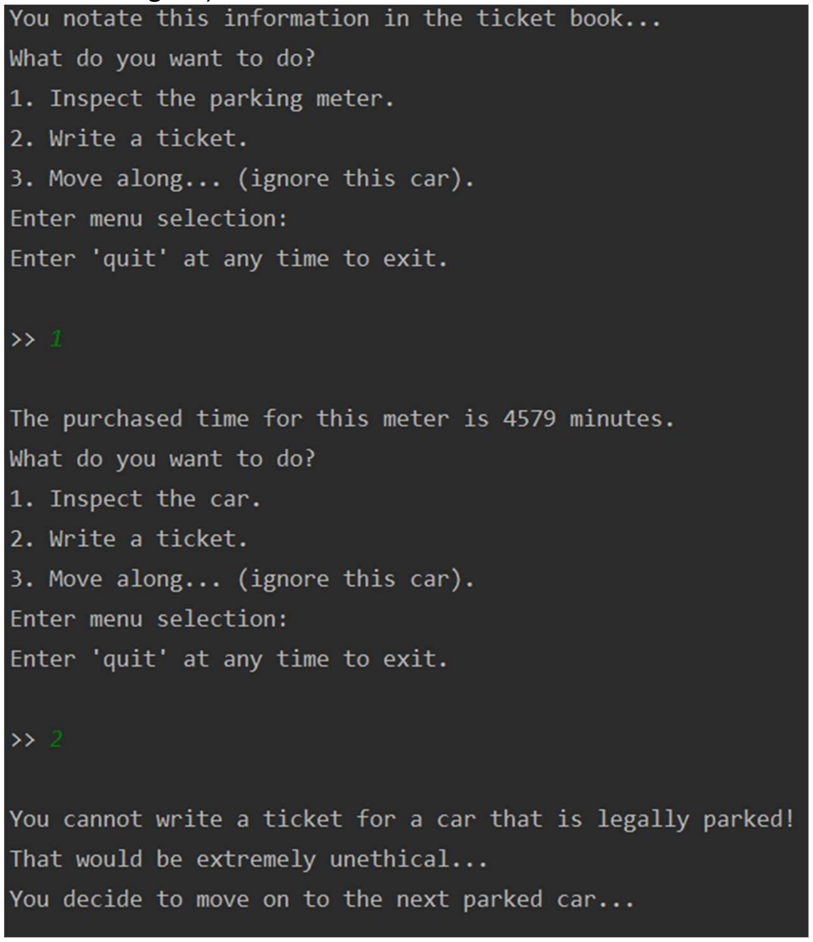
```
You come across a parked car.
What do you want to do?
1. Inspect the car.
2. Inspect the parking meter.
3. Write a ticket for the car.
4. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 1

The car has been parked for 3309 minutes
You have not recorded this car's information.
```

Record car information



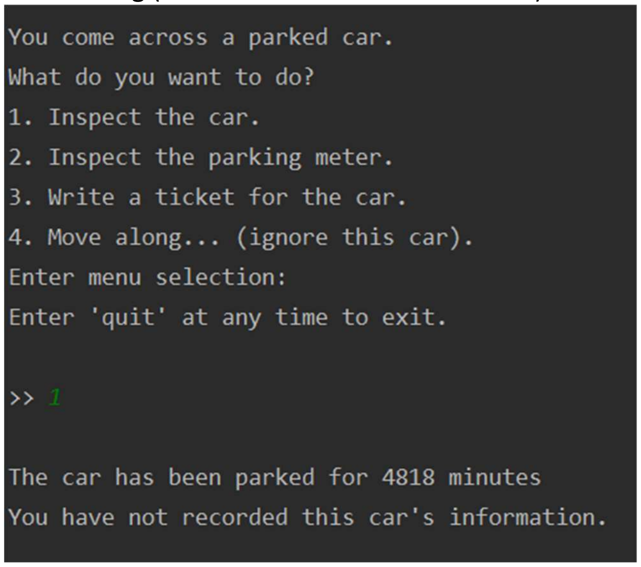
```
What do you want to do?
1. Record or change the car information
2. Inspect the parking meter.
3. Write a ticket.
4. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 1

Enter car make:
Enter 'quit' at any time to exit.

>> Chevy

Enter car model:
Enter 'quit' at any time to exit.

>> Sonic

Enter car color:
Enter 'quit' at any time to exit.

>> Grey

Enter license plate:
Enter 'quit' at any time to exit.

>> ZHQ765
```

Car information recorded, select next action (inspect parking meter, then attempt to write a ticket for non-violating car)



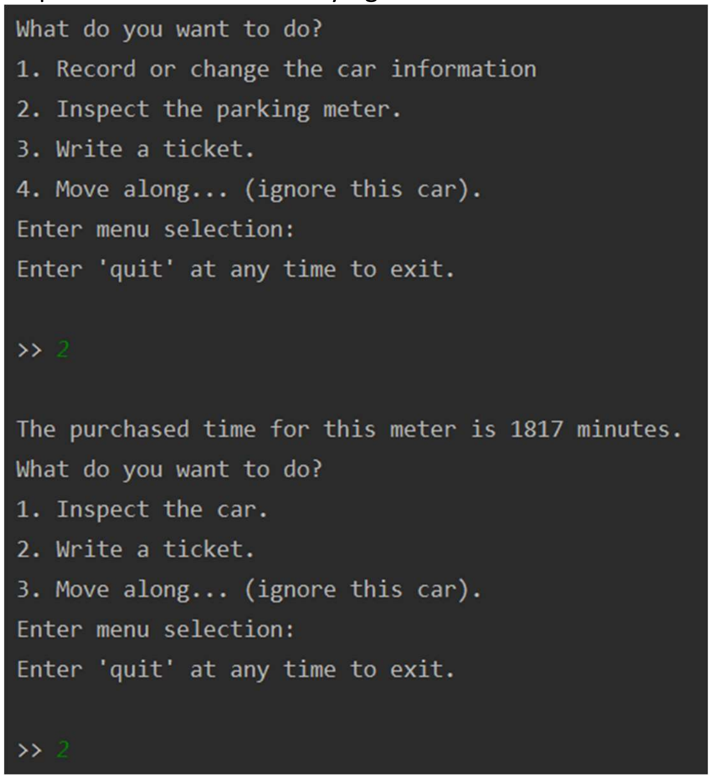
```
You notate this information in the ticket book...
What do you want to do?
1. Inspect the parking meter.
2. Write a ticket.
3. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 1

The purchased time for this meter is 4579 minutes.
What do you want to do?
1. Inspect the car.
2. Write a ticket.
3. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 2

You cannot write a ticket for a car that is legally parked!
That would be extremely unethical...
You decide to move on to the next parked car...
```

Move along (new car and meter constructed)



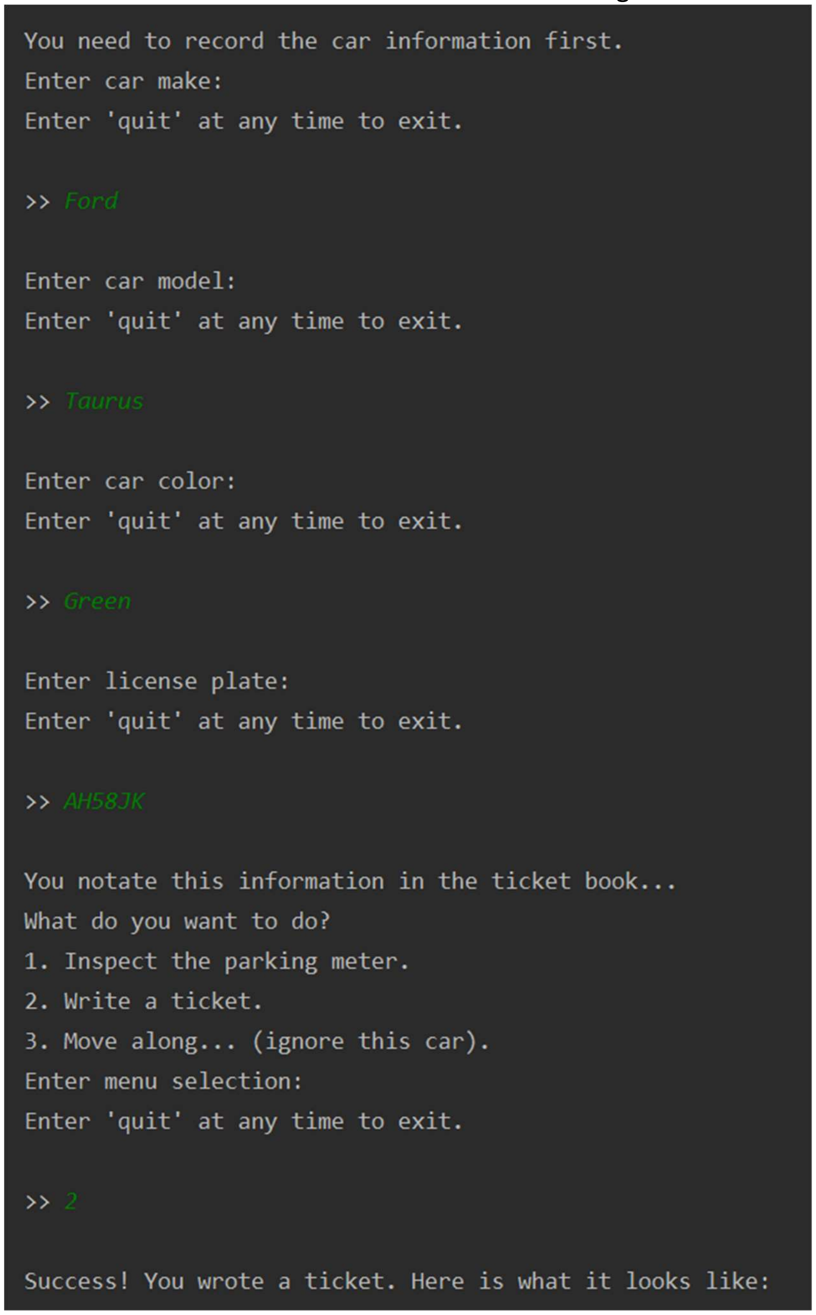
```
You come across a parked car.
What do you want to do?
1. Inspect the car.
2. Inspect the parking meter.
3. Write a ticket for the car.
4. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 1

The car has been parked for 4818 minutes
You have not recorded this car's information.
```

Inspect the meter before trying to write a ticket



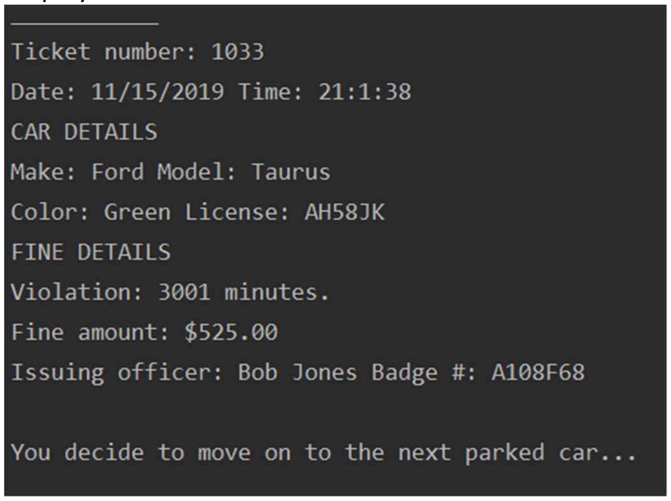
```
What do you want to do?
1. Record or change the car information
2. Inspect the parking meter.
3. Write a ticket.
4. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 2

The purchased time for this meter is 1817 minutes.
What do you want to do?
1. Inspect the car.
2. Write a ticket.
3. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 2
```

Car information needs to be recorded before writing a ticket



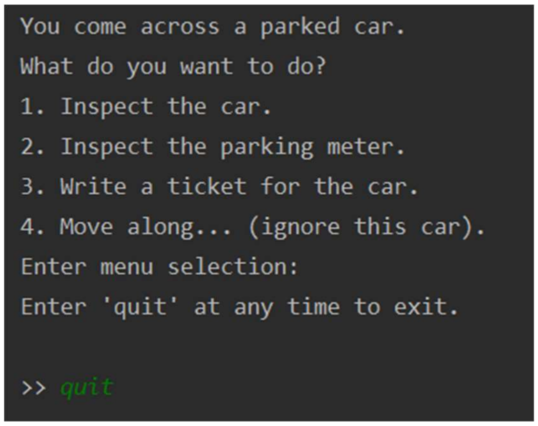
```
You need to record the car information first.
Enter car make:
Enter 'quit' at any time to exit.

>> Ford

Enter car model:
Enter 'quit' at any time to exit.

>> Taurus

Enter car color:
Enter 'quit' at any time to exit.

>> Green

Enter license plate:
Enter 'quit' at any time to exit.

>> AH58JK

You notate this information in the ticket book...
What do you want to do?
1. Inspect the parking meter.
2. Write a ticket.
3. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 2

Success! You wrote a ticket. Here is what it looks like:
```

Display the ticket



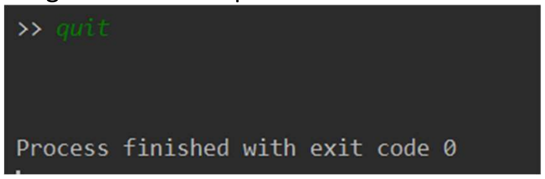
```
____________
Ticket number: 1033
Date: 11/15/2019 Time: 21:1:38
CAR DETAILS
Make: Ford Model: Taurus
Color: Green License: AH58JK
FINE DETAILS
Violation: 3001 minutes.
Fine amount: $525.00
Issuing officer: Bob Jones Badge #: A108F68

You decide to move on to the next parked car...
```

Quit



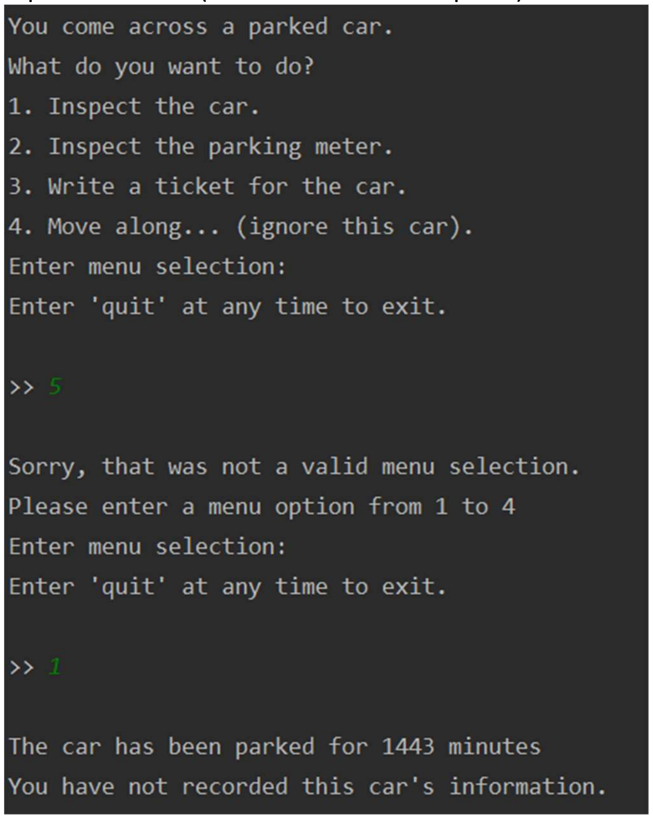
```
You come across a parked car.
What do you want to do?
1. Inspect the car.
2. Inspect the parking meter.
3. Write a ticket for the car.
4. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> quit
```

Program exits on “quit”



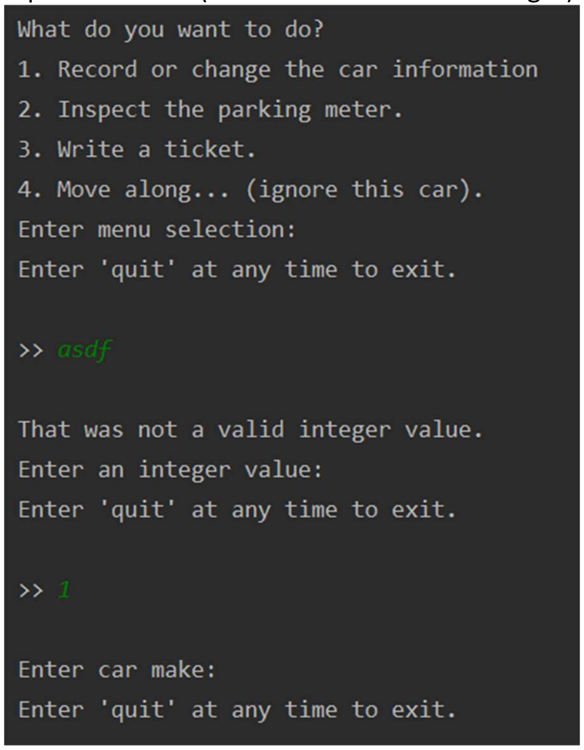
```
>> quit



Process finished with exit code 0
```

Input validation (menu value not an option)



```
You come across a parked car.
What do you want to do?
1. Inspect the car.
2. Inspect the parking meter.
3. Write a ticket for the car.
4. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> 5

Sorry, that was not a valid menu selection.
Please enter a menu option from 1 to 4
Enter menu selection:
Enter 'quit' at any time to exit.

>> 1

The car has been parked for 1443 minutes
You have not recorded this car's information.
```

Input validation (menu selection not an integer)



```
What do you want to do?
1. Record or change the car information
2. Inspect the parking meter.
3. Write a ticket.
4. Move along... (ignore this car).
Enter menu selection:
Enter 'quit' at any time to exit.

>> asdf

That was not a valid integer value.
Enter an integer value:
Enter 'quit' at any time to exit.

>> 1

Enter car make:
Enter 'quit' at any time to exit.
```