Jacob Buckelew, Ryan King, Cameron Reeves
Parking Garage Project
CMS 270
12/3/2019

Narrative

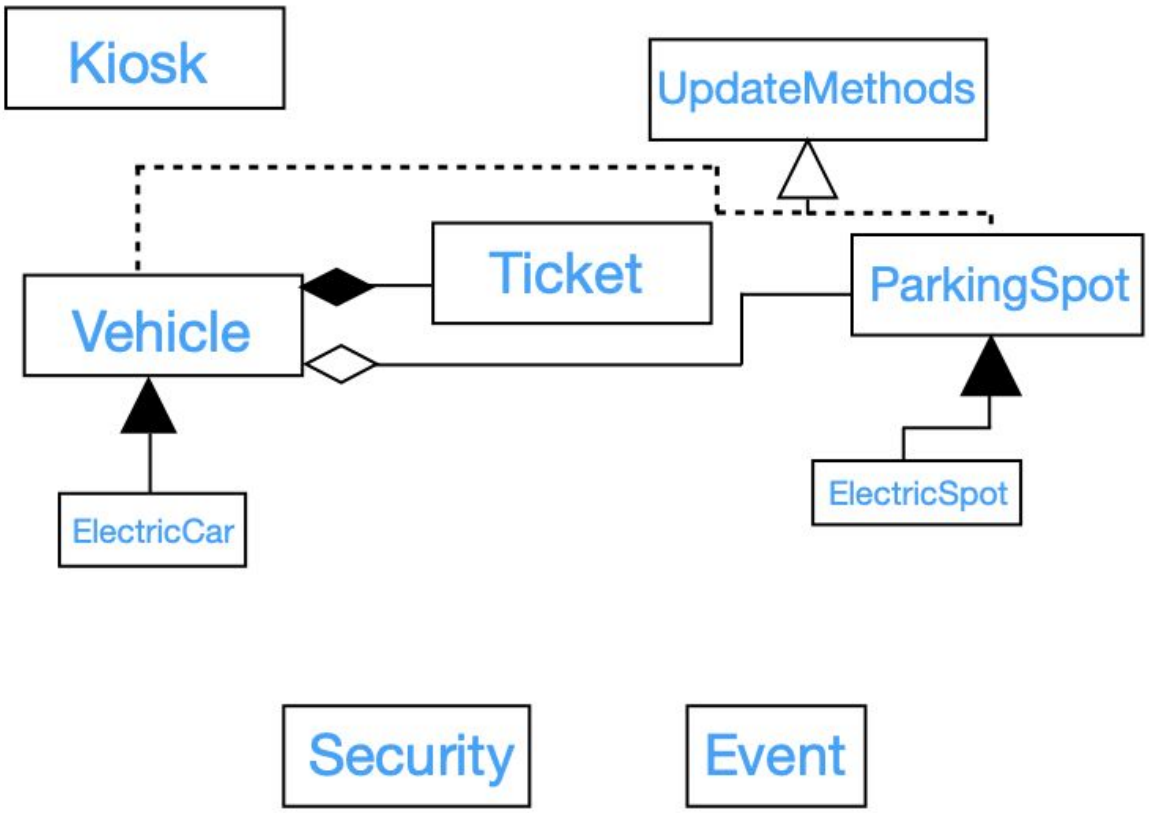
The 5 star parking garage is located in the busiest area of downtown Orlando where demand for parking is high. It was created to address this problem as well as generate revenue for the city. The garage was built with a maximum capacity of 500 vehicles.

The garage has a kiosk, which will print the ticket for a vehicle, receive payment from the user, and determine whether or not the garage is full.

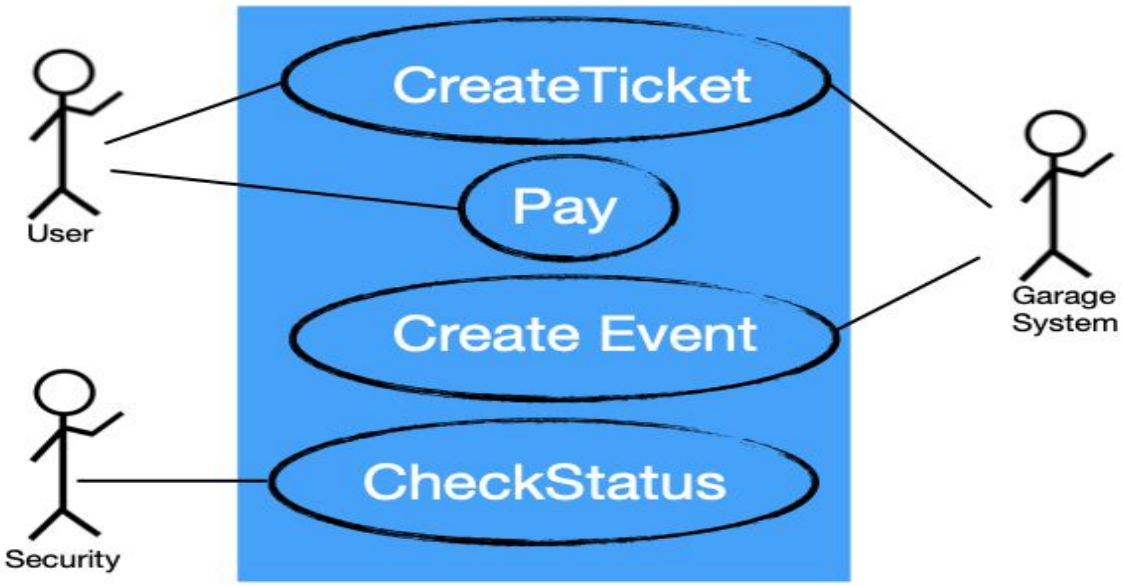
The garage has three types of spots, regular spots, electric car spots and motorcycle spots. Some of the regular spots are also handicap spots. Occasionally events are held in the local area in which case spots can be reserved. The amount of spots reserved for the event can be specified. Each vehicle must use a spot of its type, meaning electric cars can only park in electric car spots, motorcycles can only park in motorcycle spots, etc. Any type of vehicle can park in a handicap spot if it has a handicap tag.

Each vehicle is given a ticket when it enters the garage, and assigned to a spot depending on which type of vehicle it is. A ticket can be a one, six, or twelve hour ticket, and the price of staying will depend on the type of ticket. If a vehicle has a one hour ticket, it must exit the parking garage less than one hour after it receives its ticket. The six and twelve hour tickets work the same way with a deadline of six and twelve hours respectively. Ticket prices are fixed, so the price will solely depend on which type of ticket the user purchased, not how long the car actually stays. A one hour ticket is $25, a six hour ticket is $50 and a twelve hour ticket is $100. A user parking in the garage can only pay with card. If a car overstays its time limit, it will be removed by security.

Abstraction



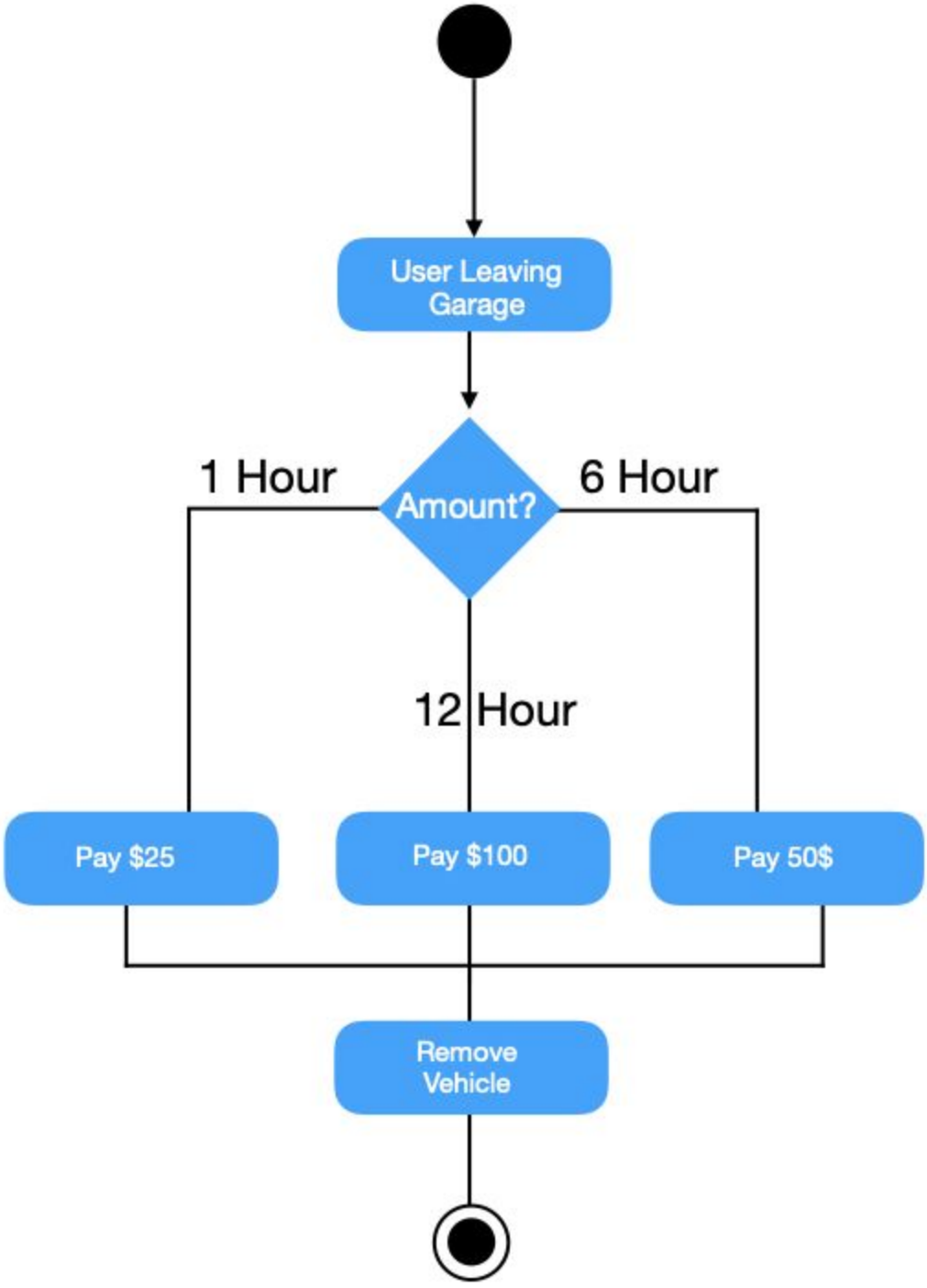
Use Case Diagram



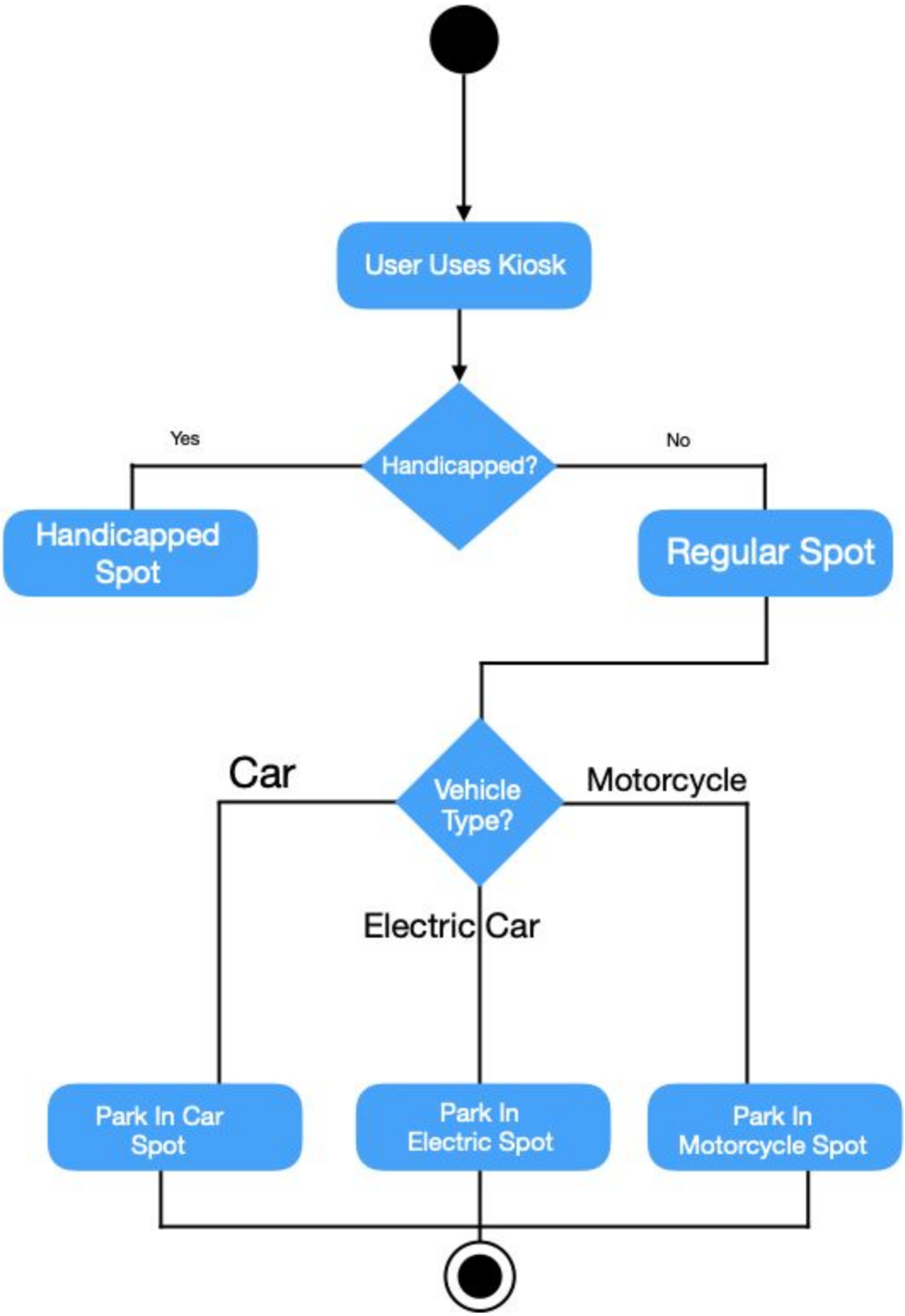
Activity Diagrams

Check Status:

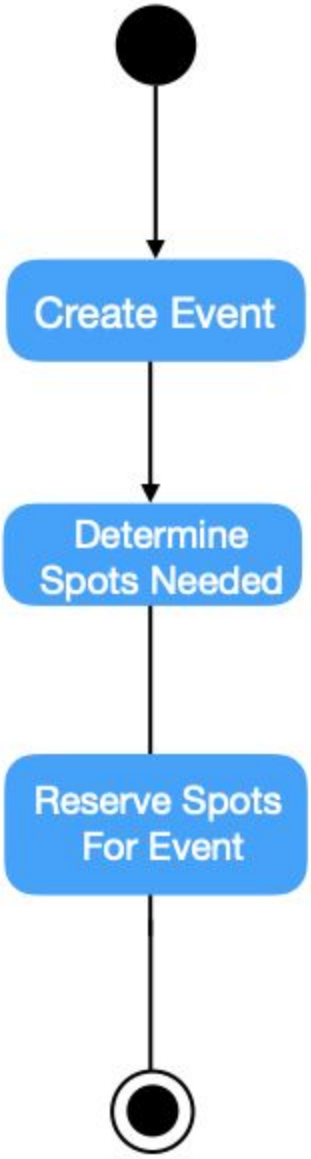
Pay:


User Leaving Garage
1 Hour
Amount?
6 Hour
12 Hour
Pay $25
Pay $100
Pay 50$
Remove Vehicle

Create Ticket:



Create Event:



```
import java.util.*;
/*
 * The Event Class contains all of the information regarding events near the garage and
 * holds a String type, and an int numSpots.
 */
public class Event {

	private String type;
	private int numSpots;

	// No Args Constructor
	public Event() {

	}

	// Args Constructor that creates an event using a String type and int numSpots
	public Event(String type, int numSpots) {
		this();
		this.type = type;
		this.numSpots = numSpots;
	}

	// Getters and Setters
	public String getType() {
		return type;
	}


	public void setType(String type) {
		this.type = type;
	}

	public int getNumSpots() {
		return numSpots;
	}

	/*
	 * setEventSpots() sets certain spots in an ArrayList<ParkingSpot>
	 * to be reserved for events.
	 */
	public void setEventSpots(ArrayList<ParkingSpot> spots) {
```

```
                int i;
                for (i = 25; i < this.getNumSpots() + 25; i++) {
                        spots.get(i).setReserved(true);
                        spots.get(i).setReservee(this.getType() + " parking only");
                }
        }


}
```

```
import java.util.*;
/*
 * The ParkingSpot class contains spots which are attributes are a boolean for occupied,
 * a boolean for reserved, a String for reservee, a boolean for handicap, an int for num,
 * and a String for type
 */
public class ParkingSpot {

	private boolean occupied;
	private boolean reserved;
	private String reservee;
	private boolean handicap;
	private int num;
	private String type;


	// No Args Constructor
	public ParkingSpot() {

	}

	//Args Constructor taking a boolean reserved and String reservee
	public ParkingSpot(boolean reserved, String reservee) {
		this();
		this.reserved = reserved;
		this.reservee = reservee;
	}
	//Args Constructor taking a boolean handicap
	public ParkingSpot(boolean handicap) {
		this();
		this.handicap = handicap;
	}
	// Args Constructor taking an int num
	public ParkingSpot(int num) {
		this.num = num;
	}

	// Getters and Setters

	public boolean isOccupied() {
		return occupied;
	}
```

```
public void setOccupied(boolean occupied) {
	this.occupied = occupied;
}

public boolean isReserved() {
	return reserved;
}

public void setReserved(boolean reserved) {
	this.reserved = reserved;
}

public boolean isHandicap() {
	return handicap;
}

public void setHandicap(boolean handicap) {
	this.handicap = handicap;
}

public void setReservee(String reservee) {
	this.reservee = reservee;
}

public String getReservee(String reservee) {
	return this.reservee;
}

public int getNum() {
	return num;
}

public void setNum(int num) {
	this.num = num;
}

public String getType() {
	return type;
}

public void setType(String type) {
	this.type = type;
}
```

```
	// isEmpty() checks if a spot is empty, will return true if occupied is false
	public boolean isEmpty() {
		if(occupied = false) {
			return true;
		}
		else {
			return false;
		}
	}
	// isFull() checks an ArrayList<ParkingSpot> to determing if the collection is fully
	// occupied
	public boolean isFull(ArrayList<ParkingSpot> spots) {

		for(int i = 0; i < spots.size(); i++) {
			if (spots.get(i).isOccupied() == false) {
				return false;
			}
		}
		return true;

	}
// updateEventStatus is from the updateMethods interfaces and is implemented
	// so that reserved status for events can be reverted back to normal.
	public void updateEventStatus() {
		this.reserved = false;
		this.reservee = null;
	}


}
```

```
/*
 * The ElectricSpot extends ParkingSpot and has a boolean charging
 * and String sign.
 */
public class ElectricSpot extends ParkingSpot {

	private boolean charging;
	private static String sign = "Electric Car Parking Only";


	// No Args Constructor
	public ElectricSpot() {
		super();
	}
	// Args Constructor that takes an int num
	public ElectricSpot(int num) {
		super(num);
	}

	// Getters and Setters

	public boolean isCharging() {
		return charging;
	}

	public String getSign() {
		return sign;
	}

	public void setCharging(boolean charging) {
		this.charging = charging;
	}
}
```

```
import java.util.*;

/*
 * The Kiosk class controls the garage and has static int for regularSpots,
 * motorcyclespots, electricspots, totalSpots, handicapSpots, and eventSpots. It
 * Also controls using a boolean barUp , boolean vacancy, and has a String name,
 * and Date date.
 */
public class Kiosk {

	private static int regularSpots = 300;
	private static int motorcycleSpots = 25;
	private static int electricSpots = 75;
	private static int totalSpots = 500;
	private static int handicapSpots = 25;
	private static int eventSpots = 75;
	private boolean barUp;
	private boolean vacancy;
	private static String name = "5-Star Parking";
	private static Date date = new Date();

	// No Args Constructor
	public Kiosk() {
		this.barUp = false;
		this.vacancy = true;
	}
	// Getters and Setters

	public static int getRegularSpots() {
		return regularSpots;
	}

	public static void setRegularSpots(int regularSpots) {
		Kiosk.regularSpots = regularSpots;
	}

	public static int getMotorcycleSpots() {
		return motorcycleSpots;
	}

	public static void setMotorcycleSpots(int motorcycleSpots) {
		Kiosk.motorcycleSpots = motorcycleSpots;
```

```
}

public static int getElectricSpots() {
	return electricSpots;
}

public static void setElectricSpots(int electricSpots) {
	Kiosk.electricSpots = electricSpots;
}

public static int getTotalSpots() {
	return totalSpots;
}

public static void setTotalSpots(int totalSpots) {
	Kiosk.totalSpots = totalSpots;
}

public boolean isBarUp() {
	return barUp;
}

public void setBarUp(boolean barUp) {
	this.barUp = barUp;
}

public boolean isVacant() {
	return vacancy;
}

public void setVacancy(boolean vacancy) {
	this.vacancy = vacancy;
}


public static String getName() {
	return name;
}

public static void setName(String name) {
	Kiosk.name = name;
}
```

```
public static Date getDate() {
	return date;
}

public static void setDate(Date date) {
	Kiosk.date = date;
}

public static int getHandicapSpots() {
	return handicapSpots;
}

public static void setHandicapSpots(int handicapSpots) {
	Kiosk.handicapSpots = handicapSpots;
}

public static void setEventSpots(int eventSpots) {
	Kiosk.eventSpots = eventSpots;
}

public static int getEventSpots() {
	return eventSpots;
}
/*
 * updateVacancy updates the vacancy boolean depending on the
 * totalSpots value
 */
public void updateVacancy() {
	if(totalSpots == 0) {
		setVacancy(false);
	}
}

/*
 * printGreeting() prints a greeting from the Kiosk
 */
public void printGreeting() {
	System.out.println("Welcome to 5-Star Parking");
}

/*
 * payForParking returns a double, total, and takes
 * a Vehicle as an arg.
```

```
 */
public double payForParking(Vehicle vehicle) {

	double total = 0;

	Ticket ticket = vehicle.getTicket();
	if(ticket.getType().equals("1hr")) {
		total = 25.00;
	}
	else if (ticket.getType().equals("6hr")) {
		total = 50.00;
	}
	else if (ticket.getType().equals("12hr")) {
		total = 100.00;
	}

	if(vehicle.getType().equals("ElectricCar")) {
		ElectricCar car = (ElectricCar) vehicle;
		total = total + car.calculateFee();
	}

	return total;

}

/*
 * printTicket takes a Vehicle as an arg and prints
 * ticket information
 */
public void printTicket(Vehicle vehicle) {
	if(vacancy == false) {
		System.out.println("Sorry, 5-Star Parking is currently full");
	}
	else {

		if(vehicle.isAttendingEvent()) {
			if ( 0 < eventSpots && eventSpots <= 75) {
				eventSpots--;
				totalSpots--;
				System.out.println(vehicle.getTicket().toString());
			}
			else {
```

```
                                System.out.println("Sorry, there are no more event
spots available");
                        }
                }
                else if (vehicle.isHandicap()) {
                        if ( 0 < handicapSpots && handicapSpots <= 25) {
                                handicapSpots--;
                                totalSpots--;
                                System.out.println(vehicle.getTicket().toString());

                        }
                        else {
                                System.out.println("Sorry, there are no more handicap
spots available");
                        }
                }
                else if(vehicle.getType().equals("Motorcycle")) {
                        if ( 0 < motorcycleSpots && motorcycleSpots <= 25) {
                                motorcycleSpots--;
                                totalSpots--;
                                System.out.println(vehicle.getTicket().toString());
                        }
                        else {
                                System.out.println("Sorry, there are no more
motorcycle spots available.");
                        }
                }
                else if(vehicle.getType().equals("Car")) {
                        if ( 0 < regularSpots && regularSpots <= 300) {
                                regularSpots--;
                                totalSpots--;
                                System.out.println(vehicle.getTicket().toString());
                        }
                        else {
                                System.out.println("Sorry, there are no more car spots
available.");
                        }
                }
                else if (vehicle.getType().equals("ElectricCar")) {
                        if ( 0 < electricSpots && electricSpots <= 75) {
                                electricSpots--;
                                totalSpots--;
                                System.out.println(vehicle.getTicket().toString());
```

```
                        }
                        else {
                                System.out.println("Sorry there are no more regular
spots available.");
                        }
                }
        }
        System.out.println("-----------------------------");
        updateVacancy();
    }

    /*
     * removeVehicle() removes Vehicles using a Vehicle and an ArrayList<Vehicle> as
     * args
     */
    public void removeVehicle(Vehicle vehicle, ArrayList<Vehicle> vehicles) {

        if(vehicle.isAttendingEvent() == true) {
            eventSpots++;
        }
        else if(vehicle.isHandicap()) {
            handicapSpots++;
        }
        else if(vehicle.getType().equals("Car")) {
            regularSpots++;
        }
        else if (vehicle.getType().equals("ElectricCar")) {
            electricSpots++;
        }
        else if (vehicle.getType().equals("Motorcycle")) {
            motorcycleSpots++;
        }
        totalSpots++;
        vehicle.getSpot().setOccupied(false);
        updateVacancy();

        vehicles.remove(vehicle);

        System.out.println("Vehicle at spot " + vehicle.getSpot().getNum() +
                                            " has exited the garage.");

    }
```

```
	/*
	 * toString() prints a Kiosk object as a String
	 */
	public String toString() {
		if(vacancy == false) {
			return "We are sorry to inform you that all parking spots"
					+ " are full at this moment." + "\n" +
"-----------------------------";
		}
		return  "Regular spots available: " + getRegularSpots() +
				"\n" + "Electric spots available: " + getElectricSpots() + "\n" +
"Motorcycle spots available: "
				+ getMotorcycleSpots() + "\n" + "Handicap spots available: "
+ getHandicapSpots() + "\n" +
				"Event spots available: " + getEventSpots() + "\n" +
"-----------------------------";
	}


}
```

```
/*
 * The Vehicle class has a String type, ParkingSpot spot, Ticket ticket, int hoursSpent,
boolean handicap,
 * and boolean attendingEvent.
 */
public class Vehicle {

	private String type;
	private ParkingSpot spot;
	private Ticket ticket;
	private int hoursSpent;
	private boolean handicap;
	private boolean attendingEvent;

	// No Args Constructor
	public Vehicle() {

	}

	// Args Constructor with String type, int hoursSpent, boolean handicap, boolean
event
	public Vehicle(String type, int hoursSpent, boolean handicap, boolean event) {
		this();
		this.type = type;
		this.hoursSpent = hoursSpent;
		this.handicap = handicap;
		this.attendingEvent = event;
	}

	// Getters and Setters
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}
	public ParkingSpot getSpot() {
		return spot;
	}
	public void setSpot(ParkingSpot spot) {
		this.spot = spot;
	}
	public Ticket getTicket() {
```

```
            return ticket;
        }
        public void setTicket(Ticket ticket) {
            this.ticket = ticket;
        }

        public boolean isHandicap() {
            return handicap;
        }

        public void setHandicap(boolean b) {
            this.handicap = b;
        }

        public boolean isAttendingEvent() {
            return attendingEvent;
        }

        public int getHoursSpent() {
            return hoursSpent;
        }

        // to String() prints out a vehicle using its type
        public String toString() {
            return "Type: " + getType();
        }


// updateEventStatus comes from the updateMethods interface and
        // reverts a vehicle to not attending an event in case there is an error
        // and the customer does not want to attend Event
        public void updateEventStatus() {
            attendingEvent = false;
        }

}
```

```
/*
 * ElectricCar extends Vehicle and has an int for numHoursCharging
 */
public class ElectricCar extends Vehicle {

	private int numHoursCharging;

	// No Args Constructor
	public ElectricCar() {

	}

	// Args Constructor taking a String type, int hoursSpent, boolean handicap,
boolean event, and int hours
	public ElectricCar(String type, int hoursSpent, boolean handicap, boolean event,
int hours) {
		super(type, hoursSpent, handicap, event);
		numHoursCharging = hours;
	}
	// Getters and Setters
	public int getNumHoursCharging() {
		return numHoursCharging;
	}
	// calculateFee() determines fee for charging and returns a total as a double
	public double calculateFee() {
		return numHoursCharging * 4.00;
	}

}


import java.util.*;
import java.time.*;
import java.time.temporal.*;
import java.time.format.*;
```

```
/*
 * The Ticket class has a String name, LocalDate date, LocalTime entryTime, String type,
LocalTime exitTime,
 * long cardNum, and int spotNum.
 */

public class Ticket {

	private String name;
	private LocalDate date;
	private LocalTime entryTime;
	private String type;
	private LocalTime exitTime;
	private long cardNum;
	private int spotNum;


	// Args constructor taking String n, String type, and long cardNum
	public Ticket(String n, String type, long cardNum) {
		name = n;
		this.cardNum = cardNum;
		date = LocalDate.now();
		entryTime = LocalTime.now().truncatedTo(ChronoUnit.MINUTES);
		this.type = type;
		if(type.equals("12hr")) {
			exitTime = entryTime.plus(12, ChronoUnit.HOURS);
			exitTime = exitTime.truncatedTo(ChronoUnit.MINUTES);
		}
		else if(type.equals("6hr")) {
			exitTime = entryTime.plus(6, ChronoUnit.HOURS);
			exitTime = exitTime.truncatedTo(ChronoUnit.MINUTES);
		}
		else {
			exitTime = entryTime.plus(1, ChronoUnit.HOURS);
			exitTime = exitTime.truncatedTo(ChronoUnit.MINUTES);
		}
	}


	// Getters and Setters
	public String getName() {
		return name;
	}
```

```
public void setName(String name) {
	this.name = name;
}


public LocalDate getDate() {
	return date;
}


public void setDate(LocalDate date) {
	this.date = date;
}


public LocalTime getEntryTime() {
	return entryTime;
}


public void setEntryTime(LocalTime entryTime) {
	this.entryTime = entryTime;
}


public String getType() {
	return type;
}


public void setType(String type) {
	this.type = type;
}
```

```
	public LocalTime getExitTime() {
		return exitTime;
	}



	public void setExitTime(LocalTime exitTime) {
		this.exitTime = exitTime;
	}

	public int getSpotNum() {
		return spotNum;
	}

	public void setSpotNum(int num) {
		spotNum = num;
	}

	// toString() prints a Ticket using name, date, type, entrytime, spotnum, and
exittime
	public String toString() {
		return "Name: " + getName() + "\n" + "Date: " +
				getDate() + "\n" + "Ticket: " + getType() +
				"\n" + "Entry Time: " + getEntryTime() + "\n" +
				"Spot Number: " + getSpotNum() + "\n" +
				"If your vehicle is here past " + getExitTime() +
				", Security will have to remove your vehicle from "
				+ "the garage.";
	}
}
```

```
import java.util.ArrayList;


/*
 * The Security class contains an ArrayList<ParkingSpot> spots and checks status of parked vehicles
 */
public class Security {

	private ArrayList<ParkingSpot> spots;

	// No Args Constructor
	public Security() {

	}
	// Args constructor taking an ArrayList<ParkingSpot> spots, and Kiosk kiosk
	public Security(ArrayList<ParkingSpot> spots, Kiosk kiosk) {
		this();
		this.spots = spots;
	}

	/*
	 * checkStatuses() checks to see if a vehicle has overstayed. It takes
	 * an ArrayList<Vehicle> and a kiosk as arguments
	 */
	public void checkStatuses(ArrayList<Vehicle> vehicles, Kiosk kiosk) {

		int i;
		for (i = 0; i < vehicles.size() ; i++) {
			if(vehicles.get(i).getTicket().getType().equals("1hr")) {
				if(vehicles.get(i).getHoursSpent() > 1) {
					towVehicle(vehicles.get(i), vehicles, kiosk);
				}
			}
			if(vehicles.get(i).getTicket().getType().equals("6hr")) {
				if(vehicles.get(i).getHoursSpent() > 6) {
					towVehicle(vehicles.get(i), vehicles, kiosk);
				}
			}
			if(vehicles.get(i).getTicket().getType().contentEquals("12hr")) {
				if(vehicles.get(i).getHoursSpent() > 12) {
					towVehicle(vehicles.get(i), vehicles, kiosk);
				}
```

```
                        }
                }


        }
        /*
         * towVehicle will tow a vehicle if it has stayed too long. It takes a Vehicle,
         * an ArrayList<Vehicle>, and a Kiosk as args)
         */
        public static void towVehicle(Vehicle vehicle, ArrayList<Vehicle> vehicles, Kiosk
kiosk) {

                if(vehicle.isAttendingEvent() == true) {
                        Kiosk.setEventSpots(Kiosk.getEventSpots() + 1);
                }
                else if(vehicle.isHandicap()) {
                        Kiosk.setHandicapSpots(Kiosk.getHandicapSpots() + 1);
                }
                else if(vehicle.getType().equals("Car")) {
                        Kiosk.setRegularSpots(Kiosk.getRegularSpots() + 1);
                }
                else if (vehicle.getType().equals("ElectricCar")) {
                        Kiosk.setElectricSpots(Kiosk.getElectricSpots() + 1);
                }
                else if (vehicle.getType().equals("Motorcycle")) {
                        Kiosk.setMotorcycleSpots(Kiosk.getMotorcycleSpots() + 1);
                }
                Kiosk.setTotalSpots(Kiosk.getTotalSpots() + 1);
                vehicle.getSpot().setOccupied(false);
                kiosk.updateVacancy();

                vehicles.remove(vehicle);

                System.out.println("Vehicle at spot " + vehicle.getSpot().getNum() +
                                                        " has been towed away from the
garage.");

                System.out.println("-----------------------------");

        }
}
```

```
/*
 * The updateMethods interface contains a shareable method updateEventStatus to
update Vehicles and ParkingSpots.
 */
public interface updateMethods {

	public void updateEventStatus();

}
```

```
import java.util.*;
import java.io.*;

/*
 * The Driver class runs the main program for the Parking Garage project.
 * Jacob Buckelew, Ryan King, Cameron Reeves
 * Parking Garage Project
 * CMS 270
 *
 */
public class Driver {

	/*
	 * The generateParkingSpots() method takes no args and returns an
ArrayList<ParkingSpot>
	 * by generating a new parking lot including normal spots and electric spots
	 */
	public static ArrayList<ParkingSpot> generateParkingSpots() {

		ArrayList<ParkingSpot> parkingSpots =  new ArrayList<ParkingSpot>();
		int i;
		for ( i = 0; i < 425; i++) {
			parkingSpots.add(new ParkingSpot(i));
			if(i < 25) {
				parkingSpots.get(i).setHandicap(true);
			}
			else if (i < 100) {
				parkingSpots.get(i).setReserved(true);
			}
			else if( i < 400) {
				parkingSpots.get(i).setType("Car");
			}
			else if (i < 425) {
				parkingSpots.get(i).setType("Motorcycle");
			}
		}
		ArrayList<ElectricSpot> electricSpots = new ArrayList<ElectricSpot>();
		for ( i = 0; i < 75; i++) {
			electricSpots.add(new ElectricSpot());
			electricSpots.get(i).setType("Electric Car");
		}
		parkingSpots.addAll(electricSpots);
		return parkingSpots;
```

```
	}

	/*
	 * registerVehicles() takes an ArrayList<ParkingSpot> and a Kiosk as args and returns an ArrayList<Vehicle>
	 * to be used as a collection of all vehicles in the system
	 */
	public static ArrayList<Vehicle> registerVehicles(ArrayList<ParkingSpot> lot, Kiosk kiosk) {

		ArrayList<Vehicle> vehicles = new ArrayList<>();

		try {
		File vehicleText = new File("Vehicles.txt");
		Scanner scan = new Scanner(vehicleText);

		while(scan.hasNextLine() && kiosk.isVacant() != false) {
			String name = scan.next() + " " + scan.next();
			String type = scan.next();
			if(type.equals("Car")) {
				Vehicle vehicle = new Vehicle(type, scan.nextInt(), scan.nextBoolean(), scan.nextBoolean());
				Ticket ticket = new Ticket(name, scan.next(), scan.nextLong());
				vehicle.setTicket(ticket);
				vehicles.add(vehicle);
			}
			if(type.equals("ElectricCar")) {
				ElectricCar electric = new ElectricCar(type, scan.nextInt(), scan.nextBoolean(),scan.nextBoolean(), scan.nextInt());
				Ticket ticket = new Ticket(name, scan.next(), scan.nextLong());
				electric.setTicket(ticket);
				vehicles.add(electric);
			}
			if (type.equals("Motorcycle")) {
				Vehicle motorcycle = new Vehicle(type, scan.nextInt(), scan.nextBoolean(), scan.nextBoolean());
				Ticket ticket = new Ticket(name, scan.next(), scan.nextLong());
				motorcycle.setTicket(ticket);
				vehicles.add(motorcycle);
			}
```

```
			}
			scan.close();
			}
			catch (Exception FileNotFoundException) {
				System.out.println("file not found.");

			}
			return vehicles;
		}

		/*
		 * fillpots() takes an ArrayList<Vehicle>, ArrayList<ParkingSpot>, and a Kiosk as args and then
		 * assigns each spot to a vehicle since each vehicle "has" a spot.
		 */
		public static void fillSpots(ArrayList<Vehicle> vehicles, ArrayList<ParkingSpot> spots, Kiosk kiosk) {

			int count1 = 0;
			int count2 = 25;
			int count3 = 100;
			int count4 = 400;
			int count5 = 425;

			int i;

			for (i = 0; i < vehicles.size(); i++) {
				kiosk.setBarUp(false);
				String type = vehicles.get(i).getType();
				if( vehicles.get(i).isAttendingEvent()) {
					vehicles.get(i).setSpot(spots.get(count2));
					count2++;
				}
				else if (type.equals("Handicap")) {
					vehicles.get(i).setSpot(spots.get(count1));
					count1++;
				}
				else if (type.equals("Car")) {
					vehicles.get(i).setSpot(spots.get(count3));
					count3++;
				}
				else if (type.equals("Motorcycle")) {
					vehicles.get(i).setSpot(spots.get(count4));
```

```
                        count4++;
                    }
                    else if (type.equals("ElectricCar")) {
                        vehicles.get(i).setSpot(spots.get(count5));
                    }


vehicles.get(i).getTicket().setSpotNum(vehicles.get(i).getSpot().getNum());
                    vehicles.get(i).getSpot().setOccupied(true);
                    kiosk.printTicket(vehicles.get(i));
                    kiosk.setBarUp(true);

                }


        }

        /*
         * pay takes a Vehicle, ArrayList<Vehicle>, and Kiosk to allow a customer to pay.
This
         * also includes removing a vehicle from the garage.
         */

        public static void pay(Vehicle vehicle, ArrayList<Vehicle> vehicles, Kiosk kiosk) {

                double total = kiosk.payForParking(vehicle);
                System.out.println("Your total is " + total + "\n" + "Your card will be
charged"
                                                    + " the appropriate amount.");
                kiosk.setBarUp(true);
                System.out.println("You may now exit. Have a nice day.");
                kiosk.setBarUp(false);
                kiosk.removeVehicle(vehicle, vehicles);

                System.out.println("------------------------------");

        }

        /*
         * The main() function is where the ParkingSpots will be instantiated and
         * different test cases will be ran to show how the garage system functions.
         */
        public static void main(String[] args) {
```

```
        Kiosk kiosk = new Kiosk();
        kiosk.printGreeting();
        System.out.println(kiosk.toString());

        ArrayList<ParkingSpot> defaultLot = generateParkingSpots();
        Event UFC = new Event("Sporting Event", 75);
        UFC.setEventSpots(defaultLot);

        ArrayList<Vehicle> vehicles = registerVehicles(defaultLot, kiosk);
        fillSpots(vehicles, defaultLot, kiosk);

        Security security = new Security(defaultLot, kiosk);
        security.checkStatuses(vehicles, kiosk);
        System.out.println(kiosk.toString());

        pay(vehicles.get(1), vehicles, kiosk);
        System.out.println(kiosk.toString());

        pay(vehicles.get(0), vehicles, kiosk);
        pay(vehicles.get(0), vehicles, kiosk);
        System.out.println(kiosk.toString());



    }
}
```

Reflection

Challenges we faced:

- Conflicting schedules made getting together as a group difficult.
- Balancing our time between this project and other work we had to do
- Deciding how to divide the work between ourselves.
- Determining the design strategy for our garage.
- Determining which ideas to use and which to throw out.
- Organizing our code in a correct object-oriented style.
- Various bugs in our program.

Things we learned:

- The importance of communication in team programming projects.
- To leave more time for debugging and solving problems in our code.
- Plan ahead of time for periods of time where we will be unable to meet (such as Thanksgiving break).
- Plan ahead of time how the classes will interact with everyone.
- To check with the whole team more frequently to make sure we're all on the same page.
- We developed a better understanding of object-oriented programming.