Luis Gutierrez

CS 152-01

10/04/18

# P2

## Car.java

```java
/**
 * Class representing a car and it's properties
 *
 * @author (Luis Gutierrez)
 * @version (08/30/2018)
 */
public class Car
{
    private String vin, make, model;
    private double cost;
    private int year;

    /**
     * Constructor for objects of class Car
     */
    public Car(
        String vin,
        String make,
        String model,
        double cost,
        int year)
    {
        this.vin = vin;
        this.make = make;
        this.model = model;
        this.cost = cost;
        this.year = year;
    }

    /**
     * Constructor for objects of class car
     *
     * Assumes that array items are in the correct order
     * @param tokens - A String array containing strings that correspond to car properties
     */
    public Car(String[] tokens){
        this.vin = tokens[0];
        this.make = tokens[1];
        this.model = tokens[2];
        this.cost = Double.parseDouble(tokens[3]);
        this.year = Integer.parseInt(tokens[4]);
    }

    /**
     * Returns the car's cost
     *
     * @return this.cost
     */
    public double getCost()
    {
        return this.cost;
    }

    /**
     * Returns this car's make
     *
     * @return this.make
     */
    public String getMake(){
        return this.make;
    }

    /**
     * Returns true if cost of car is greater than $30,000
     */
    public boolean isExpensive(){
        return this.cost > (double) 30000;
    }

    /**
     * Returns true if car's model year is before 1968
     */
    public boolean isAntique(){
        return this.year < 1968;
    }

    /**
     * Returns a string representation of this car
     *
     * @return String
     */
    public String toString(){
        String template = "%s\t%s\t%s\t%s\t%s";

        return String.format(template,
            this.vin,
            this.make,
            this.model,
            this.cost,
            this.year
        );
    }
}
```

## CarCollectio.java

```
/**
 * A class with methods meant to recursively iterate over
 * arrays of Car objects
 *
 * @author (Luis Gutierrez)
 * @version (10/03/18)
 */
public class CarCollection
{
    /**
     * Returns a string representation of all cars in an array
     *
     * @param  list  A list of car obects
     * @param len Length of list
     * @return    String
     */
    public static String toString(Car[] list, int len)
    {
        if (len > 0)
        {
            return list[len - 1].toString() + "\n" + toString(list, len - 1);
        }
        else
        {
            return "";
        }
    }

    /**
     * Returns the number of antique cars in thre list
     *
     * @param  list  A list of car obects
     * @param len Length of list
     * @return    int
     */
    public static int countAntique(Car[] list, int len)
    {
```

```java
        if (len > 0)
        {
            if (list[len - 1].isAntique())
            {
                return 1 + countAntique(list, len - 1);
            }
            else
            {
                return countAntique(list, len - 1);
            }
        }else
        {
            return 0;
        }
    }

    /**
     * Prints all cars from the list that are expensive
     *
     * @param  list  A list of car obects
     * @param len Length of list
     */
    public static void printExpensiveCars(Car[] list, int len)
    {
        if(len > 0)
        {
            Car car = list[len - 1];
            printExpensiveCars(list, len - 1);
            if (car.isExpensive())
            {
                System.out.println(car.toString());
            }
        }
    }

    /**
     * Prints all cars from the list of a certain make
     *
     * @param  list  A list of car obects
     * @param len Length of list
     */
    public static void printCarsWithMake(Car[] list, int len, String make)
    {
        if(len > 0)
        {
            Car car = list[len - 1];
            printCarsWithMake(list, len - 1, make);
            if (car.getMake().compareToIgnoreCase(make) == 0)
            {
                System.out.println(car.toString());
            }
        }
    }

    public static Car cheapestCar(Car[] list, int len)
    {
        if (len > 0)
        {
            if (list[len - 1].getCost() < cheapestCar(list, len - 1).getCost()){
                return list[len - 1];
            }
            else
            {
                return cheapestCar(list,len - 1);
            }
        }
        else
        {
            return list[len];
        }
    }
}
```

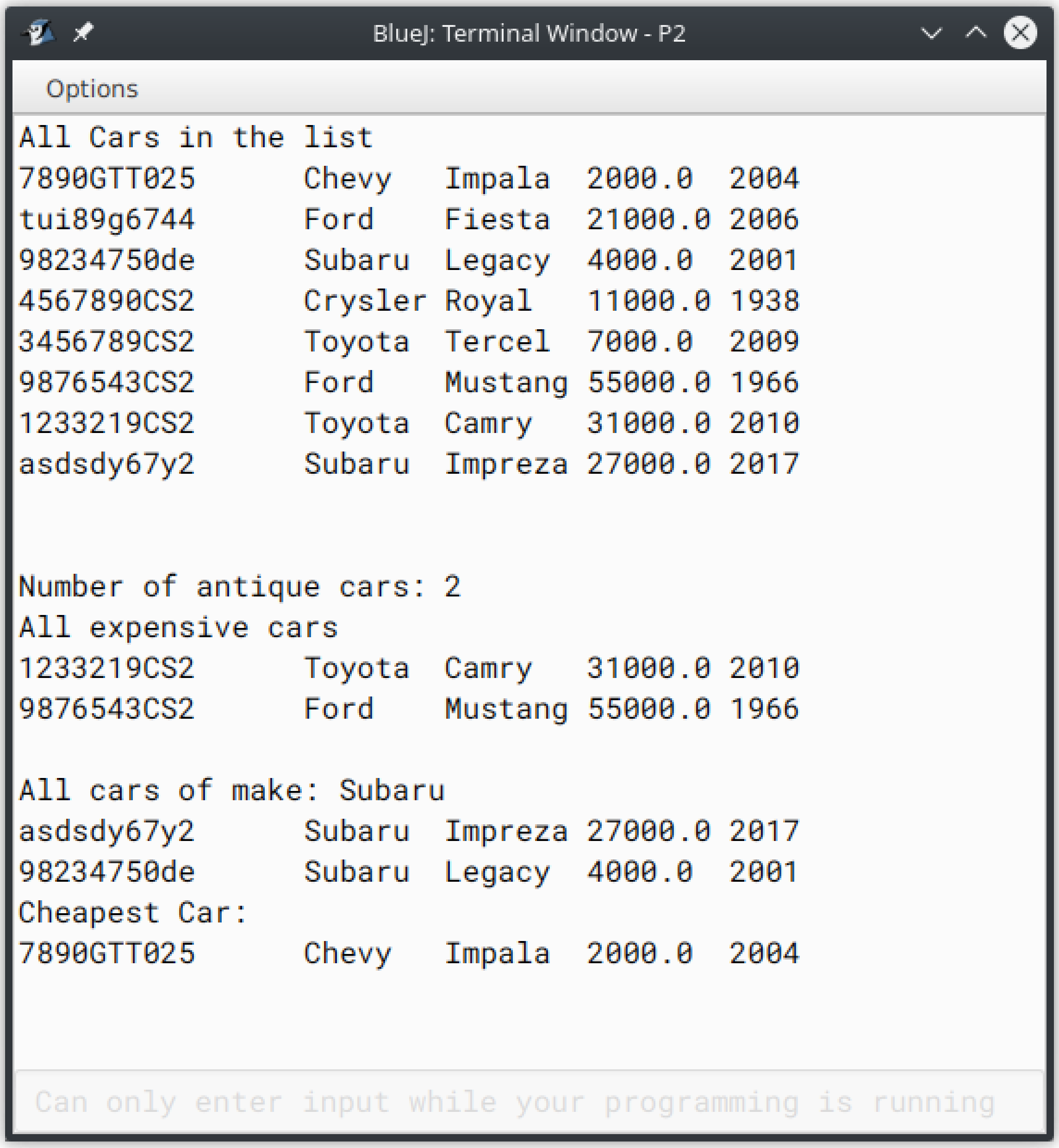
## TestCarList.java

```
import java.util.*;
import java.io.*;
/**
 * Write a description of class TestCarList here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class TestCarList
{
    public static void main(String[] args) throws IOException
    {
        File file = new File("inData.txt");
        Car[] myCars = new Car[100];

        Scanner sc = new Scanner(file);
        int carCount = 0;
        while(sc.hasNextLine())
        {
            String[] tokens = sc.nextLine().toString().split(" ");
            myCars[carCount] = new Car(tokens);
            carCount++;
        }
        System.out.println("All Cars in the list");
        System.out.println(CarCollection.toString(myCars, carCount) + "\n");

        System.out.println("Number of antique cars:\t" +
            CarCollection.countAntique(myCars, carCount));

        System.out.println("All expensive cars");
        CarCollection.printExpensiveCars(myCars, carCount);
        System.out.println();

        String make = "Subaru";
        System.out.println("All cars of make: " + make);
        CarCollection.printCarsWithMake(myCars, carCount, make);

        System.out.println("Cheapest Car:");
        System.out.println(CarCollection.cheapestCar(myCars, carCount));
    }
}
```

## input file

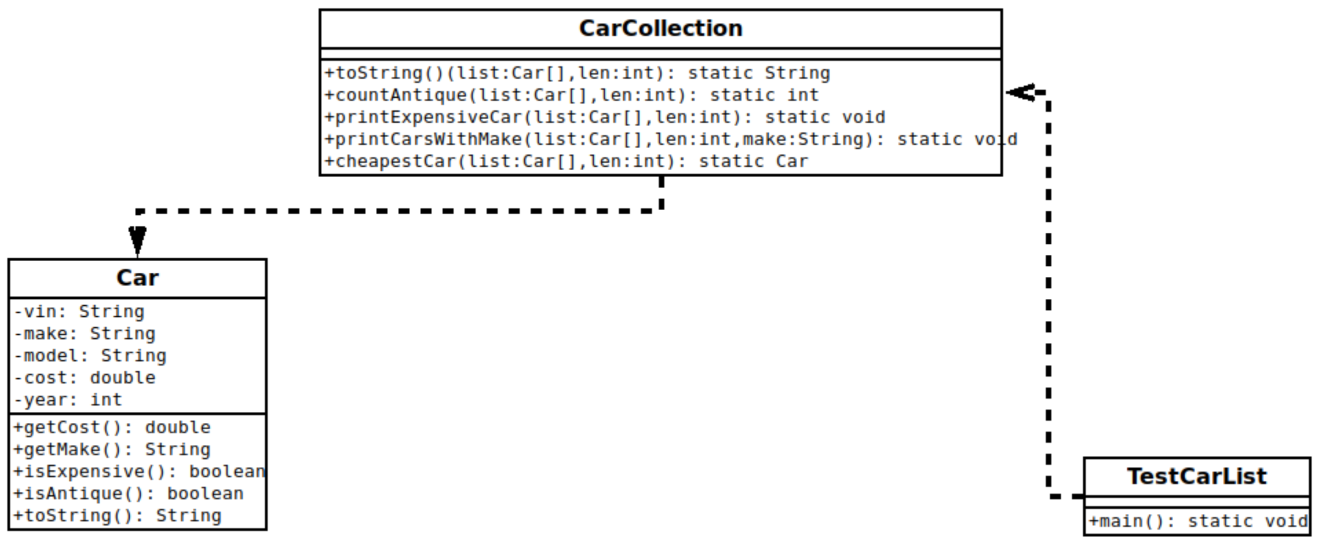
```
asdsdy67y2 Subaru Impreza 27000 2017
1233219CS2 Toyota Camry 31000 2010
9876543CS2 Ford Mustang 55000 1966
3456789CS2 Toyota Tercel 7000 2009
4567890CS2 Crysler Royal 11000 1938
98234750de Subaru Legacy 4000 2001
tui89g6744 Ford Fiesta 21000 2006
7890GTT025 Chevy Impala 2000 2004
```

## Console screenshot



```
All Cars in the list
7890GTT025      Chevy   Impala  2000.0  2004
tui89g6744      Ford    Fiesta  21000.0 2006
98234750de      Subaru  Legacy  4000.0  2001
4567890CS2      Crysler Royal   11000.0 1938
3456789CS2      Toyota  Tercel  7000.0  2009
9876543CS2      Ford    Mustang 55000.0 1966
1233219CS2      Toyota  Camry   31000.0 2010
asdsdy67y2      Subaru  Impreza 27000.0 2017


Number of antique cars: 2
All expensive cars
1233219CS2      Toyota  Camry   31000.0 2010
9876543CS2      Ford    Mustang 55000.0 1966

All cars of make: Subaru
asdsdy67y2      Subaru  Impreza 27000.0 2017
98234750de      Subaru  Legacy  4000.0  2001
Cheapest Car:
7890GTT025      Chevy   Impala  2000.0  2004
```

## UML Diagram


CarCollection
+toString()(list:Car[],len:int): static String
+countAntique(list:Car[],len:int): static int
+printExpensiveCar(list:Car[],len:int): static void
+printCarsWithMake(list:Car[],len:int,make:String): static void
+cheapestCar(list:Car[],len:int): static Car
Car
-vin: String
-make: String
-model: String
-cost: double
-year: int
+getCost(): double
+getMake(): String
+isExpensive(): boolean
+isAntique(): boolean
+toString(): String
TestCarList
+main(): static void