# Lab: Defining Classes

This document defines the lab for the ["Java Advanced" course @ Software University](https://softuni.bg/modules/59/java-advanced). Please submit your solutions (source code) to all below-described problems in [Judge](https://judge.softuni.org/Contests/1517/Defining-Classes-Lab).

# I: Defining Classes

## Car Info

Create a class named Car.

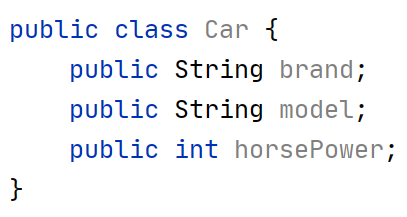
The class should have **public** fields for:

* Brand: **String**
* Model: **String**
* Horsepower: **int**

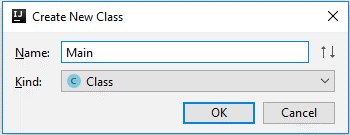
Create a **new class** and ensure **proper naming:**



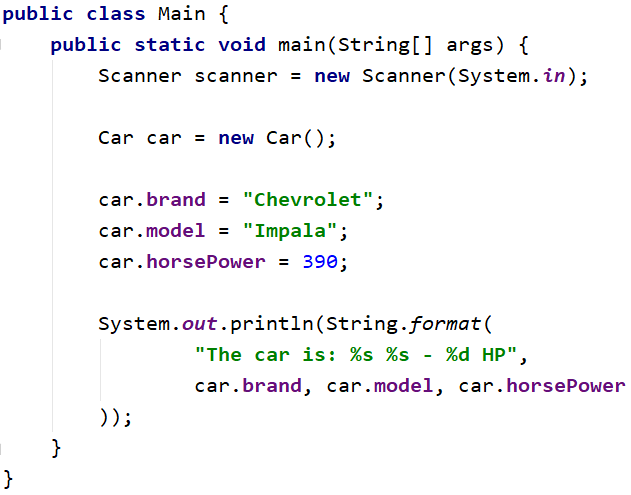
Define the fields:



Create a Test client in the **same** **package:**



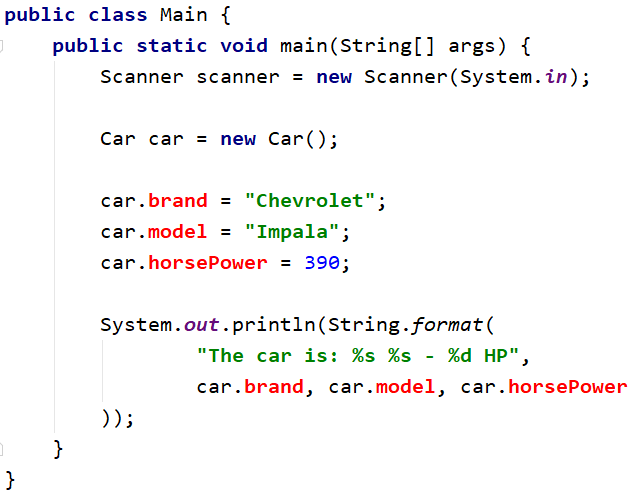
You should now be able to use your class:



### Private Fields

Change the access modifiers of all class fields to **private.**

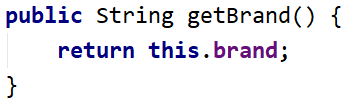
When done, go back to the main method you should have **compilation** errors like this:



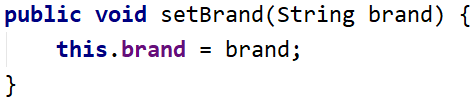
#### Getters and Setters

**Create getters** and **setters** for each class field.

**Getter** for the car brand:



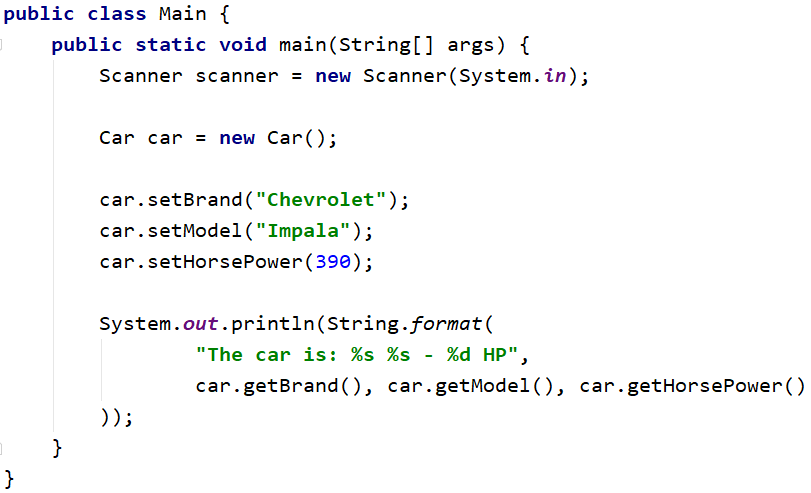
**Setter** for the car brand:



Do the same for **all** the fields.

**Fix Main Method**

You should **set** and **get** the **values** by using the correct methods



### Create Car Info Method

This method should return the info about any car object in the following format:

"**The car is: {brand} {model} – {horsePower} HP.**"

You have to figure out how to create a method and use it in the outside code, as shown below:



**Test the Program**

Read cars objects, add them to the collection of your choice, and print each one on the console using the **carInfo()** method. The input consists of a single integer **N,** the number of lines representing car objects. On each line you will read car info in the following format "**{brand} {model} {horsePower}**"separated by single space.

#### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Chevrolet Impala 390  Mercedes Benz 500  Volga 24 49 | The car is: Chevrolet Impala - 390 HP.  The car is: Mercedes Benz - 500 HP.  The car is: Volga 24 - 49 HP. |
| 5  This Car 1  Was Made 2  Only For 3  Test Purposes 4  No Way 5 | The car is: This Car - 1 HP.  The car is: Was Made - 2 HP.  The car is: Only For - 3 HP.  The car is: Test Purposes - 4 HP.  The car is: No Way - 5 HP. |

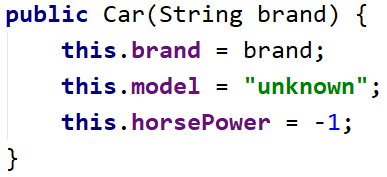
# II: Constructors

## Car Constructors

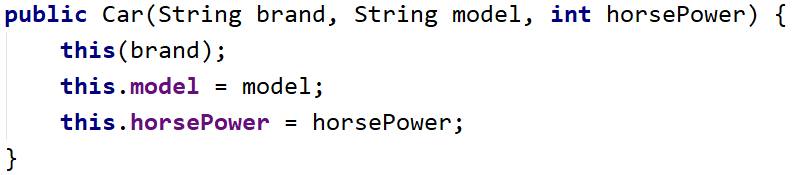
Make proper constructors for the Car class so you can create car objects with a different type of input information.

If you miss information about the field of **type String** set the value to "**unknown**",and for an **integer,** fieldset **-1.**

First, **declare** a **constructor** which takes only the car brand as a parameter:



Also, create a **constructor** which **sets** all the **fields**:



Read information about cars the same way as the previous task, however, this time uses **constructors** to create the objects, not creating an object with the **default** constructor. You should be able to handle **different** **types** of input, the format will be the same as the previous task, but this time some of the data may be missing. For example, you can get only the car **brand** – which means you have to set the car model to "**unknown**"and the Horsepower value to **-1**. There will be lines with **complete** car data, declare constructor which sets all the fields.

You have to add the car objects to a **collection** of your choice and print the data as in the previous task. The input will **always** have one or three elements on each line.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  Chevrolet  Golf Polo 49 | The car is: Chevrolet unknown - -1 HP.  The car is: Golf Polo - 49 HP. |
| 4  Was  Only For 3  Test Purposes 4  No Way 5 | The car is: Was unknown - -1 HP.  The car is: Only For - 3 HP.  The car is: Test Purposes - 4 HP.  The car is: No Way - 5 HP. |

## Bank Account

Create class BankAccount.

The class should have **private fields** for:

* Id: **int** (Starts from **1** and **increments** for every **new** **account**)
* Balance: **double**
* Interest rate: **double** (Shared for all accounts. **Default value: 0.02**)

The class should also have **public** methods for:

* setInterestRate(double interest): void (static)
* getInterest(int Years): double
* deposit(double amount): void

Create a test client supporting the following commands:

* **Create**
* **Deposit {Id} {Amount}**
* **SetInterest {Interest}**
* **GetInterest {ID} {Years}**
* **End**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| Create  Deposit 1 20  GetInterest 1 10  End | Account ID1 created  Deposited 20 to ID1  4.00 |  |
| Create  Create  Deposit 1 20  Deposit 3 20  Deposit 2 10  SetInterest 1.5  GetInterest 1 1  GetInterest 2 1  GetInterest 3 1  End | Account ID1 created  Account ID2 created  Deposited 20 to ID1  Account does not exist  Deposited 10 to ID2  30.00  15.00  Account does not exist | Sets the global interest rate to 1.  Prints interest for a bank account with id 1 for 1 year period. |

### Solution

Create the class as usual and create a **constant** for the default interest rate:



Create the static and non-static fields, **all private:**



Set the id of an account upon creation while **incrementing** the global account count:



Create a setter for the global interest rate. Making the method **static** will let you access it through the class name:



Implement deposit() and getInterest():

