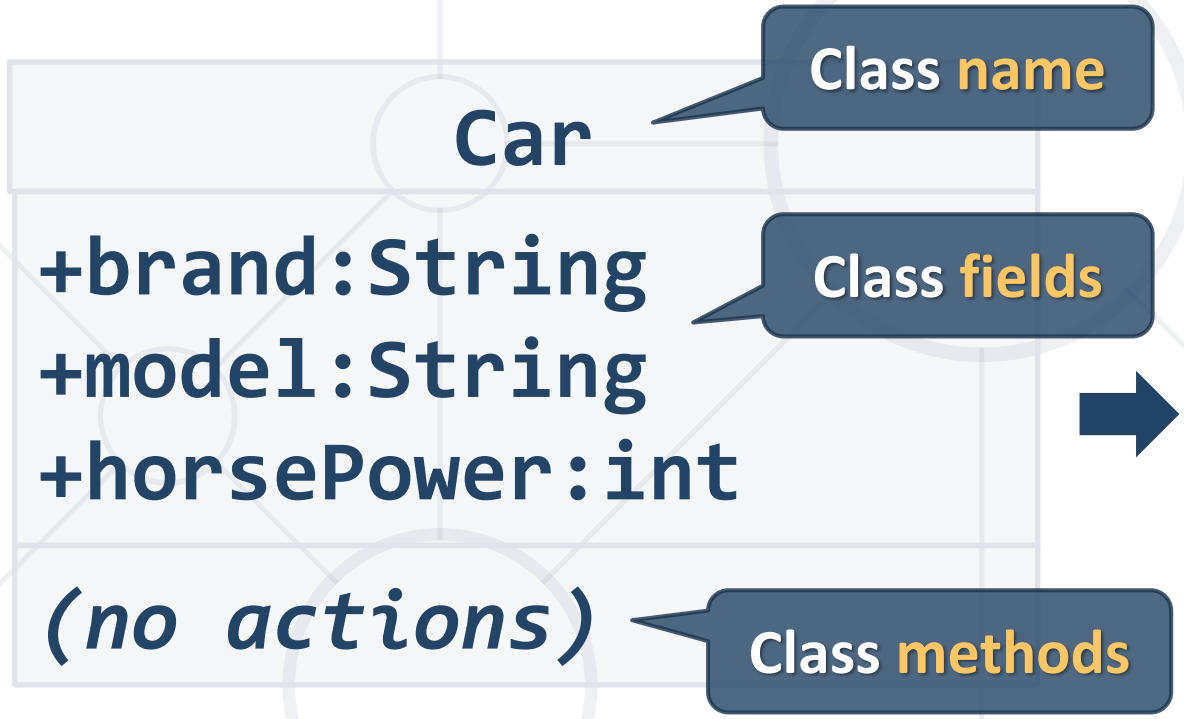
Defining Classes

Exercises for: Private and Public Members, Properties, Methods, Constructors, Static Methods and Fields

Problem: Define a "Car" Class

- Create a simple class **Car**

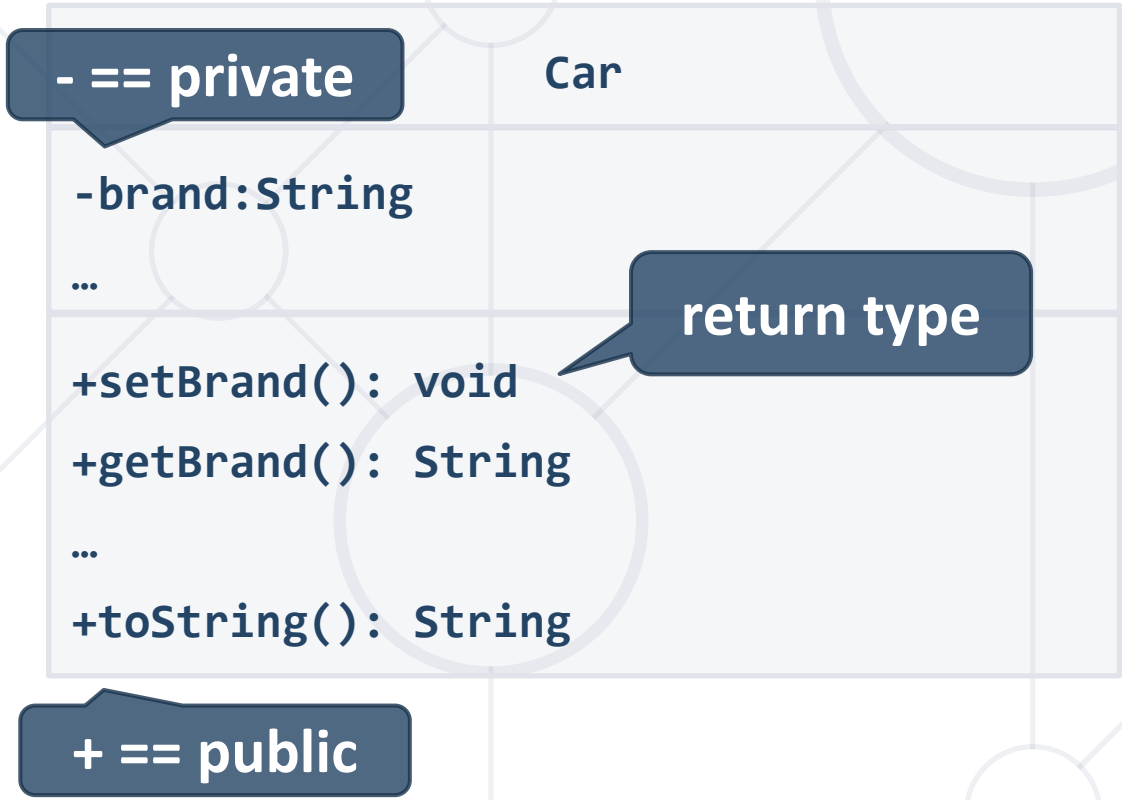



```
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        Car car = new Car();

        car.brand = "Chevrolet";
        car.model = "Impala";
        car.horsePower = 390;

        System.out.println(String.format(
                "The car is: %s %s - %d HP",
                car.brand, car.model, car.horsePower
        ));
    }
}
```

- Ensure proper naming!

Problem: Car Info

- Define a class **Car**, which holds **brand**, **model** and **horsePower**
  - Define **toString()** to get the car info in format: **brand:model:hp**




```
Car car = new Car();
car.setBrand("Tesla");
car.setModel("Model S");
car.setHorsePower(503);
System.out.println(car);
// Tesla:Model S:503
```

Problem: Car with Constructors

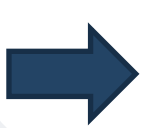
- Define a class **Car**, which holds **brand**, **model** and **horsePower**
  - Define 3 constructors:
    - **parameterless** constructor
    - by **brand** + **model**
    - by **brand** + **model** + **horsePower**
  - Define **toString()** to get the car info in format **brand:model:hp**
    - If **hp** is missing, use **brand:model**

Car
-brand:String -model:String -horsePower:int
+Car() +Car(String brand, String model) +Car(String brand, String model, int horsePower) +toString():String

Problem: Car with Constructors (2)

- Using the **Car** class, write the following Java program:
  - Read from the console: **brand**, **model**, **horsePower**
    - The **horsePower** is optional and can be missing (empty line)
  - Create a **Car** object with the specified parameter
  - Print the car info, by invoking indirectly **Car.toString()**

```
BWM
X3
250
```



```
BMW:X3:250
```

```
Ford
Figo
```

```
Ford:Figo
```

Problem: Bank Account

- Create a class **BankAccount**, with the following **fields** and **methods**:
  - **id** – unique account ID → starts from 1 and increases automatically
  - **balance** – the current amount of money deposited in the account
  - **interest** – global annual interest rate for all accounts, default = 0.15
  - **setInterest(rate)** – changes the annual interest rate for all accounts
  - **deposit(x)** – adds given amount **x** to the current account balance
  - **calcInterest(months)** – calculates the interest for the current deposit, with the current interest rate: **balance * interest * months / 12**

Problem: Bank Account – Input / Output


```
create
deposit 1 20000
create
deposit 2 5000
calc-interest 1 6
set-interest 0.353
calc-interest 2 18
calc-interest 3 8
deposit 2 3000
calc-interest 2 6
end
```

```
Account #1 created
Deposited 20000.00 to account #1
Account #2 created
Deposited 5000.00 to account #2
Interest: 1500.00
Interest rate changed: 0.35
Interest: 2647.50
Account #3 not found
Deposited 3000.00 to account #2
Interest: 1412.00
Goodbye
```

20000 * 0.15 * 6 / 12

5000 * 0.353 * 18 / 12

balance = 8000

8000 * 0.353 * 6 / 12

Solution: Bank Account


```
public class BankAccount {
  private static final double DEFAULT_INTEREST = 0.15;

  private static double interestRate = DEFAULT_INTEREST;
  private static int bankAccountsCount = 0;

  private int id;
  private double balance;

  // continue…
```

Next Steps

SoftUni

- Join the SoftUni "Learn To Code" Community

https://softuni.org

- Access the Free Coding Lessons
- Get Help from the Mentors
- Meet the Other Learners