GET SMART: WITH JAVA PROGRAMMING

YAMAN OMAR ALASHQAR

SYSTEM.OUT.PRINTLN(“WELCOME TO THIS COURSE”);

GET SMART: WITH JAVA PROGRAMMING

INTRODUCTION TO CLASSES

CONTINUE

- Real Life Objects, which has some Properties, and Methods
- In real life, a car is an object.
- A car has properties like weight and color, and methods like start and stop

Properties	Methods
car.name = Fiat	car.start()
car.model = 500	car.drive()
car.weight = 850kg	car.brake()
car.color = white	car.stop()

- All cars have the same properties, but the property values differ from car to car.
- All cars have the same methods, but they are performed at different times.
- Methods are actions that can be performed on objects

ACCESSING OBJECT PROPERTIES ??

- A method is a block of code, designed to perform a particular task

- Why methods?
  - You can reuse code: Define the code once, and use it many times.
  - You can use the same code many times with different arguments, to produce different results

- arguments are the values received by the function when it is invoked

- Inside the method, the arguments (parameters) behave as local variables
- Variables declared within a method, become LOCAL to the function.
- Local variables can only be accessed from within the function.

- Declared methods are not executed immediately.
  - They are "saved for later use",
  - and will be executed later, when they are invoked (called upon)

- When Java reaches a return statement, the method will stop executing
- If the method was invoked (called) from a statement, Java will "return" to execute the code after the invoking statement.
- Method’s often compute a return value.
    - The return value is "returned" back to the "caller“

- In a method, this refers to the "owner" of the function

Java is pass-by-value.

That means pass-by-copy.

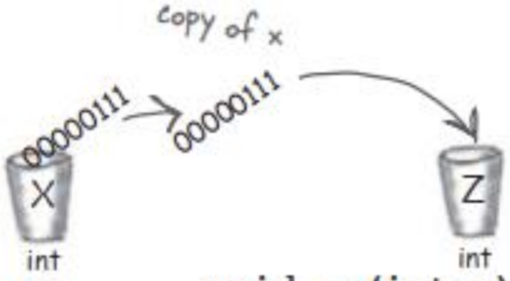
```
int x = 7;
```

00000111
X
int

1 Declare an int variable and assign it the value '7'. The bit pattern for 7 goes into the variable named x.

```
void go(int z){ }
```

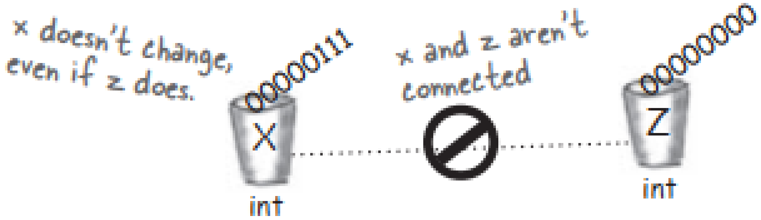
2 Declare a method with an int parameter named z.



```
foo.go(x);
```

```
void go(int z){ }
```

3 Call the go() method, passing the variable x as the argument. The bits in x are copied, and the copy lands in z.



```
void go(int z){
    z = 0;
}
```

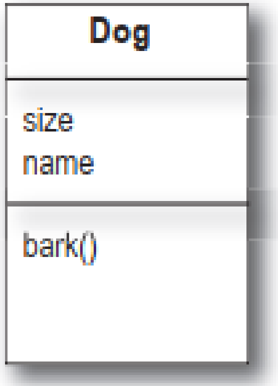
4 Change the value of z inside the method. The value of x doesn't change! The argument passed to the z parameter was only a copy of x.

The method can't change the bits that were in the calling variable x.

The size affects the bark

A small Dog’s bark is different from a big Dog’s bark.

The Dog class has an instance variable *size*, that the *bark()* method uses to decide what kind of bark sound to make.



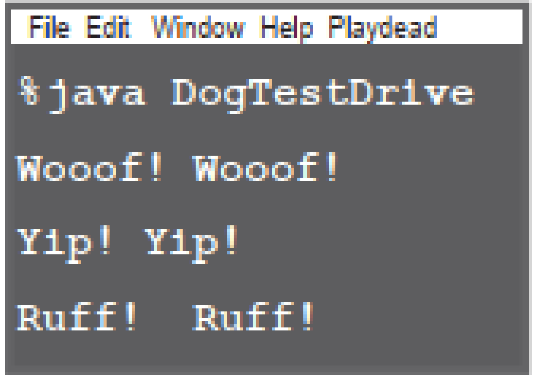
Dog

size
name

bark()

```
class Dog {
  int size;
  String name;

  void bark() {
    if (size > 60) {
       System.out.println(“Wooof! Wooof!”);
    } else if (size > 14) {
       System.out.println(“Ruff!  Ruff!”);
    } else {
       System.out.println(“Yip! Yip!”);
    }
  }
}
```

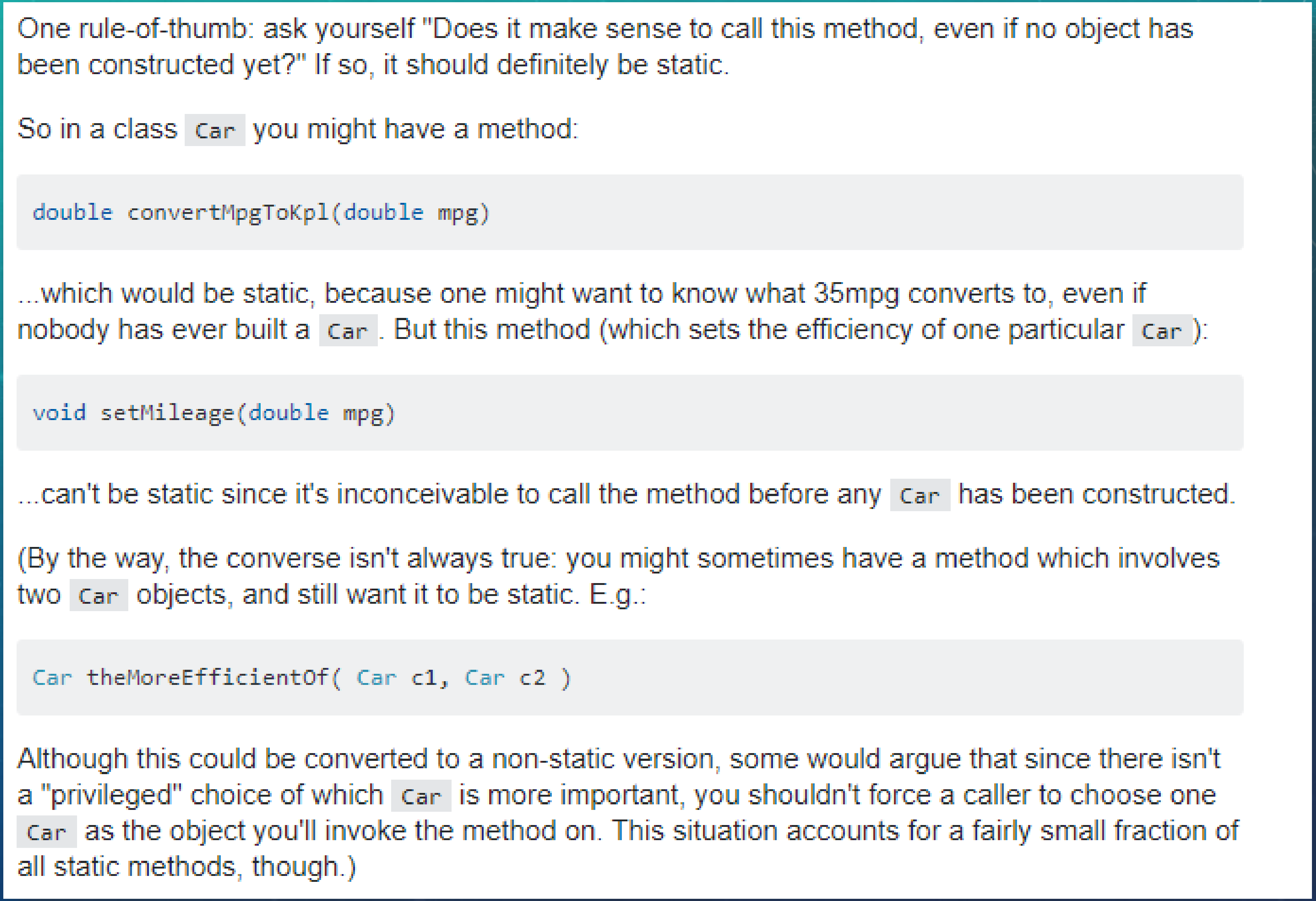
```
class DogTestDrive {

   public static void main (String[] args) {
     Dog one = new Dog();
     one.size = 70;
     Dog two = new Dog();
     two.size = 8;
     Dog three = new Dog();
     three.size = 35;

     one.bark();
     two.bark();
     three.bark();
   }
}
```



```
%java DogTestDrive
Wooof! Wooof!
Yip! Yip!
Ruff!  Ruff!
```

Exercise: Make each dog bark 3 times

- The **static members** are **used** to store data independent of any instance of an object



One rule-of-thumb: ask yourself "Does it make sense to call this method, even if no object has been constructed yet?" If so, it should definitely be static.

So in a class `Car` you might have a method:

```
double convertMpgToKpl(double mpg)
```

...which would be static, because one might want to know what 35mpg converts to, even if nobody has ever built a `Car`. But this method (which sets the efficiency of one particular `Car`):

```
void setMileage(double mpg)
```

...can't be static since it's inconceivable to call the method before any `Car` has been constructed.

(By the way, the converse isn't always true: you might sometimes have a method which involves two `Car` objects, and still want it to be static. E.g.:

```
Car theMoreEfficientOf( Car c1, Car c2 )
```

Although this could be converted to a non-static version, some would argue that since there isn't a "privileged" choice of which `Car` is more important, you shouldn't force a caller to choose one `Car` as the object you'll invoke the method on. This situation accounts for a fairly small fraction of all static methods, though.)

- Write a method that returns the largest element in a list/array
- Write a method that checks whether an element occurs in a list/array
- Write a method that returns the elements on odd positions in a list
- Write a method to return all odd values in a list of int
- Write a method that concatenates two lists
- Write a method that combines two lists by alternatingly taking elements,
    - e.g. [a,b,c], [x,y,z] → [a,x,b,y,c,z]
- Write a method that takes a number and returns a array of its digits.
    - So for 2342 it should return [0,0,2,3,4,2]
    - and for 12345 return [0,1,2,3,4,5]
    - and for 25 return [0,0,0,0,2,5]