Lesson 2: Objects and Classes

Introduction to Programming

Academic year 2023-2024

Objects and classes. Review

- Elements in the model are the **objects** that appear in the **problem domain**
- **Objects** contain **properties** (*attributes*) and **behavior** (*methods*).

Object

Properties

- Color
- Speed
- Size
- Fuel
- …

Behavior

- Stop
- Turn right
- Turn left
- Start
- …

Objects and classes. Review

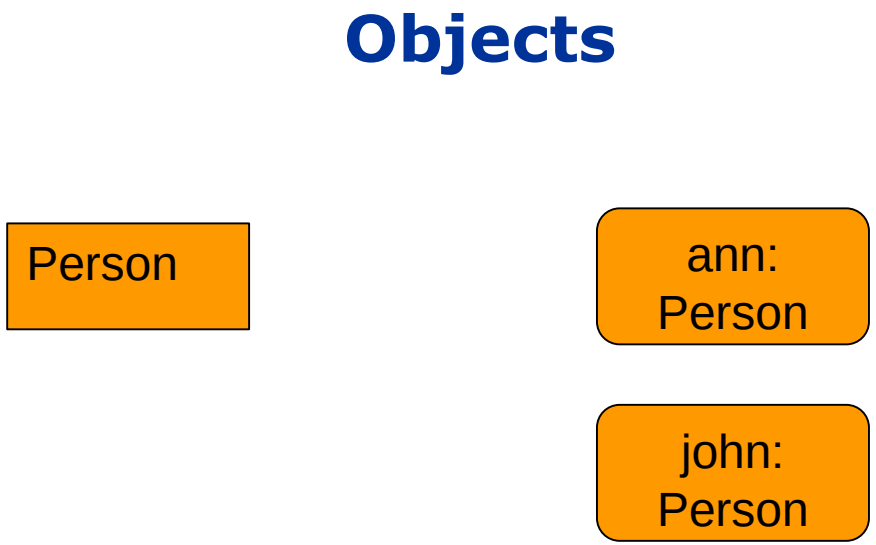
- **Objects** are organized into **categories**

A **class** defines the structure (**attributes**) and behavior (**methods**) of each and every **object** from that class.

- A **class** describes, in an abstract way, all the **objects** of a given **type**.

Creating objects

- Once a class is available, we can create objects (or **instances**) of that class.
- Example:

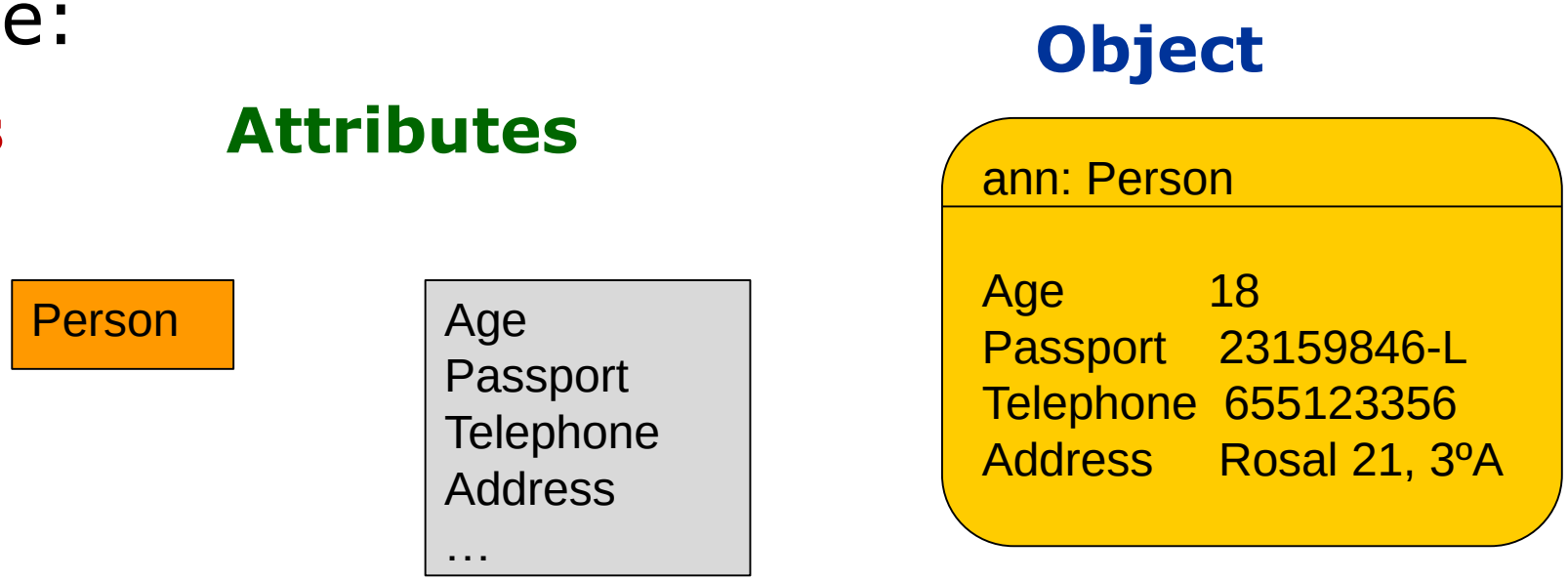


Convention

Class names begin with **upper case** while **object names** begin with **lower case**.

Status of an object

- **Objects** have a **state**.
- The state is represented by the values stored in their attributes (or properties).
- Example:



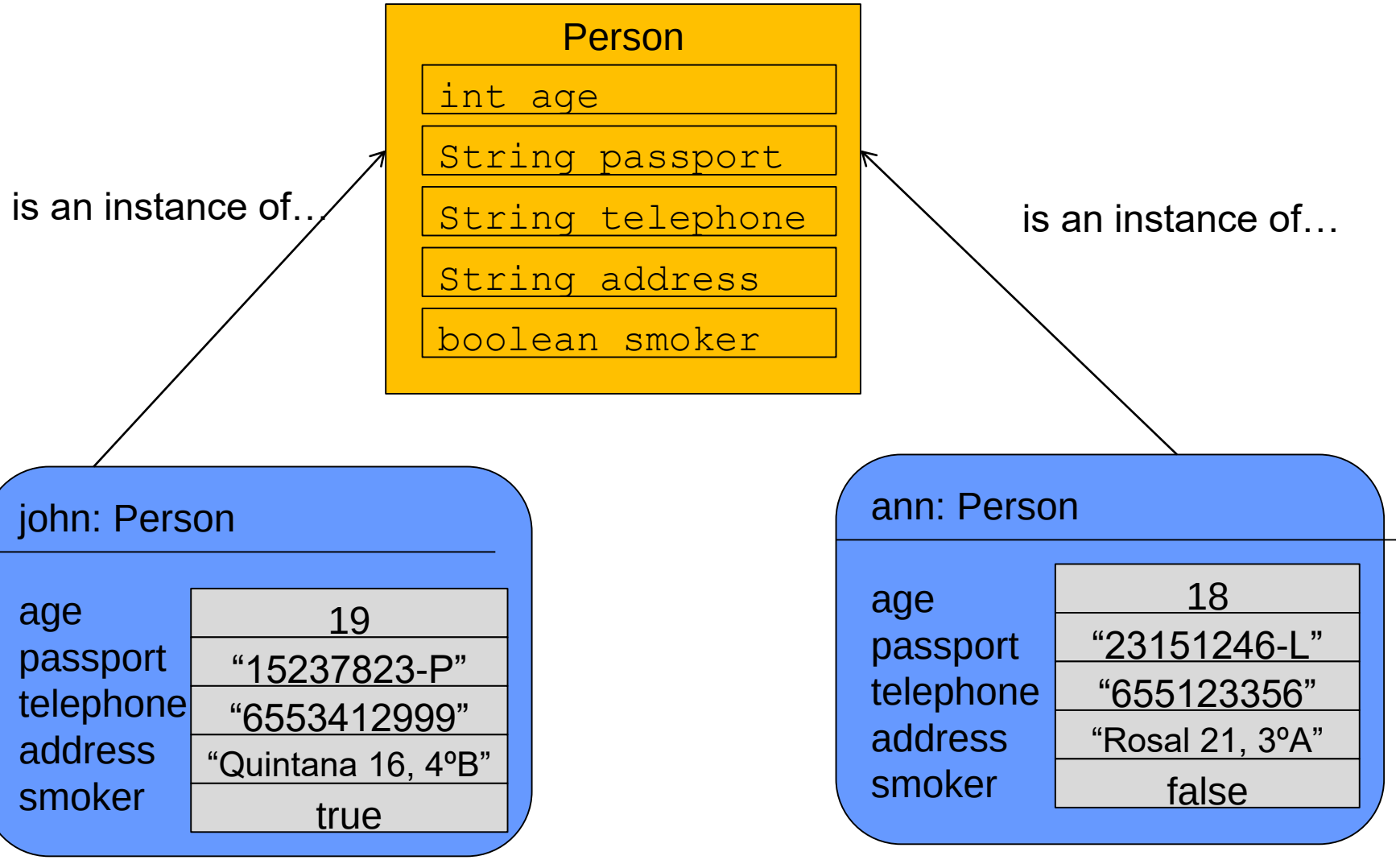
Data types

- Types specify the nature of a piece of data.
  - `int` type refers to integer numbers.
  - `String` type refers to text fragments (a word or a phrase).
  - `boolean` (logical type) has only two values: `true` and `false`.
  - There are other types: `double`, `float`, `char`, …

Multiple instances

- We can create **many similar objects** from **one single class** (as many as we want).
- **All objects** from the **same class** have the **same attributes** (or properties).
  - The number, data type, and names of the attributes are the same for all objects of a given class.
  - The attribute values can be different for each object.

Example of multiple instances



Multiple instances

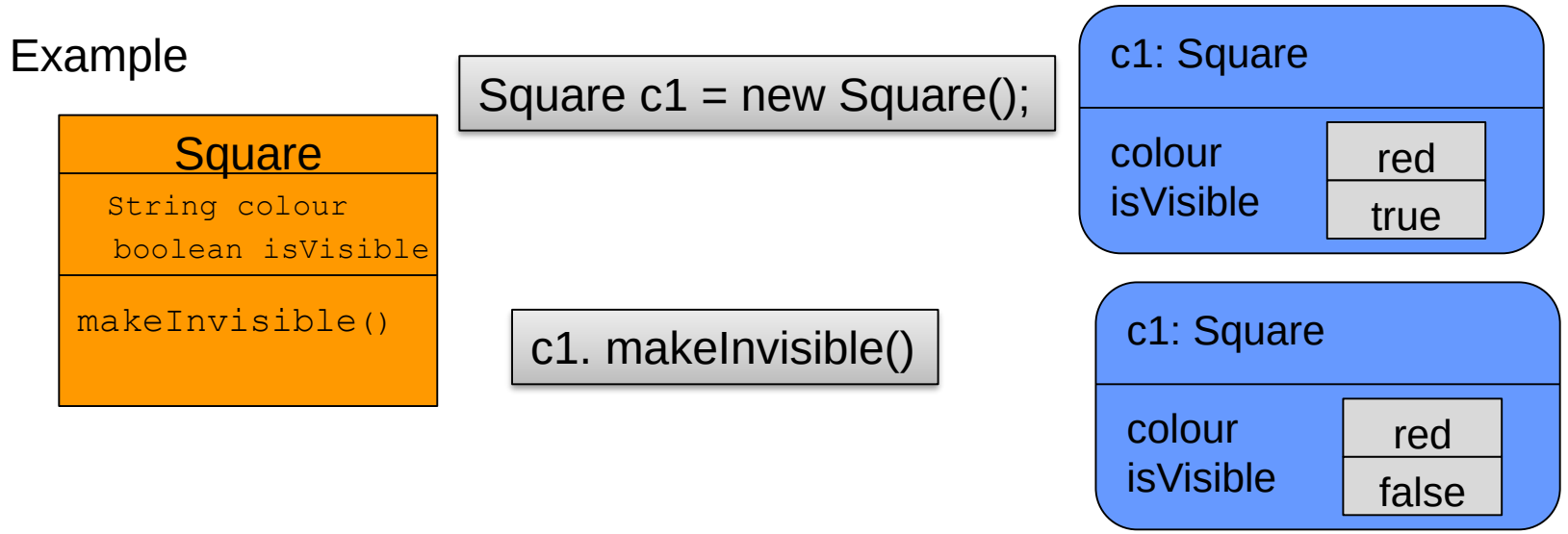
- The **number**, **data type** and **name** of the attributes are **defined in the class**, not in the objects.
  - The `Person` class defines that every person object will have 5 attributes with the previously mentioned names. It also defines the data types for each of those attributes.
  - When an object from the person class is created, it is created with all of those attributes.
- **Attribute values are stored in each object**.
  - For instance, each person will have an age different from that of other people.

Methods (I)

- They provide the **operations** that we can rely on to **manipulate an object**.
- We can **communicate with objects** calling their methods.
- Usually, **objects do something when calling one of their methods**.

Methods (II)

- **Methods are defined in the object's class**. All objects from a given class have the same methods.
- **Methods are called *on* objects.** The object whose method is called is the one that will probably change.



Methods (III)

- There are some **actions** which is wise to perform **when creating an object** (e.g. initialization of attributes).
- That is why there is **one method** which is **called automatically** when **creating** an object: the **constructor**.
  - So, we do not forget to ask the object to carry out those actions.

Constructor method

- Responsible of **creating** and **initializing** the object
- **Invoked automatically**
- *Signature*
  - Without parameters

```
public Square()
```

  - It can have parameters

```
public Square(String colour, boolean visible)
```

Parameters (I)

- **Methods** may have **parameters** to receive **additional information** to perform an action.
- A method must define the **data type** for its **parameters**.

```
public void changeColour(String colour)
```

Parameters (II)

- The **header** of a method is called ***signature*** and provides all the **information needed to call** that method.

```
public void change(String colour, int size)
```

- The fragment between brackets (…) is the information given about the required parameter(s).

- Every **parameter** needs a **data type** and a **name**.

Return values

- Methods can **return some information** by means of a **return value**.
- The **method's signature** indicates if the method **returns or not** a value and its **data type**.

```
String getName()
```

  - `String` before the method's name is the return type.

```
void changeColour(String colour)
```

  - The `void` keyword means that this method does not return any value.

Set and get methods

- Used to obtain and modify atributes's values
- A **get method** returns the value of an attribute. It is used to **obtain information** from an object. Its signature is:

```
public String getName()
```

- A **set method** changes the value of an attribute. It is used to **modify the state** of an object. Its signature is

```
public void setName(String newName)
```

Creating objects in a program

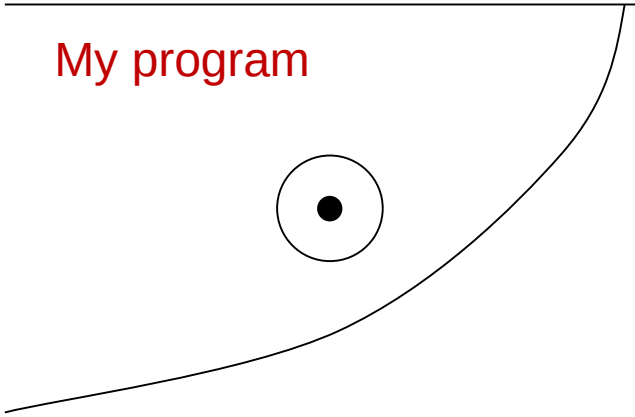
- A Java program could need hundreds or thousands of objects.
- How can we create an object within a program?

```
Person p1 = new Person();
```

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”



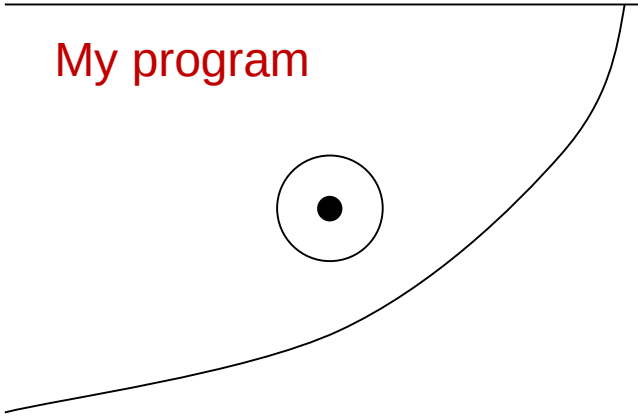
What is “**Person**” doing ?

It allocates some memory to store references to objects belonging to the `Person` class (references, **not** objects).

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”



How can I know that a reference is being used?

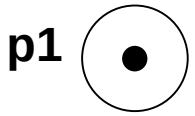
Because `Person` is not a primitive data type like `int`, `boolean`, `char`.

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”

My program



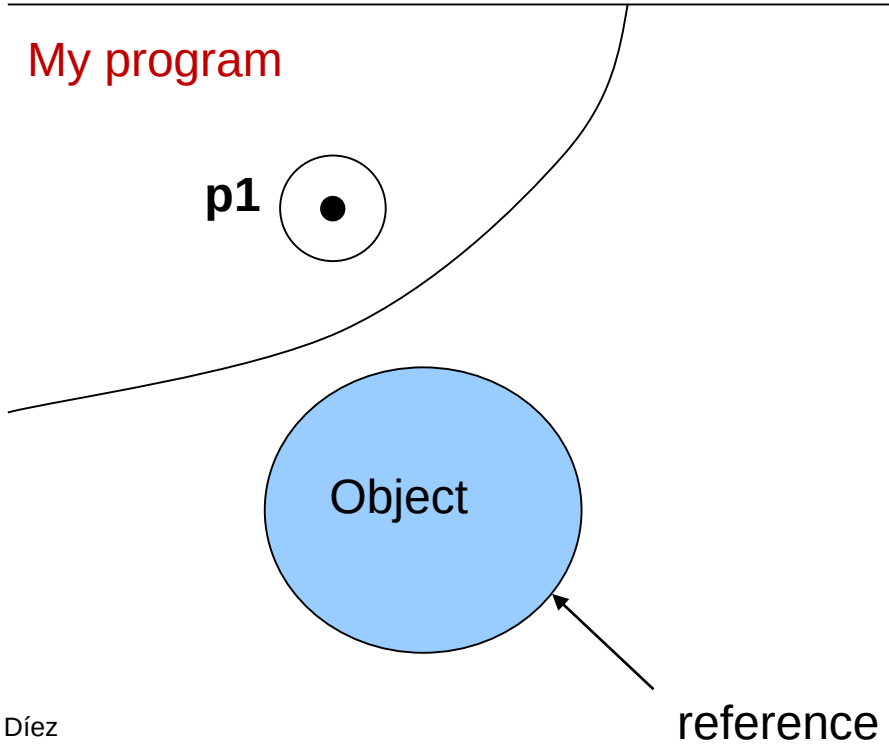
What’s the purpose of `p1`?

It provides a name to the memory allocated by `Person`. This way, we can use that memory area every time we need it.

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”



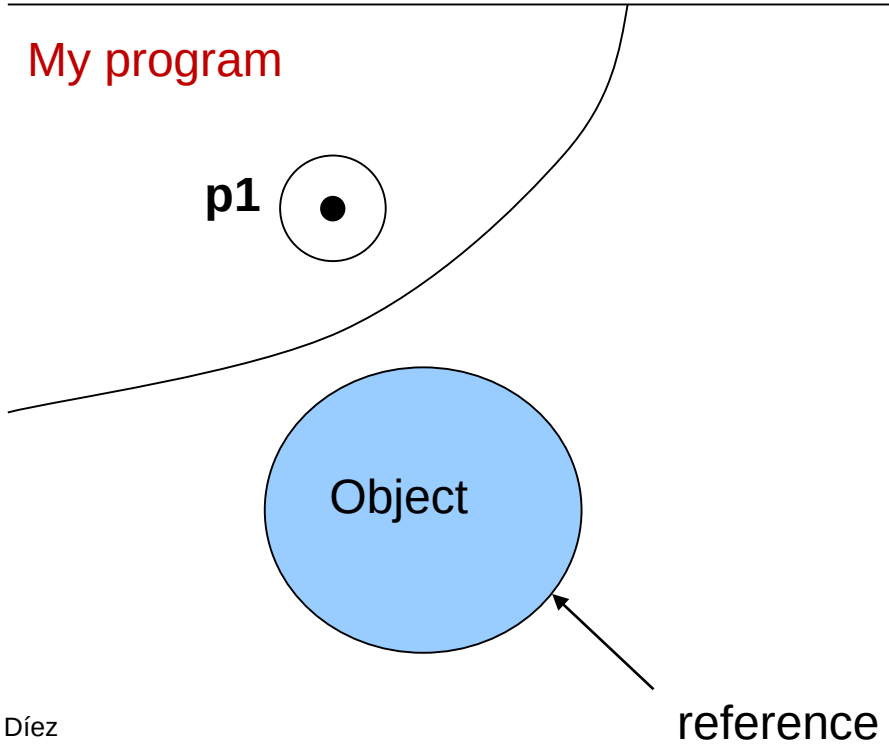
What’s “`new`” doing?

It creates a basic object belonging to the `Object` class. Besides, it creates a reference to that object.

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”



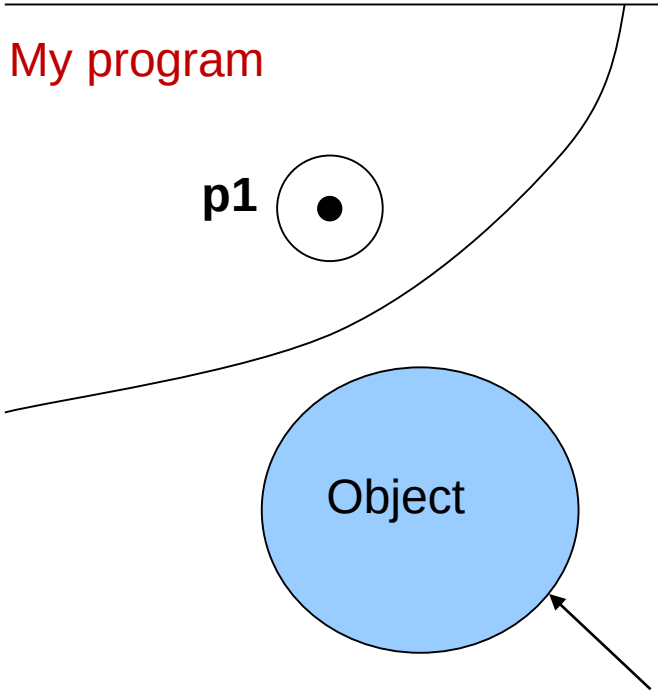
What’s “`Person()`”?

- It is a method signature.
- It is a special method: the **constructor**.

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”



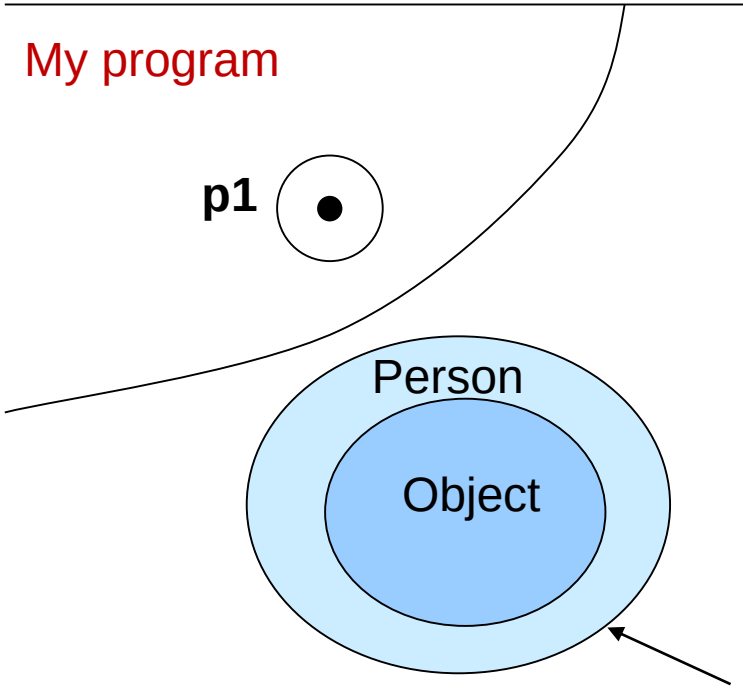
How can I know it is a **constructor**?

Because it has the same name as the class definition: **`Person`**.

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”



What’s “`Person()`” doing?

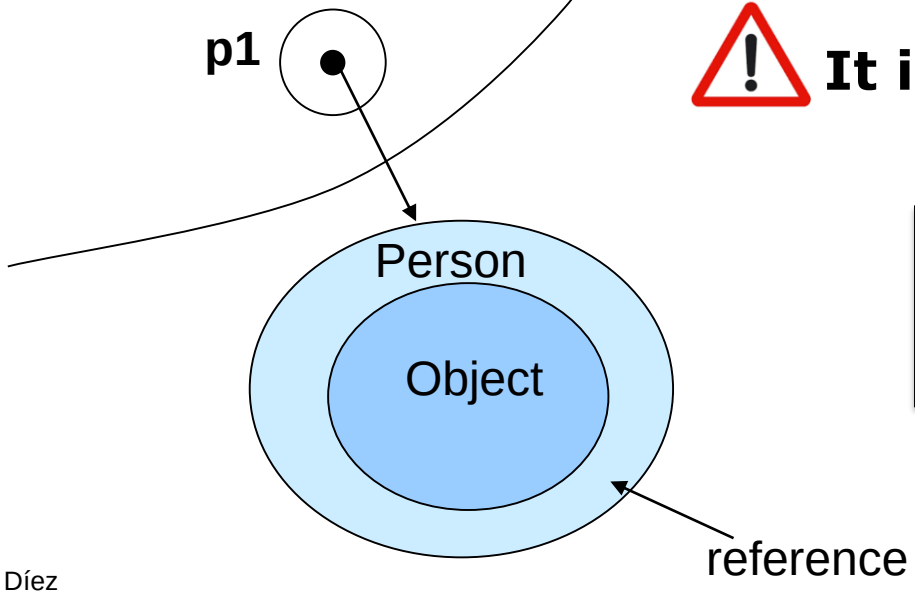
It provides a wrapping around the object created with **`new`** so it turns into a **`Person`** object.

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”

My program



What’s “=“ doing ?

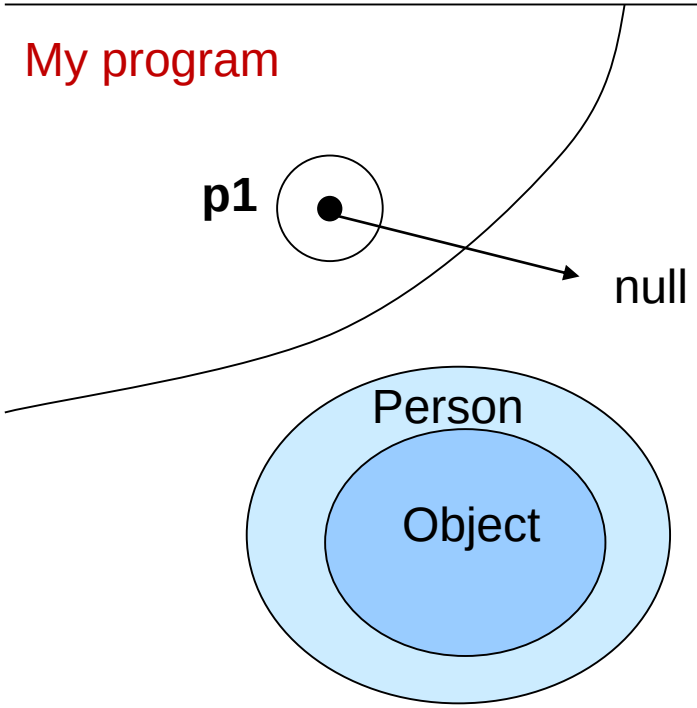
It is not the math symbol !

It **copies** the reference to the just created **`Person`** object into the memory area named **`p1`**.

Creating objects in a program

```
Person p1 = new Person();
```

Inside the “memory”



What happens to the `Person` object if we execute the following sentence?

```
p1 = null ;
```

The object still exists but the reference is lost and, thus, we cannot have access to it.

Exercise (homework)

- Represent in a graphical way the meaning of the following blocks.

BLOCK 1

```
Person p1 = new Person();

Person p2 = new Person();

p1 = p2;
```

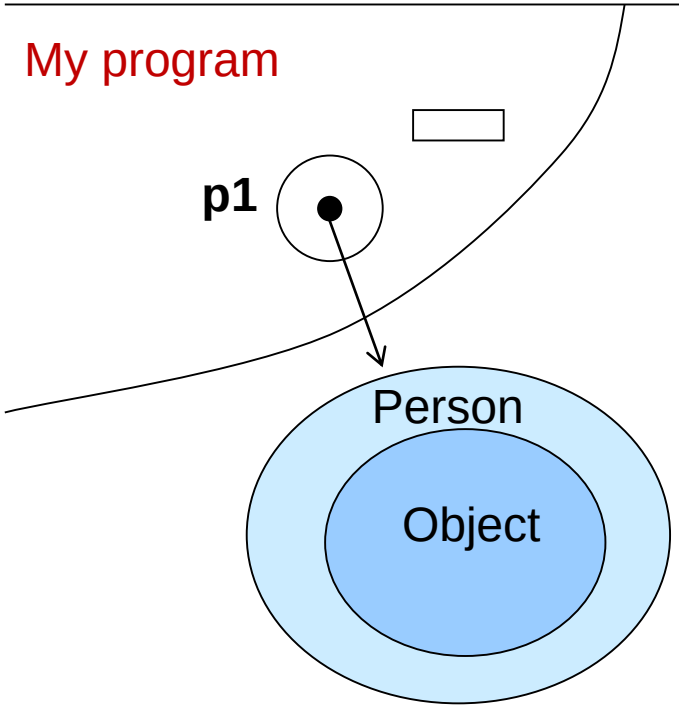
BLOCK 2

```
Person ann;

Person john = new Person();

Person peter = new Person();

Person mary = new Person();

 ann = john;

 john = null;
```

Calling methods in a program

```
int age = p1.getAge();
```

Inside the “memory”



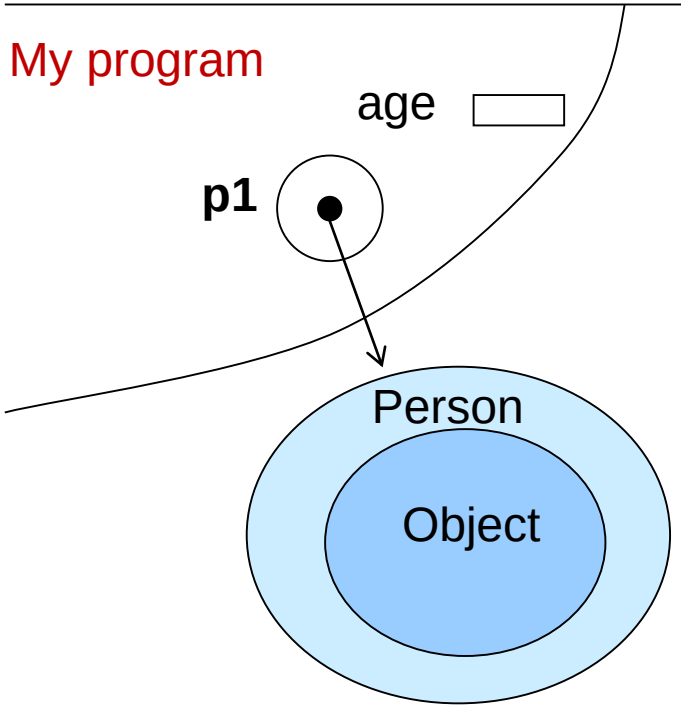
What is “`int`” doing ?

It allocates some memory to store an integer number.

Calling methods in a program

```
int age = p1.getAge();
```

Inside the “memory”



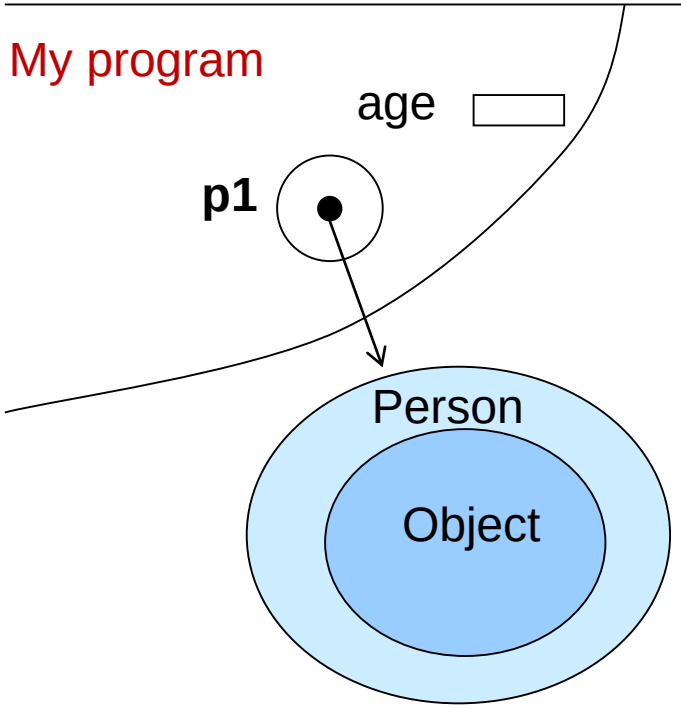
What’s “**age**” doing?

It assigns a name to the memory space allocated to store an `int` value so it can be used every time we need it.

Calling methods in a program

```
int age = p1.getAge();
```

Inside the “memory”



What’s “`p1.`” doing ?

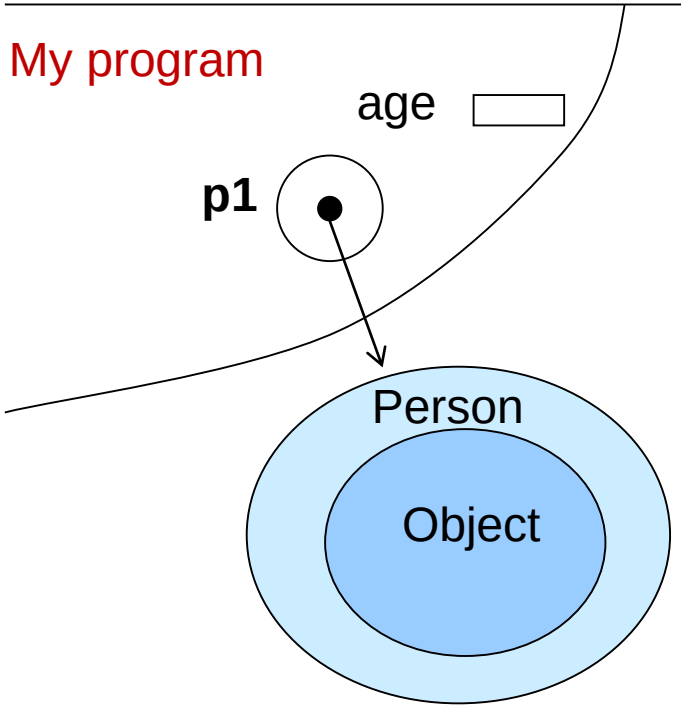
We are accessing the **`Person`** object by means of the reference stored in the variable **`p1`**.

The dot **`.`** is an operator to have access to the methods (and attributes of the object).

Calling methods in a program

```
int age = p1.getAge();
```

Inside the “memory”



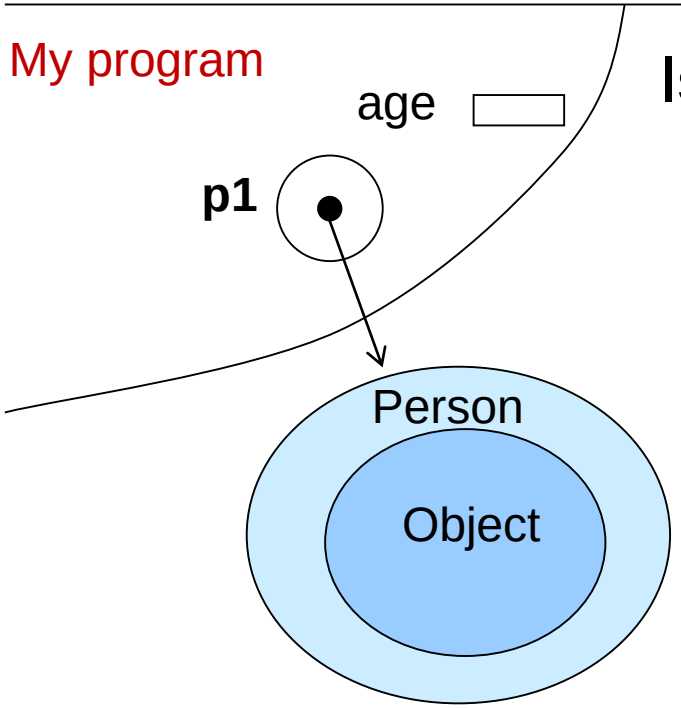
What’s “`getAge()`” ?

It’s an *invocation* to a method (or *method call*).

Calling methods in a program

```
int age = p1.getAge();
```

Inside the “memory”



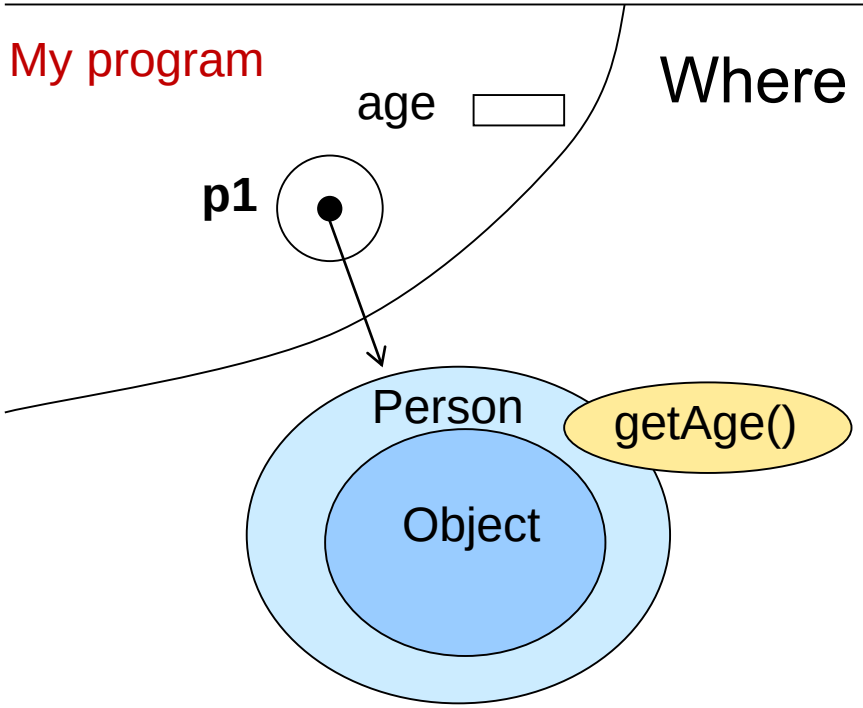
Is “**`getAge()`**” a constructor?

No. Constructors are called when creating new objects. We are not creating any new object here.

Calling methods in a program

```
int age = p1.getAge();
```

Inside the “memory”



Where is the method “**`getAge()`**?

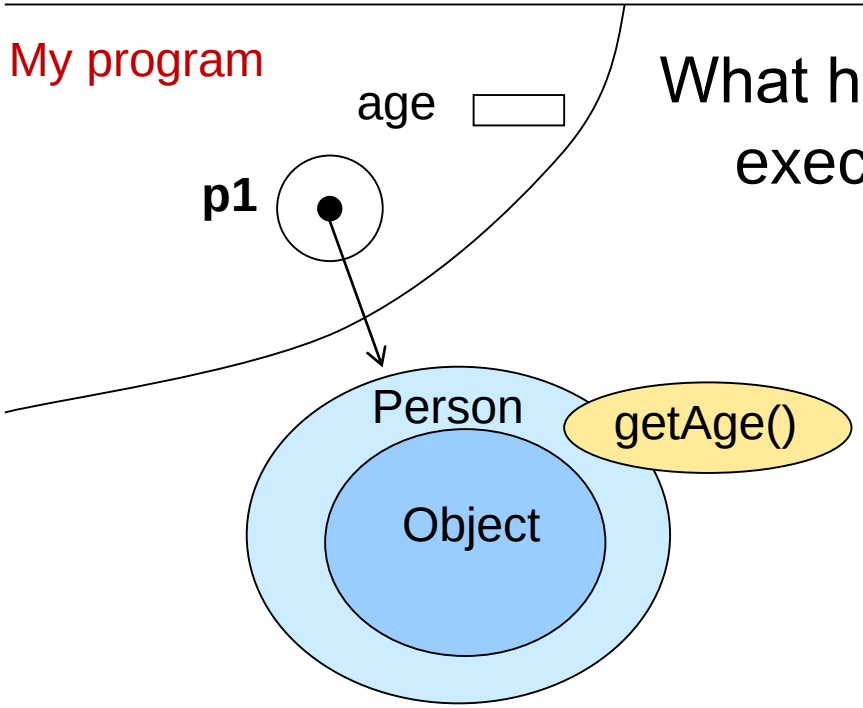
Methods are defined in classes.

Because **`p1`** refers to objects belonging to the **`Person`** class, the method **`getAge`** defined within that class is called.

Calling methods in a program

```
int age = p1.getAge();
```

Inside the “memory”



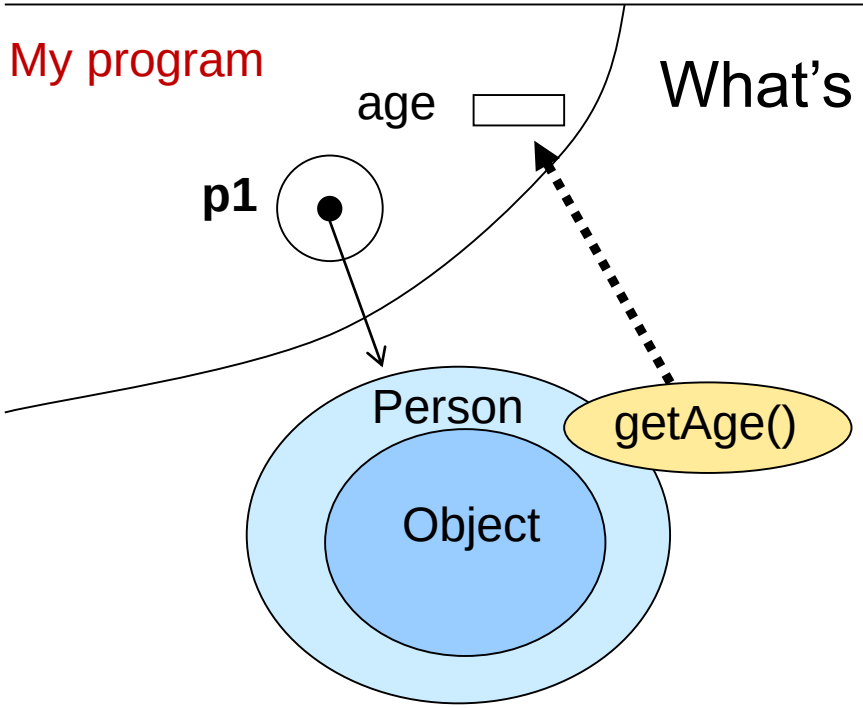
What happens when “**`p1.getAge()`**” is executed?

The method **`getAge()`** returns an integer.

Calling methods in a program

```
int age = p1.getAge();
```

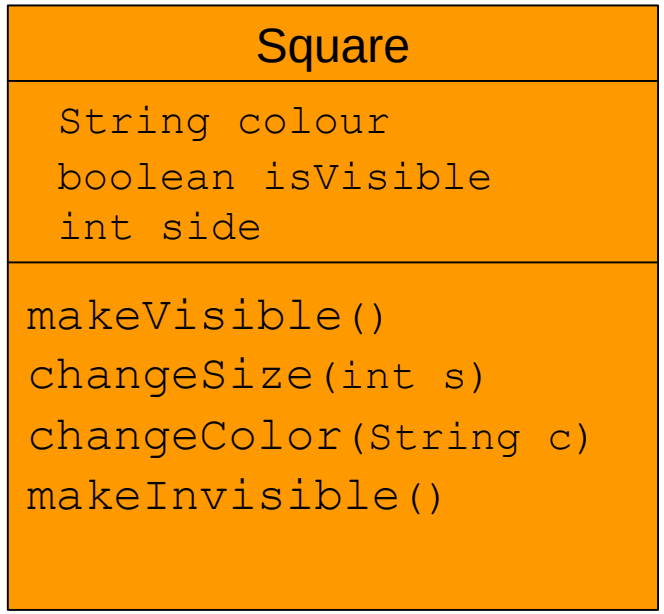
Inside the “memory”



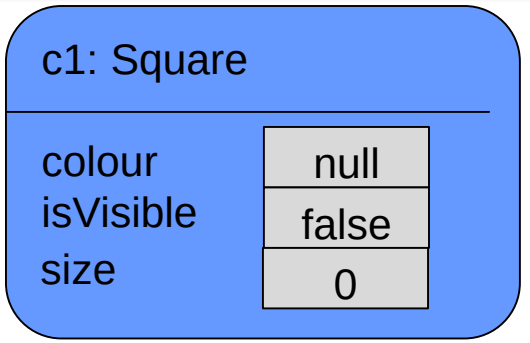
It is copying the integer returned by **`getAge()`** into the memory allocated to the variable **`age`** (which, by the way, was declared to store integer values).

Exercise (I)

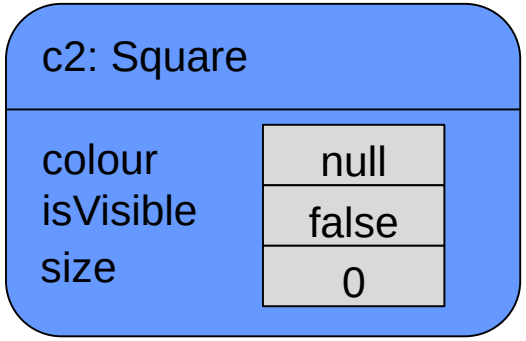
- Considering the following class definition and objects…




```
Square c1 =new Square();
```




```
Square c2 =new Square();
```



Exercise (II)

- Represent the status of the objects after executing the following sentences.

```
c1.changeSize (60);
c1.makeInvisible();
c1.changeColor("blue");
```

```
c2.changeSize (80);
c2.makeVisible();
c2.changeColor("green");
c1.makeVisible();
```

Use double quotation marks when writing a String literal value.