Lesson 3: Class definitions

Introduction to Programming

Academic year 2023-2024

Concepts

- Attributes or properties
- Constructors
- Methods
- Assignment sentences
- Conditional sentences
- Variables and operators
- Unit testing

Properties, constructors and methods

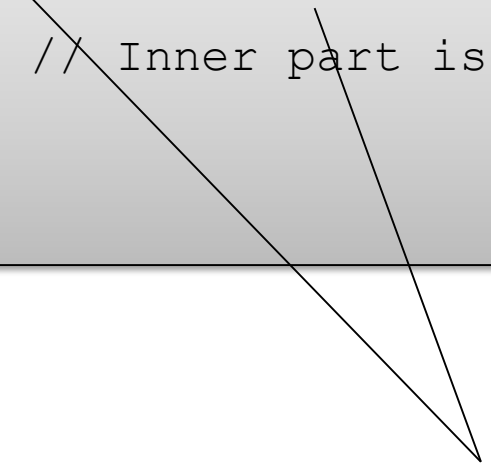
- **Code** in most of the **classes** can be divided into **two parts**.
  - A smaller outer wrapping which provides the class name.

```
public class TicketMachine {
    // Inner part is omitted
}
```

  - A larger inner part which does all of the work.

Properties, constructors and methods

```
public class TicketMachine {
    // Inner part is omitted
}
```



- ✓ **CANNOT** change the order
- ✓ You can omit the keyword `public`

Properties, constructors and methods

- In the inner part we define:
  - The **properties or attributes** ⇒ They store data (values) for each object to use them.
  - The **constructors** ⇒ They allow each object to be properly set up when created.
  - The **methods** ⇒ They implement the object's behaviour.
- They provide the class its particular features and behaviour.

Properties, constructors and methods

There is no pre-established order for them **but** you have to follow a given style.

```
public class ClassName {
  Properties or attributes
  Constructors
  Methods
}
```

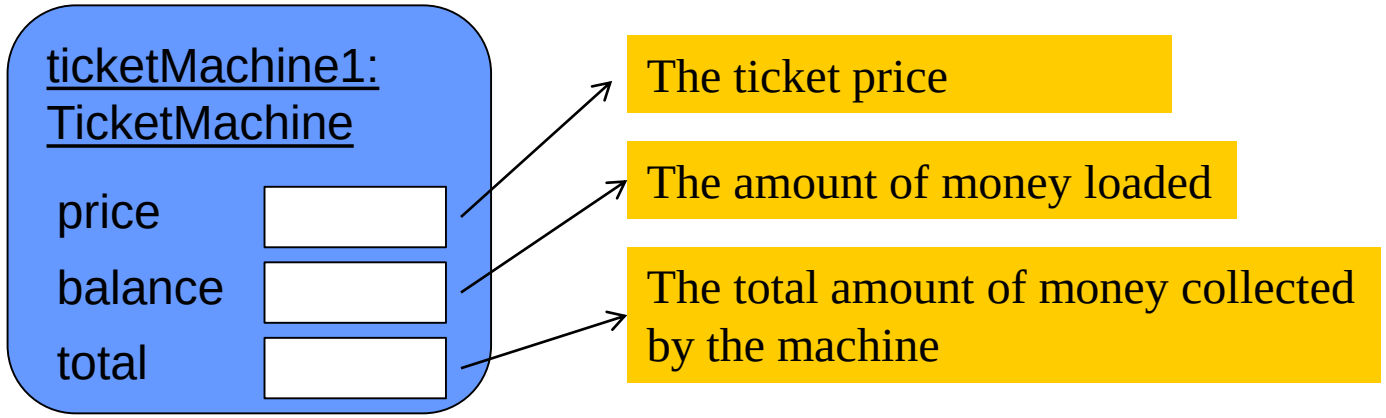
Properties

```
public class TicketMachine {
   private double price; // the ticket price
   private double balance;
   private double total;
}
```

- Each **property** has its own source code **declaration**.
- You can use **comments**:
- **One line** comments **//**
  - **multiline**, starting with **/*** and ending with ***/**

Properties or attributes

- They are also known as **instance variables** or **fields**.
- They are small data spaces inside an object to store values.
- When an object is created it has a reserved space for each field defined in its class.



Properties or attributes

- Since they store **values that can vary** over time they are called (instance) **variables**.

Which is the type of the following fields?

```
private int amount;
private Student delegate;
private Server host;
```

Which are the names for the following fields?

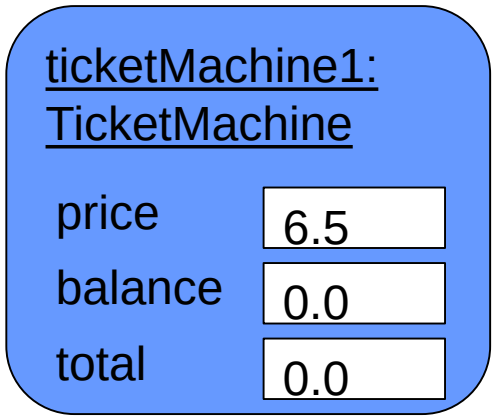
```
private boolean alive;
private Person tutor;
private Game game;
```

Constructors

- They **set up** each object so it can be used once created.
  - This operation is called **initialization**.
- They have the same name of the class where they are defined in, and they do not have a *return value*.

```
public TicketMachine (double ticketPrice){
 setPrice(ticketPrice);
 setBalance(0.0);
 setTotal(0.0);
}
```

Constructors



6.5 is the value for the ticket price
0.0 is the value for balance
0.0 is the value for the total

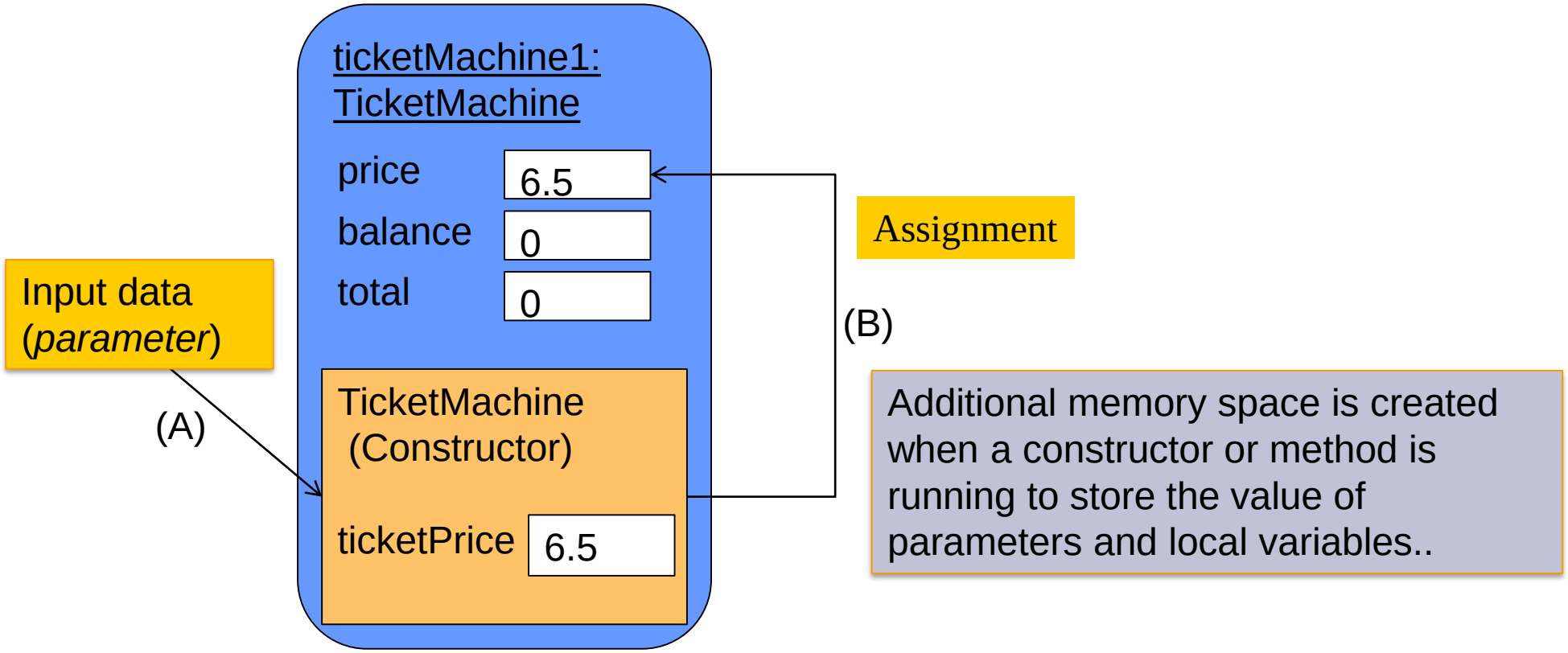
You should explicitly write the initialization code. This is a way of self-documenting your code showing that your objects are initialized and that you haven't forgotten to give them a value.

- In Java, if you do not explicitly initialize the properties, they receive a default value automatically.

`int`	`0`
`double`	`0.0`
`boolean`	`false`
`String`	`null`

Passing data via parameters (I)

- Both constructors and methods receive data via their parameters.



Passing data via parameters (II)

- You have to distinguish between the **name of the parameters inside** a method or constructor and the **values passed when** that method or constructor is **called.**
  - Names ⟹ formal parameters
  - Values ⟹ actual parameters

```
public TicketMachine (double ticketPrice)
```

```
…
t= new TicketMachine(6.5);
…
```

Formal parameter

Actual parameter

Objects as parameters

- **Objects** can be used as **parameters** for other objects' **methods**.
- If a method requires a parameter to be an object (actually a reference to an object), the **name of the class** of the expected object is used as the **parameter type** in the **method's signature**.

```
public void punch(Square square)
```

```
public void requestGroupChange(Form request)
```

Variable scope

- A **variable scope** defines the source code section in which that variable can be used.
  - A **formal parameter** is **only available inside** the method or constructor **where** it has been **declared**.
  - The **scope** for an **attribute** is the **whole class**, i.e., it can be accessed from everywhere inside the class.

Variable lifetime

- A variable's **lifetime** describes for how long it will exist before being removed from memory.
  - The **lifetime for a parameter** is limited to the **running time** of the method or constructor where it's been declared.
    - Once the method or constructor has finished, the **formal parameters disappear, the space they used is released** and their **values are lost**.
- The **lifetime for a property** is the **same** as that of the **object** to which belongs.

Exercise

- Which is the class for the following constructor?

```
public Dog (String name)
```

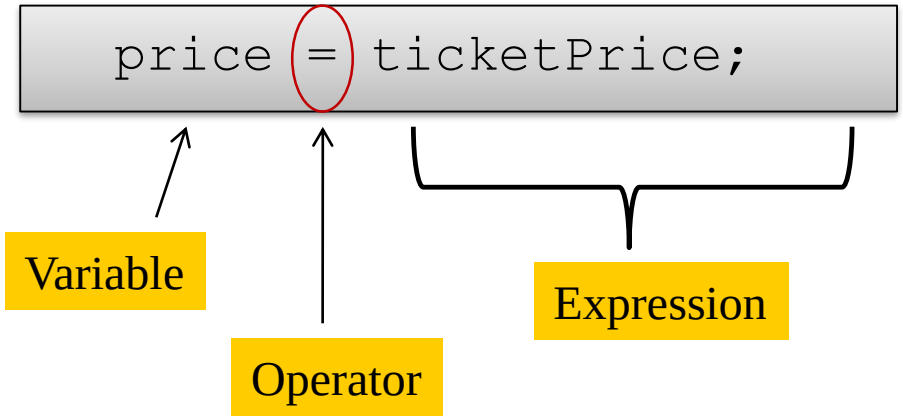
- How many parameters does this constructor have? Which are their types?

```
public Book (String title, double price)
```

- Which attributes (both names and types) could the `Book` class have?

Assignment

- The **assignment sentence copies** the **value** at the **right** of the = sign into the **variable** in the **left** side.



Rule The type of the assigned expression must be *compatible* with the variable to which it's assigned.

The same applies to the relationship between **formal** and **actual parameters**.

Primitive types vs object types

- In Java, **primitive types** are those that are not objects

Integer numbers	`byte` (8 bits) [-128 to 127] `short` (16 bits) [-32768 to 32767] `int` (32 bits) [-2147483648 to 2147483647] `long` (64 bits) [-9223372036854775808 to 9223372036854775807]
Floating-point numbers	`float` +1.40e-45f +3.4028235e38f `double` +4.9e-324 +1.79769313486231570e308
Other types	`char` (16 bits Unicode) `boolean (true or false)`

- **Object types** are those defined using classes. Some of them are defined by the system (i.e., *String*)

Using primitive data types

- Given the following attribute declaration

```
private byte age;
private short shortNumber;
private float floatNumber;
private double doubleNumber;
```

Valid assignments	Invalid assignments
`age = 127;`	`age = 128;`
`shortNumber = -32768;`	`shortNumber = -32769;`
`floatNumber = 23.345f;`	`floatNumber = 23.345;`
`doubleNumber = 23.345;`	`doubleNumber = 23.345u;`
`doubleNumber = 23.345d;`	
`doubleNumber = 23.345f;`	

Primitive types vs object types

Primitive type

```
int x;
```

x

*The stored value is **an integer.***
For instance:

12

Object type

```
Dog p;
```

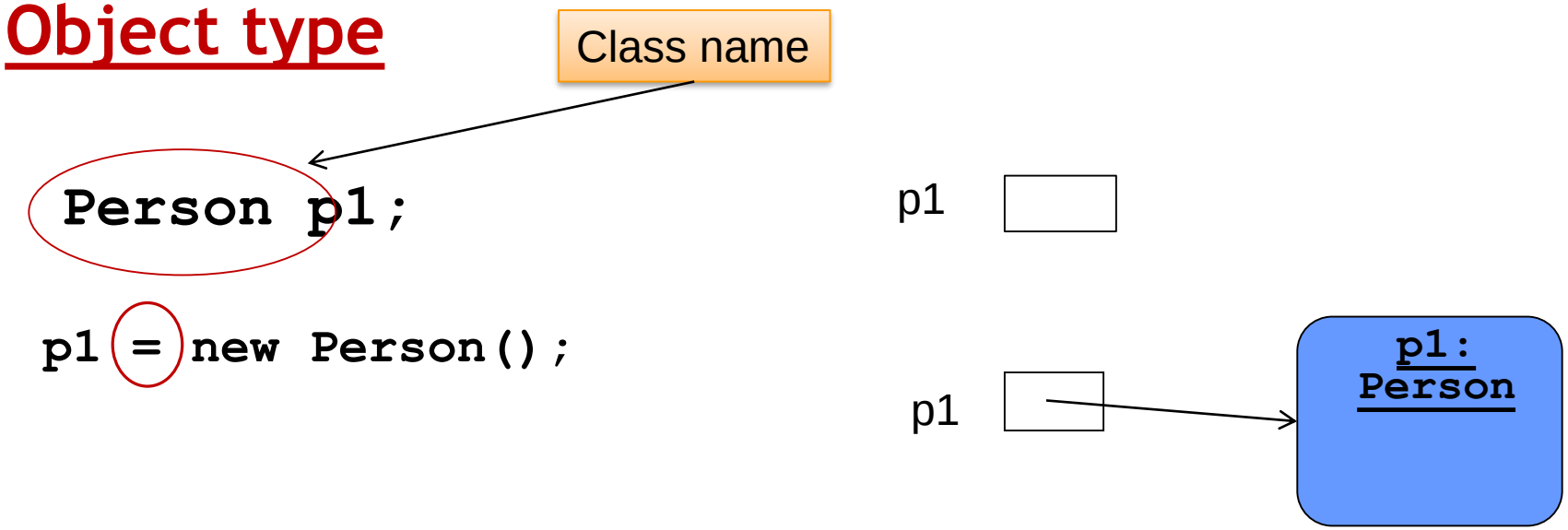
p

The stored value is
a reference.

Name for a memory zone

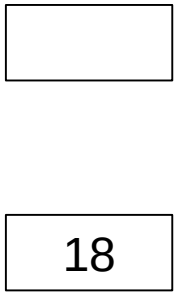
Primitive types vs object types

Object type



Primitive type

```
int age;
age = 18;
```



Primitive types vs object types

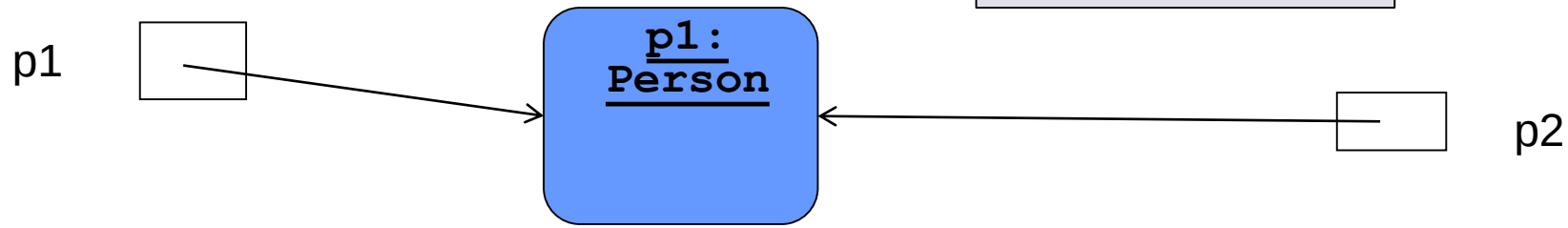
```
Person p1;
```

```
Person p2;
```

p2

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

```
p2 = p1;
```




```
int a = 32;
```

```
int b;
```

b

a 32

```
b = a;
```

b 32

Methods (I)

- Methods have two different parts
  - Header

```
// This method returns the ticket price
 public double getPrice()
```

  - Body enclosed between curly brackets { }

```
{
   return price;
}
```

    - It contains **declarations** and **sentences**.
    - A set of declarations and sentences between curly brackets is a **block**.

Methods (II)

- There are important differences between the signature of constructors and methods.

```
public TicketMachine(double ticketPrice)

public double getPrice()
```

Which differences do you notice?

Rule Constructors do not have a return type.

Methods (III)

- This sentence in the previous example:

```
return price;
```

- is a **return sentence**. It is responsible for **returning a value** (a double in this case) **compatible** with the **return type of the method** (also double this time).

```
// This method returns the ticket
// price

 public double getPrice() {
        return price;
 }
```

The return sentence is always the last *executed* one . No more sentences are executed after that.

Accessor methods

- They provide information on the object's status

```
// This method returns the ticket price

  public double getPrice(){
        return price;
  }
```

Convention

Every method returning the value of an attribute must start with the **get** prefix

Those methods returning the value of a boolean attribute must start with the **is** prefix

Mutator methods

- They change the object's status

```
// Sets a new balance

    public void setBalance(double amount){
          balance = balance + amount;
    }
```

- Compound assignment operator `balance += amount ;`

Convention

Every method that changes the value of an attribute should start with the **set** prefix.

Printing from methods

- Given this method

The + sign is the string *concatenation* operator. It is used to produce a single string.

```
// Print a ticket and reduce balance to zero
 public void printTicket()  {
    System.out.println (" Ticket");
    System.out.println (" Price:" + price);
    System.out.println();
    balance = 0.0;
 }
```

- The method `System.out.println` prints the parameter it receives to the screen (text terminal).

The conditional statement

- This version of the method does not check its parameter

```
public void insertMoney(int amount) {
     balance = balance + amount;
}
```

- Now, we check that the amount makes sense

```
public void insertMoney(int amount){
   if (amount > 0){
    balance = balance + amount;
   }
}
```

Comparison operator

The conditional statement

- Also called **if sentence**
- It provides a way to perform one of two possible actions depending on the resulting value of a given test.

```
if (test returning a true or false result) {
  The test returned true, perform this action.
}
else {
   The test returned false, perform this other action.
}
```

- A **boolean expression** or **condition**, i.e., something with only two possible values (true or false).

Some examples

if sentence

```
if (condition) {
    statements
}
```

if-else sentence

```
if (condition) {
    statements
} else {
    statements
}
```

if else-if else sentence

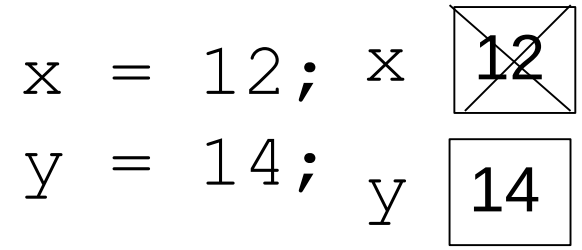
```
if (condition) {
    statements
} else if (condition) {
    statements
} else {
    statements
}
```

```
if (score >= 90) {
    grade = "A";
} else if (score >= 80) {
    grade = "B";
} else if (score >= 70) {
    grade = "C";
} else {
    grade = "F";
}
```

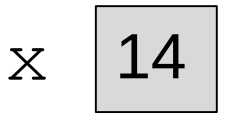
Primitive types

Assignment

```
x = 12;
y = 14;
```



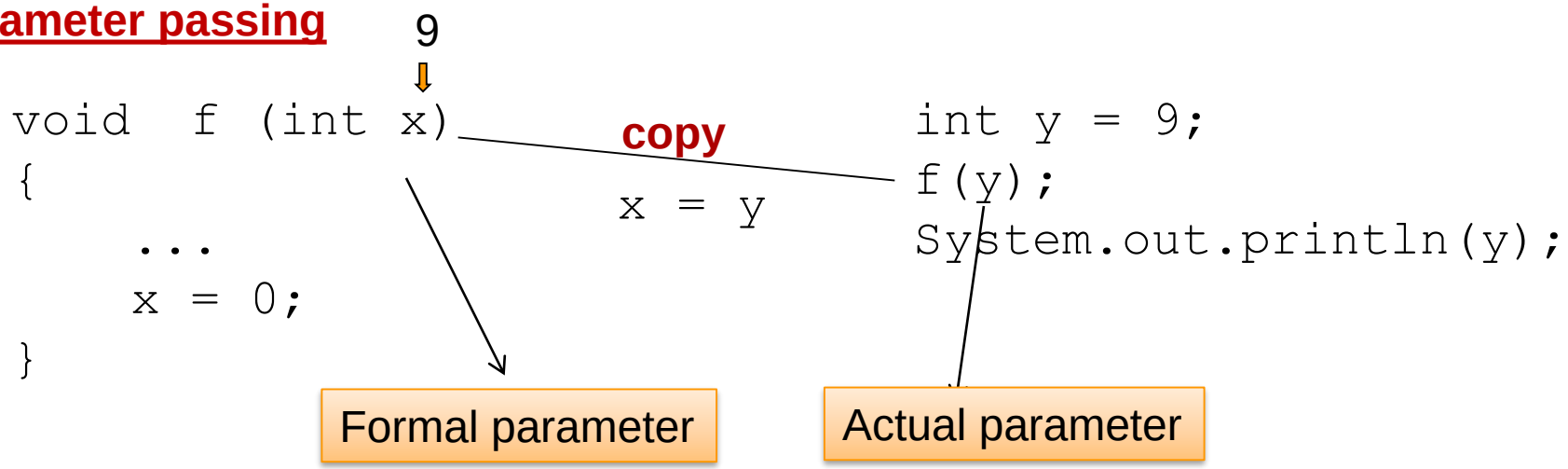

```
x = y;
```



Comparison

```
if (x == y)
   ...
```

Parameter passing



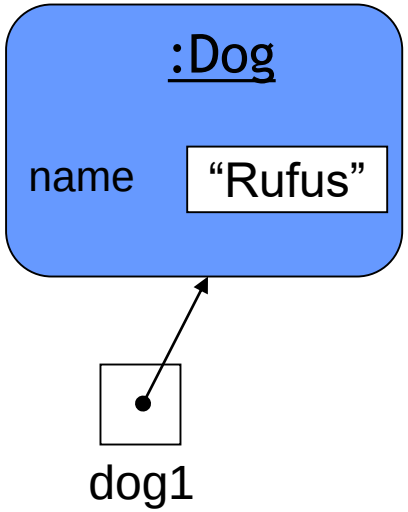
Object types

```
Dog dog1;
Dog dog2;
```

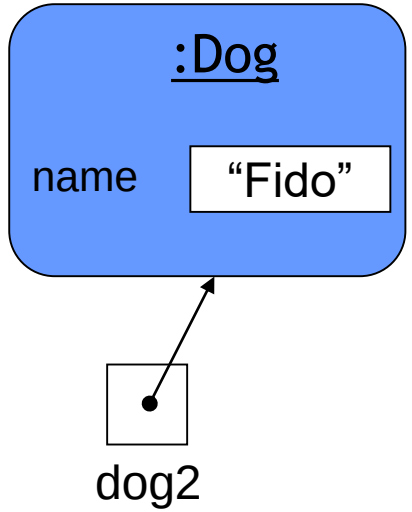
dog1

dog2

dog1 = new Dog();

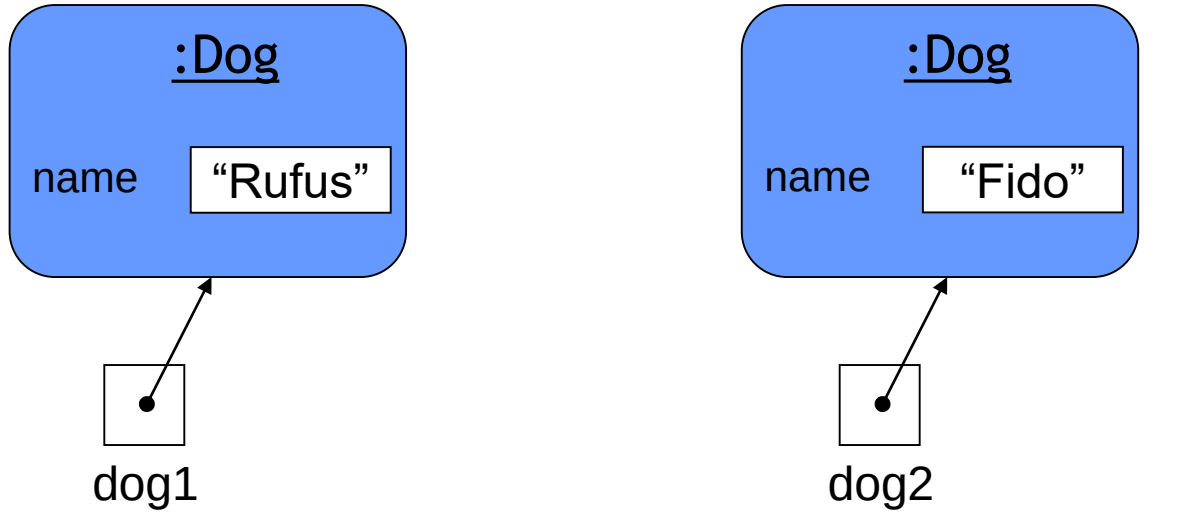


dog2 = new Dog();

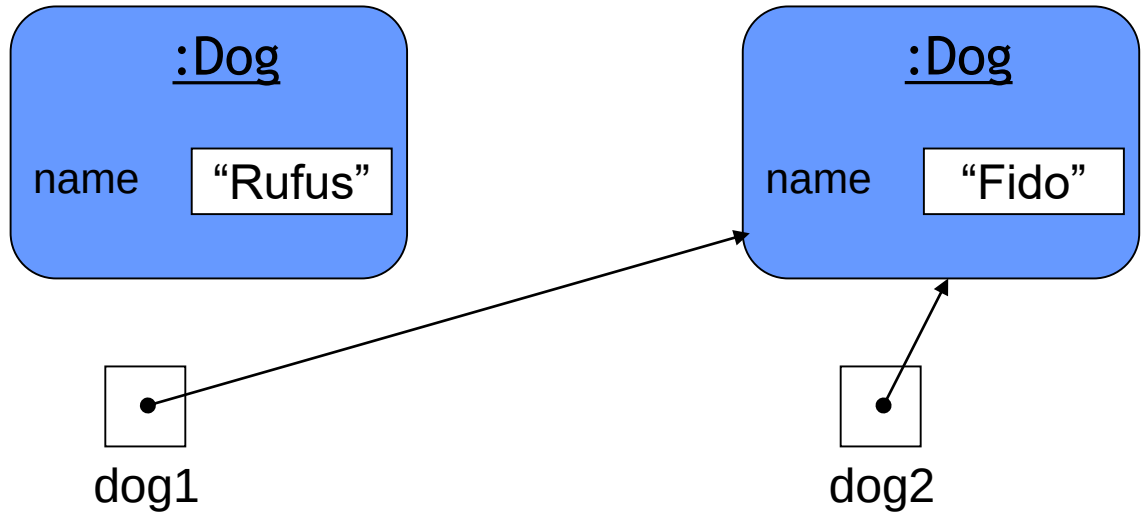


Object types

Assignment



dog1 = dog2

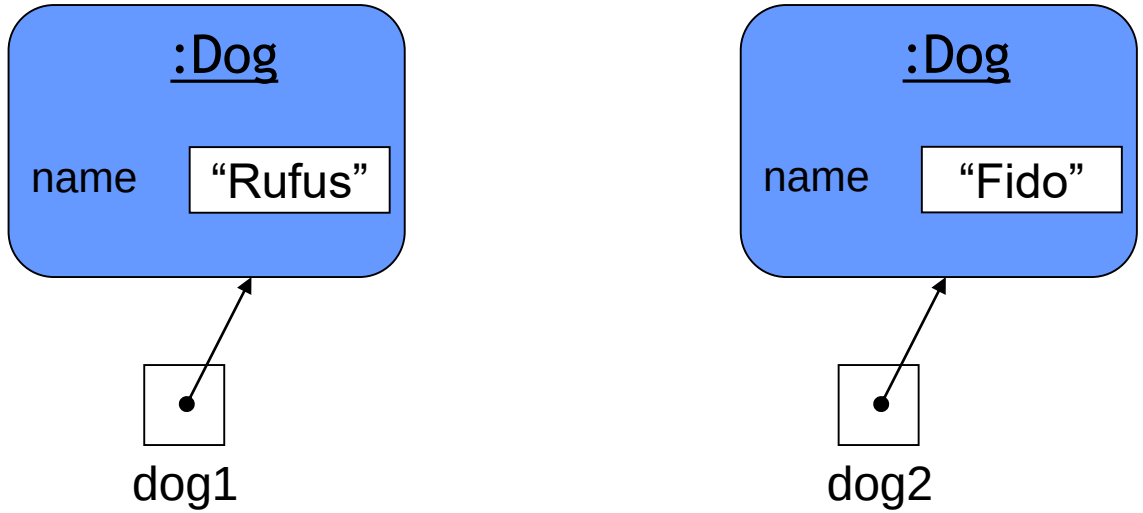


Object types

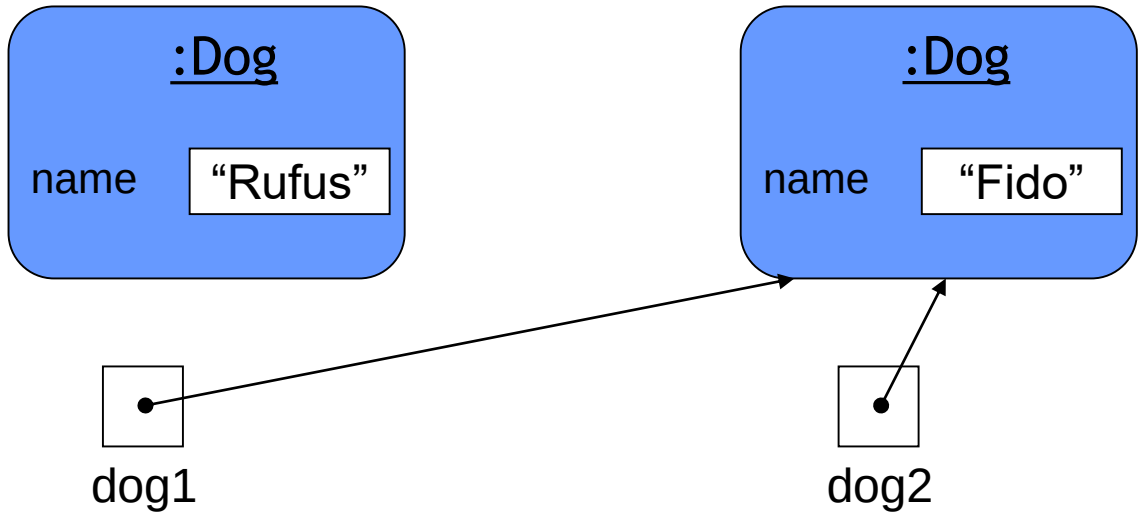
Comparison

```
if (dog1 == dog2)
```

false



true



Object types

Parameter passing

```
public void feed (Dog p){
     ...
     p = null;
}
```

copy

p = tobi

```
public void dogsHome()
{
   Dog tobi = new Dog();
   feed(tobi);
   ...
}
```

Local variables

- A local variable is a variable declared and used inside a method.
  - Its **scope** is limited to the code **inside the method**.
  - Its **lifetime** is limited to the **method's running time**.

```
/* compute the difference between the balance
*  and the ticket price
*/
     public int refundBalance() {
        double difference;
        difference = balance - price;
        balance = 0.0;
        return difference;
     }
```

Summary: properties, parameters and local variables (I)

- There are three kinds of variables.
- **Properties or attributes**
  - They are defined **outside methods** and **constructors**.
  - They are used to store data needed during the whole life of the object. Their lifetime expires when the object is destroyed.
  - The **scope** of fields is the whole **class**. That is, they can be used from any method or constructor of the class.
  - They **cannot be accessed from outside** the class if they are defined as **private**.

Summary: properties, parameters and local variables (II)

- **Parameters**
  - **Formal parameters** do exist during a constructor or method **running time**. Their values are **lost between calls**.
  - **Formal parameters** are **defined** in a constructor or method's **signature**. They are **initialized** with the **values** of the **actual parameters** used during the call.
  - Formal parameters **scope** is limited to the **method or constructor** where they are **defined**.

Summary: properties, parameters and local variables (III)

- **Local variables**
  - They are **declared within the body** of a method or constructor.
  - They are **used inside the body**. They **must be initialized** before being used, they do not have any default value.
  - They exist during the **running time of the method or constructor**. Their values are **lost between calls**.
  - Their **scope** is limited to the **block** where they are declared. They cannot be accessed from outside that block.

Logical operators

- They work on boolean values (true or false) and produce as a result a new boolean value.

Logical operators	&& (and) || (or) ! (not)

a	b	a && b	a || b	!a	!b
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	T	F	F	T	T

  - The expression **a && b** is true if both **a** and **b** are true and false otherwise.
  - The expression **a || b** is false is both **a** and **b** are false and true otherwise.
  - The expression **!a** is true when **a** is false and vice versa.

Relational operators

Operator	Name	Example	meaning
<	less than	*a<b*	*a* is less than *b*
>	Greater than	*a>b*	*a* is greather than *b*
==	Equal to	*a==b*	*a* is equal to *b*
!=	Not equal to	*a!=b*	*a* is not equal to *b*
<=	Less than or equal to	*a<=b*	*a* is less than or equal to *b*
>=	Greater than or equal to	*a>=b*	*a* is greater than or equal to *b*

Exercise

- What does this method do?

```
public void setValue(int newValue){
    if ((newValue >= 0) && (newValue < 60))
        value = newValue;
}
```

- What does it happen if you use these conditions instead of the original one?

```
if ((newValue > 0) && (newValue < 60))
if ((newValue > 0) || (newValue < 60))
```

- Which of the following expressions are true?

```
! (4 < 5)
! false
(2 > 2) || ((4 ==4) && (1 < 0))
```

```
(2 > 2) || (4 ==4) && (1 < 0)
(34 != 33) &&  ! false
(4 <= 8) && (8 > 5) || (3 < 2)
```

String concatenation

- The addition operator (+) has different meanings depending on the type of the operands.

42 + 12	54
“Java” + “with BlueJ”	“Javawith BlueJ”
“answer: “ + 27	“answer: 27”
return “0” + value	“08” if value contains an 8
return “” + value	“8” if value contains an 8

Division and modulo operators

- The modulo operator (**%**) computes the remainder for an integer division.
- The slash operator (**/**) computes the quotient for an integer division.

27 / 4	6
27 % 4	3
8 % 3	2

```
public void increment(){
  value = (value + 1) % 60;
}
```

What does this method do?

Replace this using an if sentence.

Operators (main ones) precedence

- Listed below, from highest to lowest precedence

Operators	Precedence	If several appear …
postfix	*expr++ expr--*	
unary	*++expr --expr +expr -expr* !	
multiplicative	* / %	Left to right
additive	+ -	Left to right
relational	< > <= >=	
equality	== !=	
logical AND	&&	Left to right
logical OR	||	Left to right
ternary	? :	
assignment	= += -= *= /= %=	Right to left

The **`this`** keyword

- Sometimes the same name is used to refer both a parameter and an attribute. We use **`this`** to disambiguate them.

```
public class Message {
    private String from;
    private String to;
    private String text;

    public Message (String from, String to, String text) {
        this.from = from;
        this.to = to;
        this.text = text;
    }
}
```

The **`this`** keyword

- `this` refers to the current object:

```
this.from = from;
```

- This sentence actually means:

```
Attribute with name "from" = parameter with name "from";
```

- If a given name perfectly describes something, we should use it both for parameters and attributes and rely on **`this`** to solve the ambiguity.

Error handling

- At the beginning, most of the errors are **syntax errors**.
  - The IDE highlights them in your code.
- Later, most common errors are **logic errors**.
  - IDE does not help to find them.
  - They are the well-known “bugs”.
- Some logic errors are not immediately obvious.
  - Commercial software sometimes (many times) has bugs.

“Hand made” unit testing with BlueJ

- You can create objects for each class.
- You can call each individual method.
- You can use the **Inspect** function to check the status of the object.

Making good tests is a creative process,…
However, testing is time-consuming and repetitive.
(That’s why they are not “hand made”).

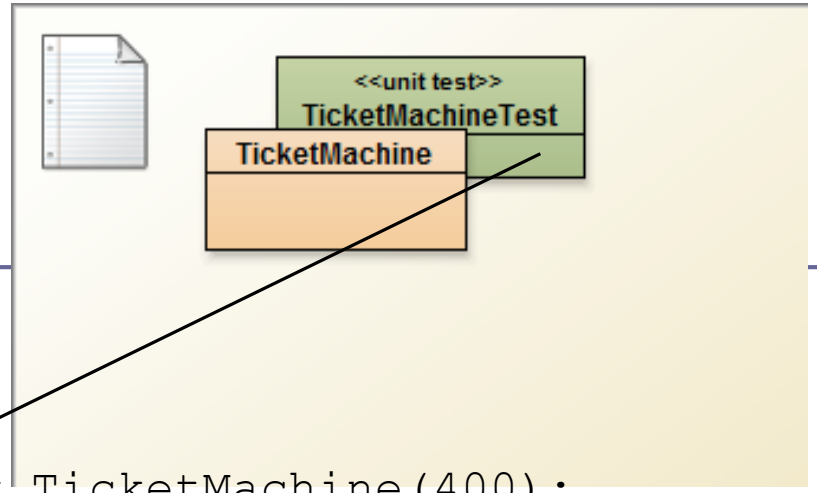
Unit testing

- Special **classes** are **developed** only to **make the tests** (they contain test methods).
- Those classes are known as **unit tests** because they are used to **check/test individual classes**.
- Each **"common" class** from the project is **associated** to a **test class**.

Unit testing with JUnit

- JUnit is a testing framework for Java.
- A JUnit test class **contains**:
  - Source code to run the tests on a given class.
  - Source code to check the tests were OK by means of assertions.
- **An assertion** is an expression establishing a condition which is assumed to be true. If it is false, it means the assertion failed and, thus, there is a bug in the program.

Testing with JUnit




```
@Test
public void insertMoney(){
    TicketMachine ticketMa1 = new TicketMachine(400);
    ticketMa1.insertMoney(900);
    assertEquals(900, ticketMa1.getBalance(), 0.1);
}

@Test
public void printTicketRightBalance() {
    TicketMachine ticketMa1 = new TicketMachine(600);
    ticketMa1.insertMoney(600);
    ticketMa1.printTicket();
    assertEquals(0, ticketMa1.getBalance(), 0.1);
}
```