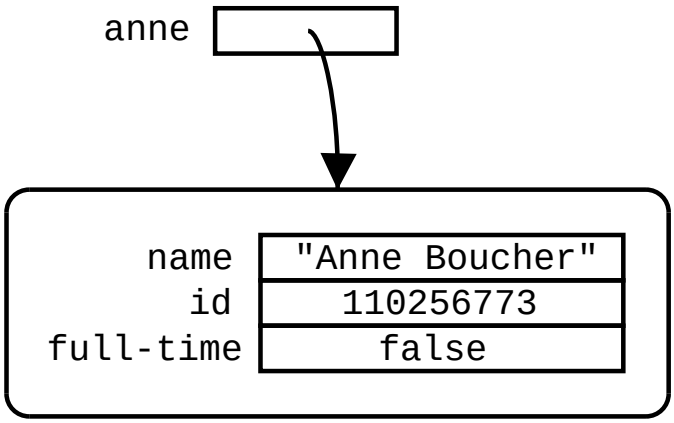
Objects and Classes

- Programs manipulate data
- Variables store data
- A variable holds either:
  - a value from a primitive data type (int, boolean, char, ...)
  - or a reference to an object
- An *object* is a composite piece of data: it is a group of variables treated as a unit



Objects and classes

- Objects:
  - An object is a composite piece of data which can react to messages sent to it.
  - The data type of an object is a class

- Classes:
  - A class is a data type
  - A class is like the "blueprint" of a set of objects
  - Classes have have *attributes* and *methods* (to describe the structure and behaviour if its objects.)
    * Attributes: variables describing the characteristics of objects in the class.
    * Methods: operations on objects of the class; how to react to messages from other objects.

Class definition: syntax

```
public class Name
{
    // Attribute definitions
    // ...

    // Method definitions
    // ...
}
```

Example: Stereo

```
public class Stereo {
    // Attributes
    float volume;
    boolean radio_on;
    boolean cd_in;
    int current_song;

    // Methods
    void play_cd()
    {
        radio_on = false;
        if (cd_in) {
            current_song = 1;
        }
        // ...
    }
    void set_volume(float v)
    {
        volume = v;
    }
    // ...
}
```

Class definition structure

- Attribute definitions

  type variable;

where *type* is either a primitive data type (`int`, `boolean`, etc.) or the name of a user-defined class.

Class definition structure (contd.)

- Method definitions

```
type method_name(list_of_parameters)
{
    statements;
}
```

where *type* is either `void` (the method doesn't return anything,) a primitive data type or a user-defined data type. The *list_of_parameters* is of the form

```
type1 arg1, type2 arg2, ..., typen argn
```

Objects and classes

- In analysis we should:
  - Discover the classes of objects involved (physical or abstract,) and
  - Identify the attributes of those classes: the "has-a" relationship.

- Example: modelling a bank account

```
public class BankAccount
{
  // Attribute definitions

  // Method definitions
}
```

- The "has-a" relationship:
  - A bank account *has a* number
  - A bank account *has a* balance
  - A bank account *has an* owner

Example

```
public class BankAccount {
    // Attributes
    long number;
    float balance;
    String owner;
}
```

Example

```
public class BankAccount {
    // Attributes
    long number;
    float balance;
    String owner;

    // Methods
    void deposit(float amount)
    {
        // ...
    }
    void withdraw(float amount)
    {
        // ...
    }
    float getBalance()
    {
        // ...
    }
}
```

Example

```
public class BankAccount {
    // Attributes
    long number;
    float balance;
    String owner;

    // Methods
    void deposit(float amount)
    {
        balance = balance + amount;
    }
    void withdraw(float amount)
    {
        // ...
    }
    float getBalance()
    {
        // ...
    }
}
```

Example

```
public class BankAccount {
    // Attributes
    long number;
    float balance;
    String owner;

    // Methods
    void deposit(float amount)
    {
        balance = balance + amount;
    }
    void withdraw(float amount)
    {
        if (amount < balance)
           balance = balance - amount;
    }
    float getBalance()
    {
        // ...
    }
}
```

Example

```
public class BankAccount {
    // Attributes
    long number;
    float balance;
    String owner;

    // Methods
    void deposit(float amount)
    {
        balance = balance + amount;
    }
    void withdraw(float amount)
    {
        if (amount < balance)
           balance = balance - amount;
    }
    float getBalance()
    {
        return balance;
    }
}
```

Objects are not classes

- A class is a data type. An object is a particular value whose type is some class.
- An *object* is an *instance* of a class.
- An object has its own separate identity and its own separate state.
- The *state* of an object is the values currently assigned to its attributes.
- Each object is stored in different memory locations.

Dealing with objects

- To be able to use a class and its objects we must be able to do three things:
  - Create instances of a class (i.e. new objects)
  - Access attributes of a given object (previously created)
  - Ask or tell a given object (previously created) to perform an operation (by sending a message to it, i.e. applying a method.)

Creating objects

- To create objects of a given class:

  First: Declare a variable of that type:

  ```
  class_name variable;
  ```

  Second: Assign the variable a new instance, using the new keyword:

  ```
  variable = new class_name();
  ```

- Example

```
BankAccount davesAccount;

davesAccount = new BankAccount();
```

Accessing attributes

- The attributes of an object can be accessed directly using the dot operator:

  objectreference.*attribute*

  ...but only if the attribute exists in the class of the *objectreference*.

- Example:

```
davesAccount.owner = “David Hilbert”;
davesAccount.number = 1117123451;
davesAccount.balance = 500.00f;
System.out.println(davesAccount.number);
System.out.println(davesAccount.balance);

davesAccount.name = “Bertrand Russell”;//WRONG!
```

Sending messages to objects

- To interact with an object we send it a message by *calling*, or *invoking* one its methods.

- Calling a method is done by using the dot operator, and passing parameters or arguments (if any):

  objectreference.*method_name*(*arguments*)

  where the type of *objectreference* is a class which has a method called *method_name*, and *arguments* is a coma-separated list of values whose type matches those of the method's parameters.

- For example:

```
davesAccount.deposit(200.00f);

float x;
x = davesAccount.getBalance();
```

Example

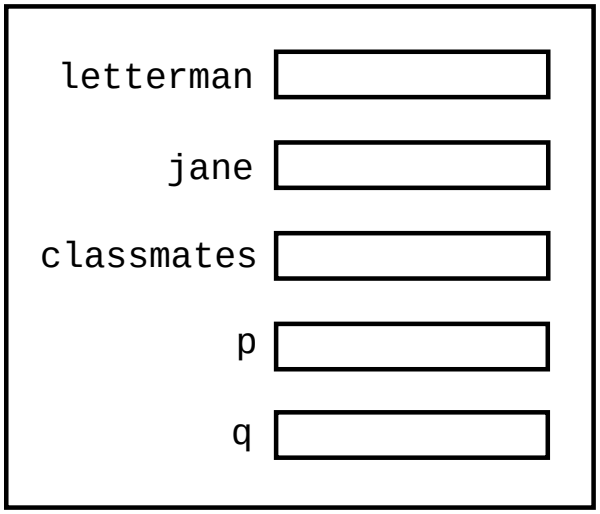
```
// in a file called Student.java
public class Student
{
    String name;
    long id;
    String program;
    String faculty;

    void set_name(String s)
    {
        name = s;
    }

    void set_id(long num)
    {
        id = num;
    }

    // Continues below ...
```

```
    String get_name()
    {
        return name;
    }

    long get_id()
    {
        return id;
    }

    void set_prog_and_faculty(String p,
                              String f)
    {
        program = p;
        faculty = f;
    }

    String get_program()
    { return program; }

    String get_faculty()
    { return faculty; }
} // Class Student ends here.
```

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

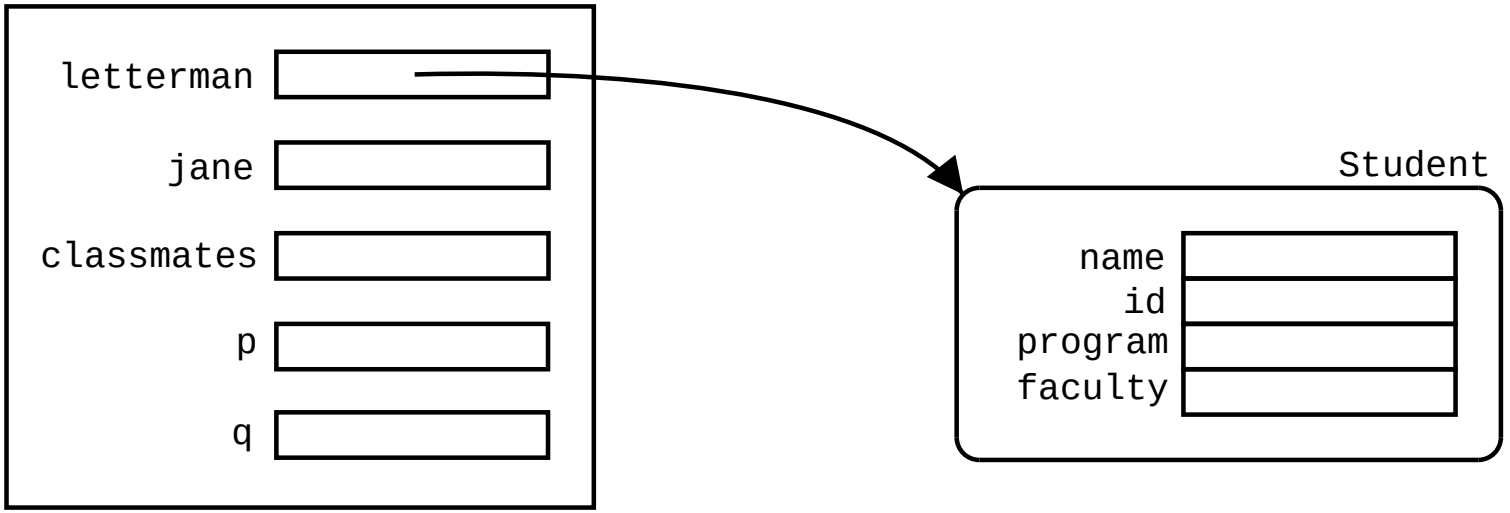
Example



Example

```
Student letterman, jane
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

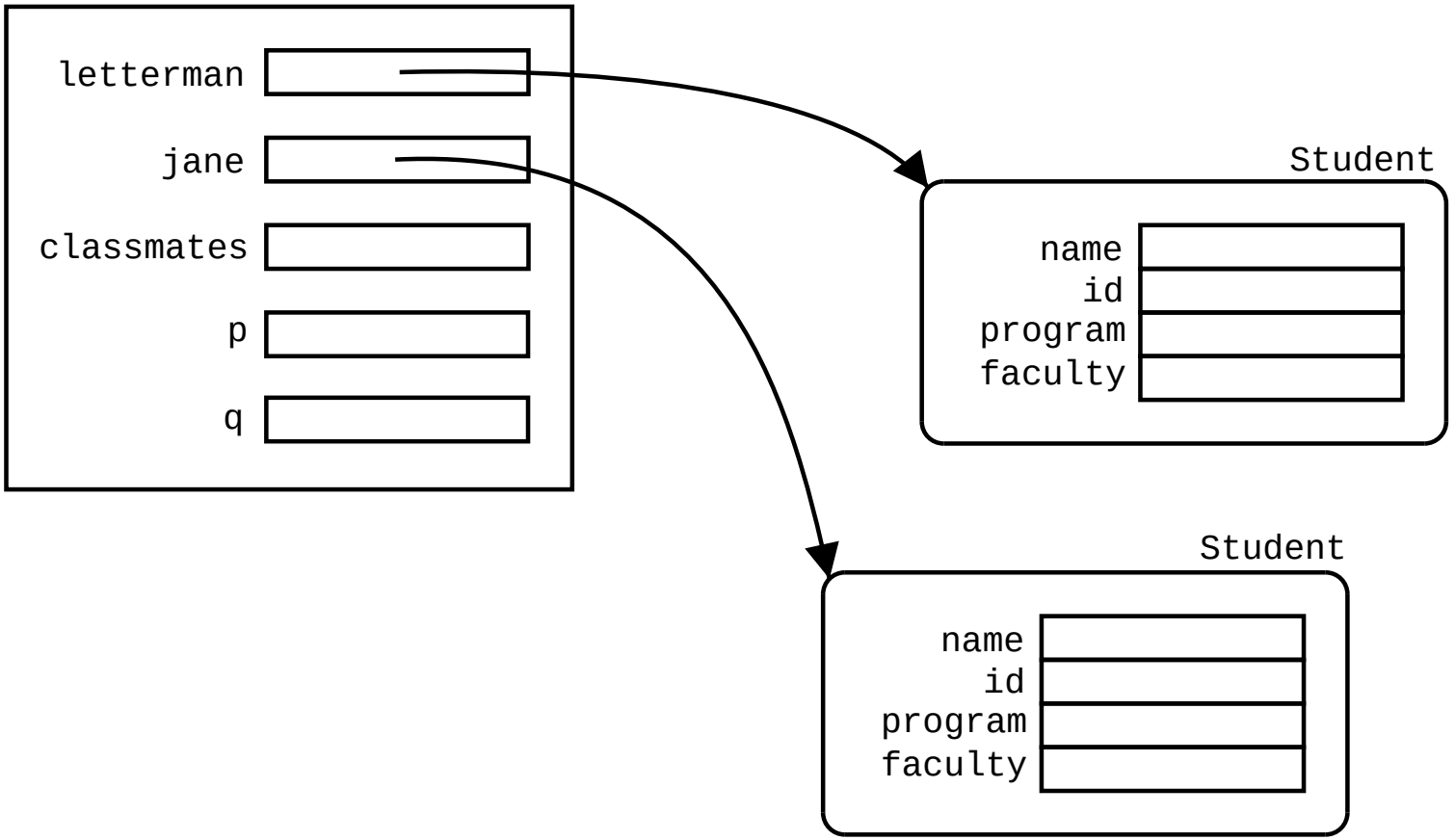
Example


letterman
jane
classmates
p
q
Student
name
id
program
faculty

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(00000001l);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

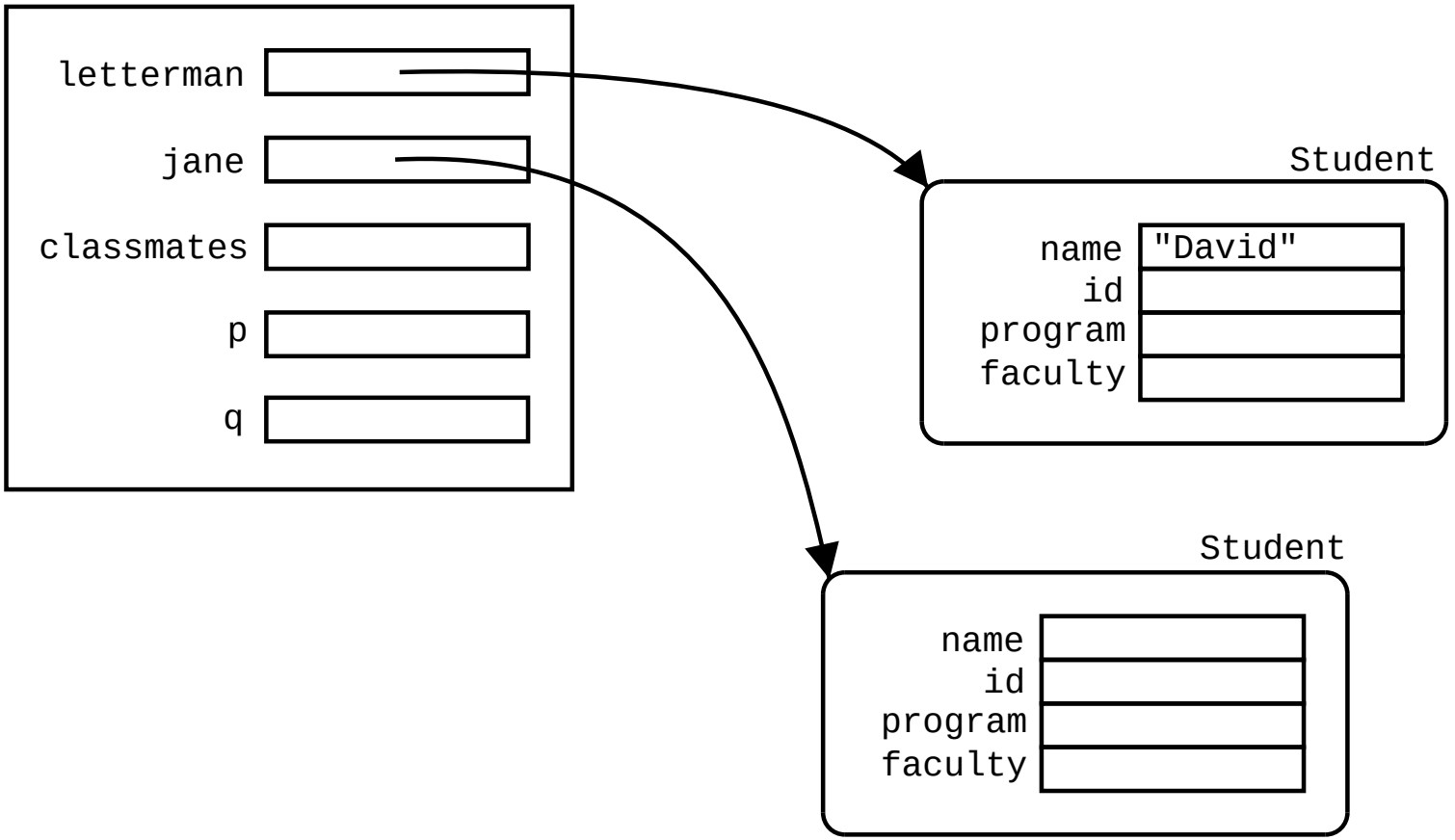
Example


letterman
jane
classmates
p
q
Student
name
id
program
faculty
Student
name
id
program
faculty

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else classm
```

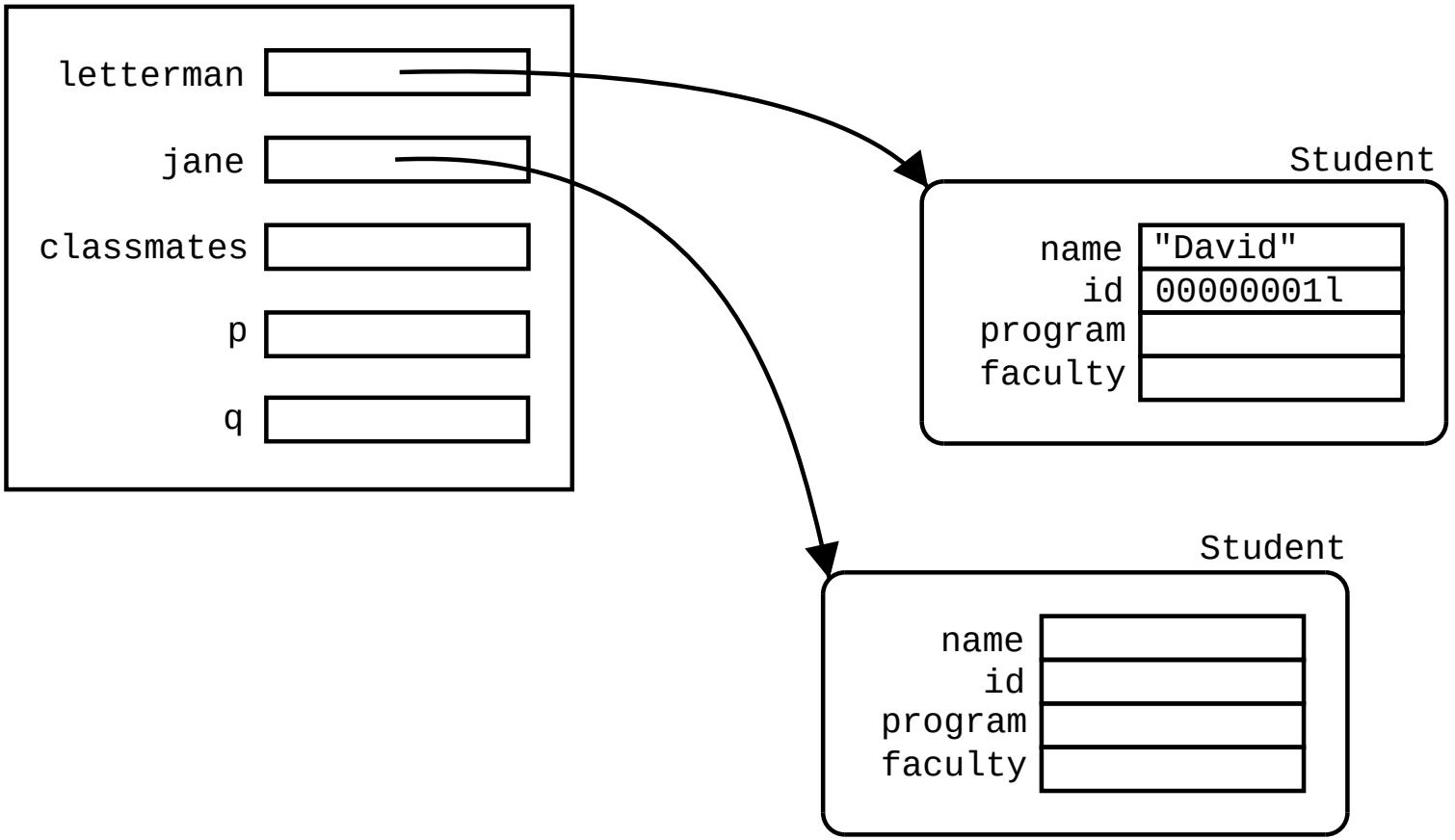
Example


letterman
jane
classmates
p
q
Student
name
"David"
id
program
faculty
Student
name
id
program
faculty

Example

```
Student letterman, jane;
String  p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

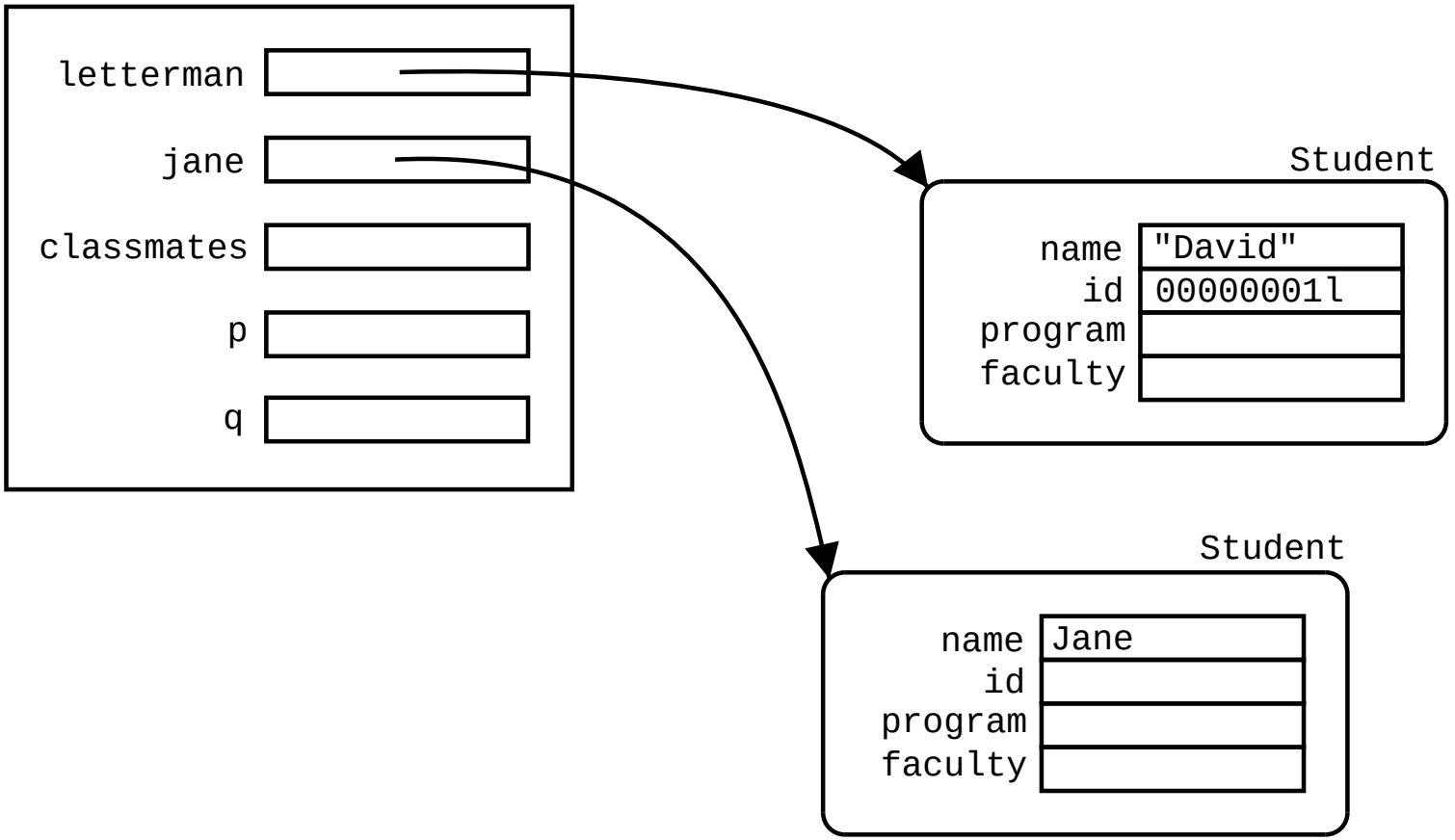
Example


letterman
jane
classmates
p
q
Student
name
id
program
faculty
"David"
00000001l
Student
name
id
program
faculty

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

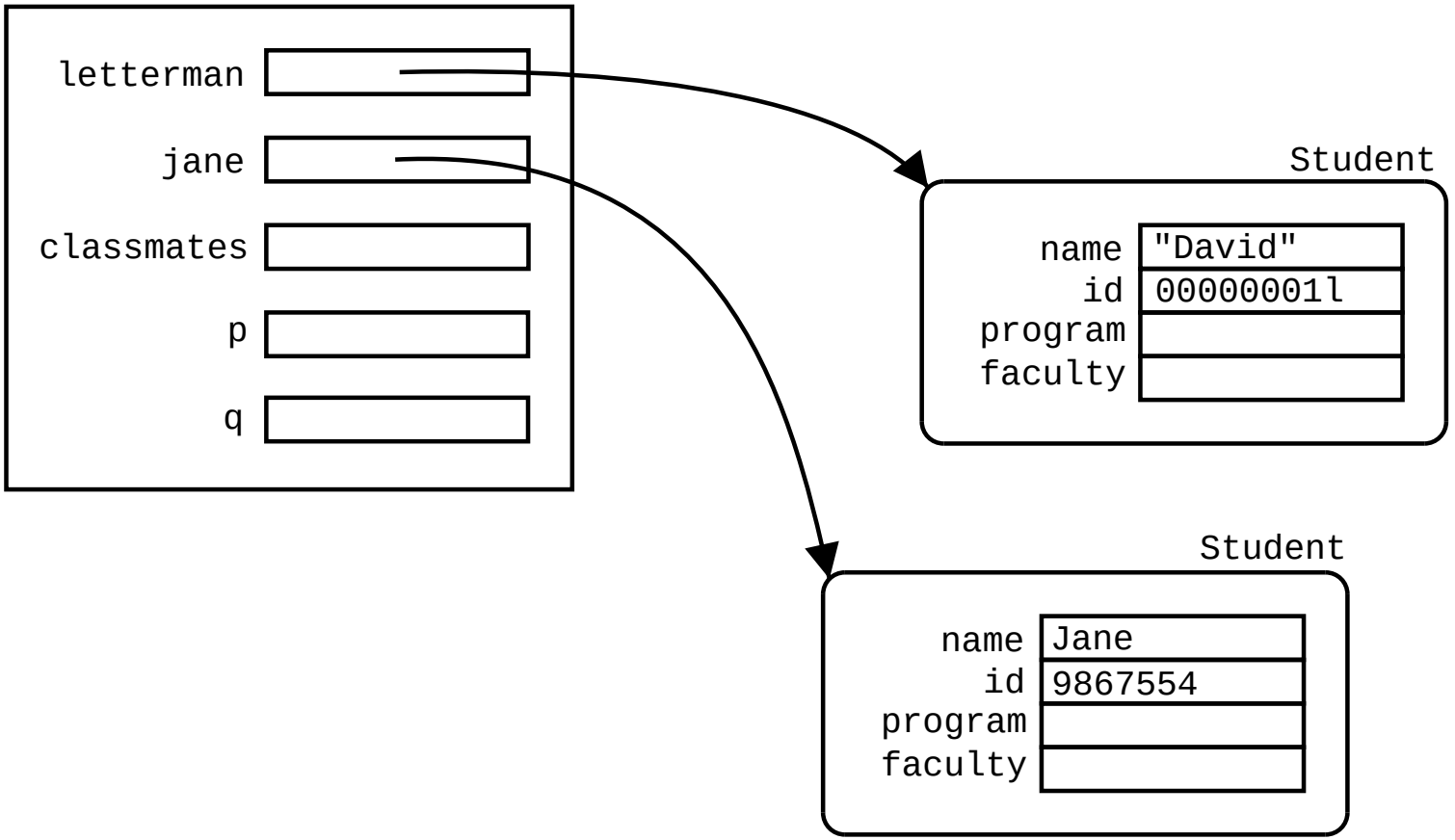
Example


letterman
jane
classmates
p
q
Student
name
"David"
id
00000001l
program
faculty
Student
name
Jane
id
program
faculty

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(00000001l);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

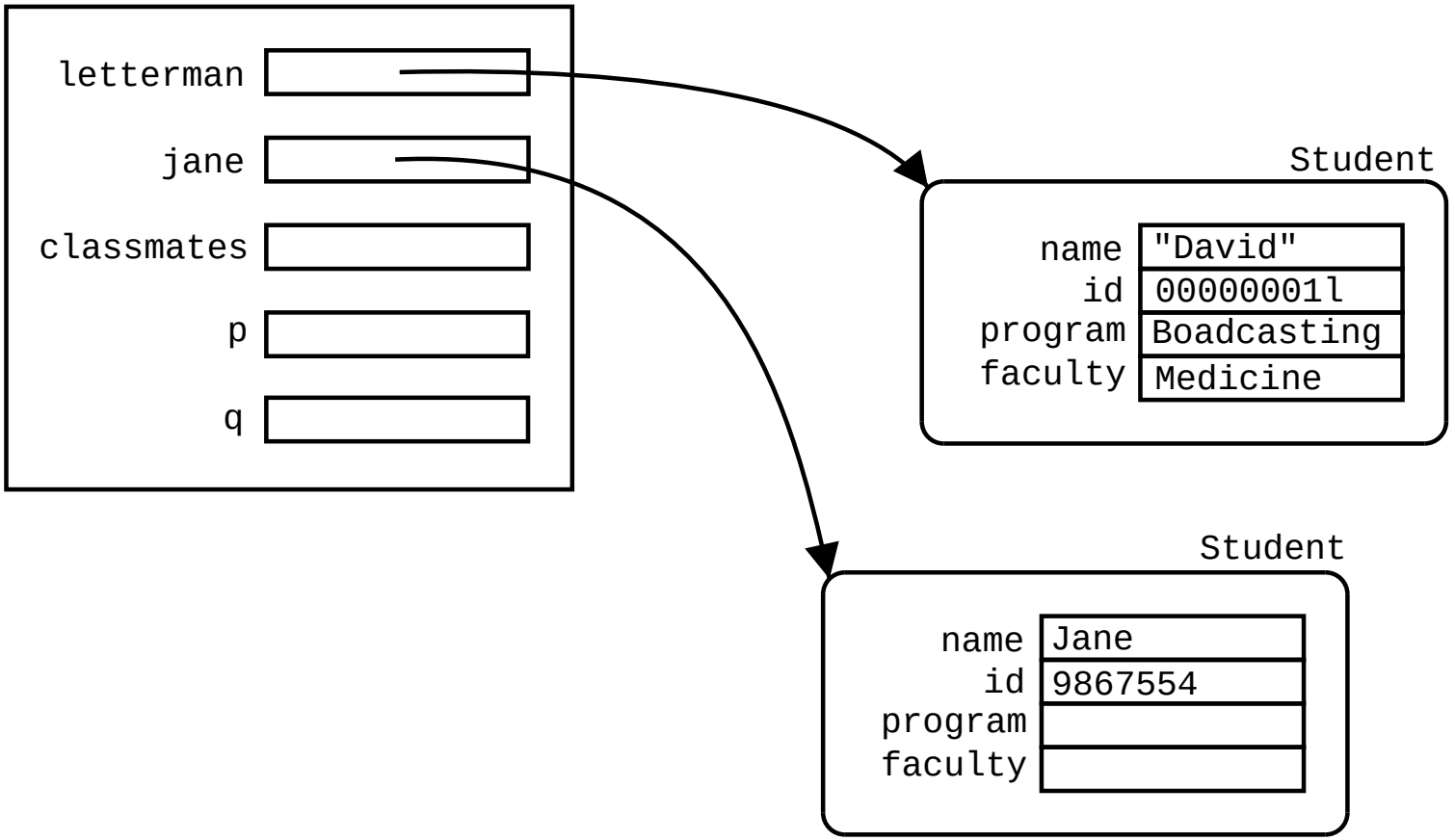
Example


letterman
jane
classmates
p
q
Student
name
"David"
id
00000001l
program
faculty
Student
name
Jane
id
9867554
program
faculty

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

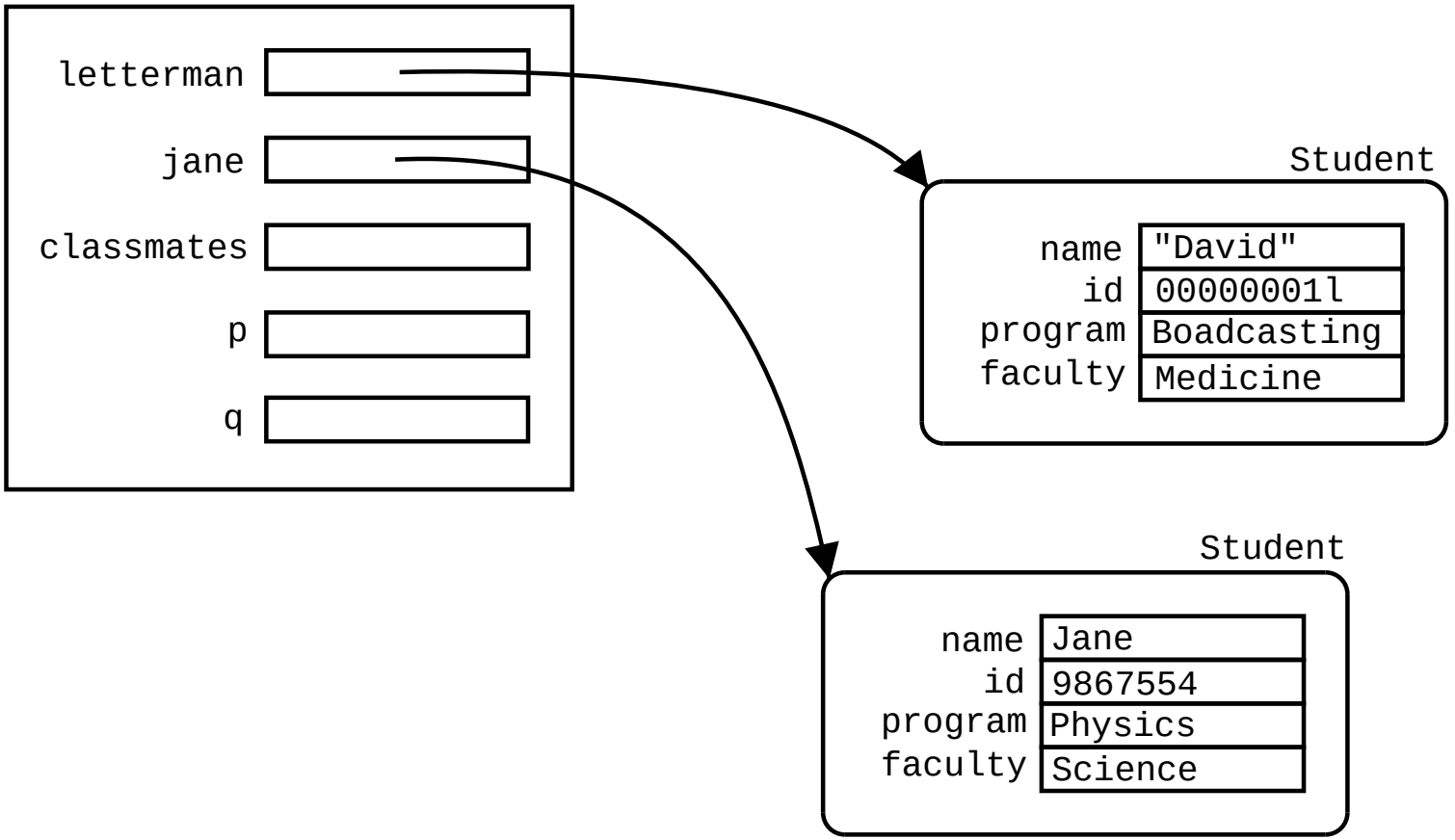
Example


letterman
jane
classmates
p
q
Student
name "David"
id 00000001l
program Boadcasting
faculty Medicine
Student
name Jane
id 9867554
program
faculty

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

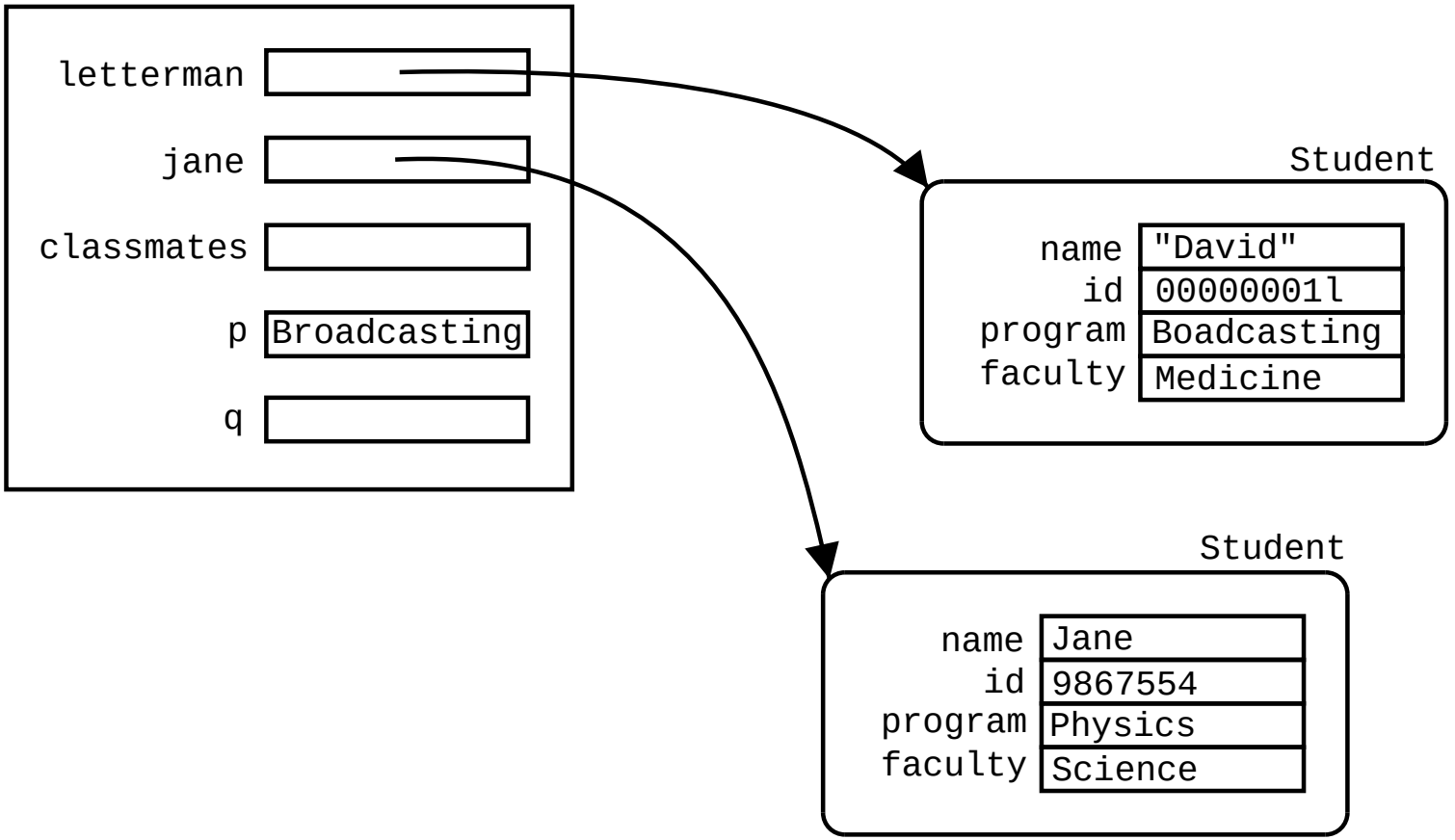
Example


letterman
jane
classmates
p
q
Student
name "David"
id 00000001l
program Boadcasting
faculty Medicine
Student
name Jane
id 9867554
program Physics
faculty Science

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(00000001l);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

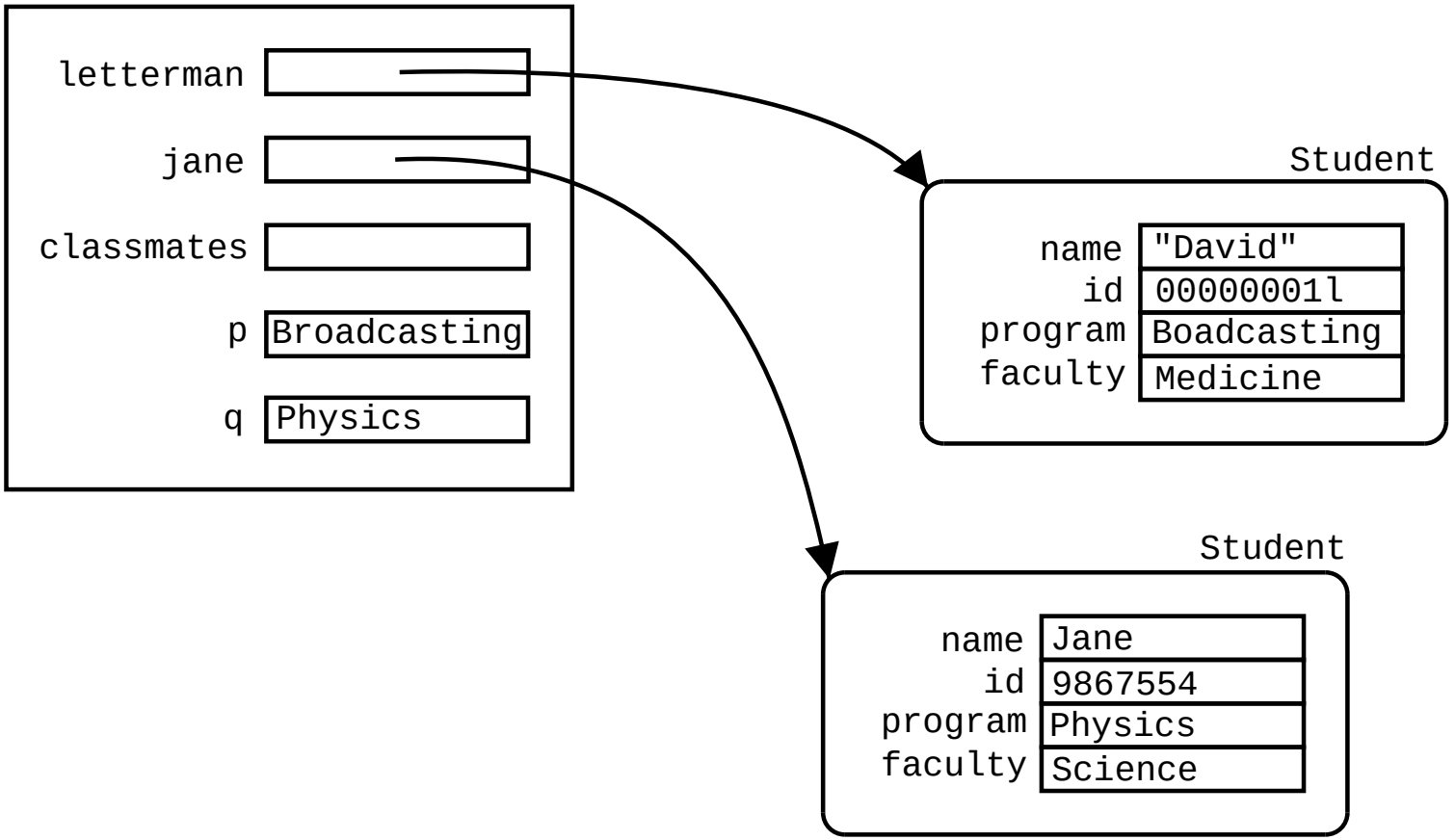
Example


letterman
jane
classmates
p
Broadcasting
q
Student
name
"David"
id
00000001l
program
Boadcasting
faculty
Medicine
Student
name
Jane
id
9867554
program
Physics
faculty
Science

Example

```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

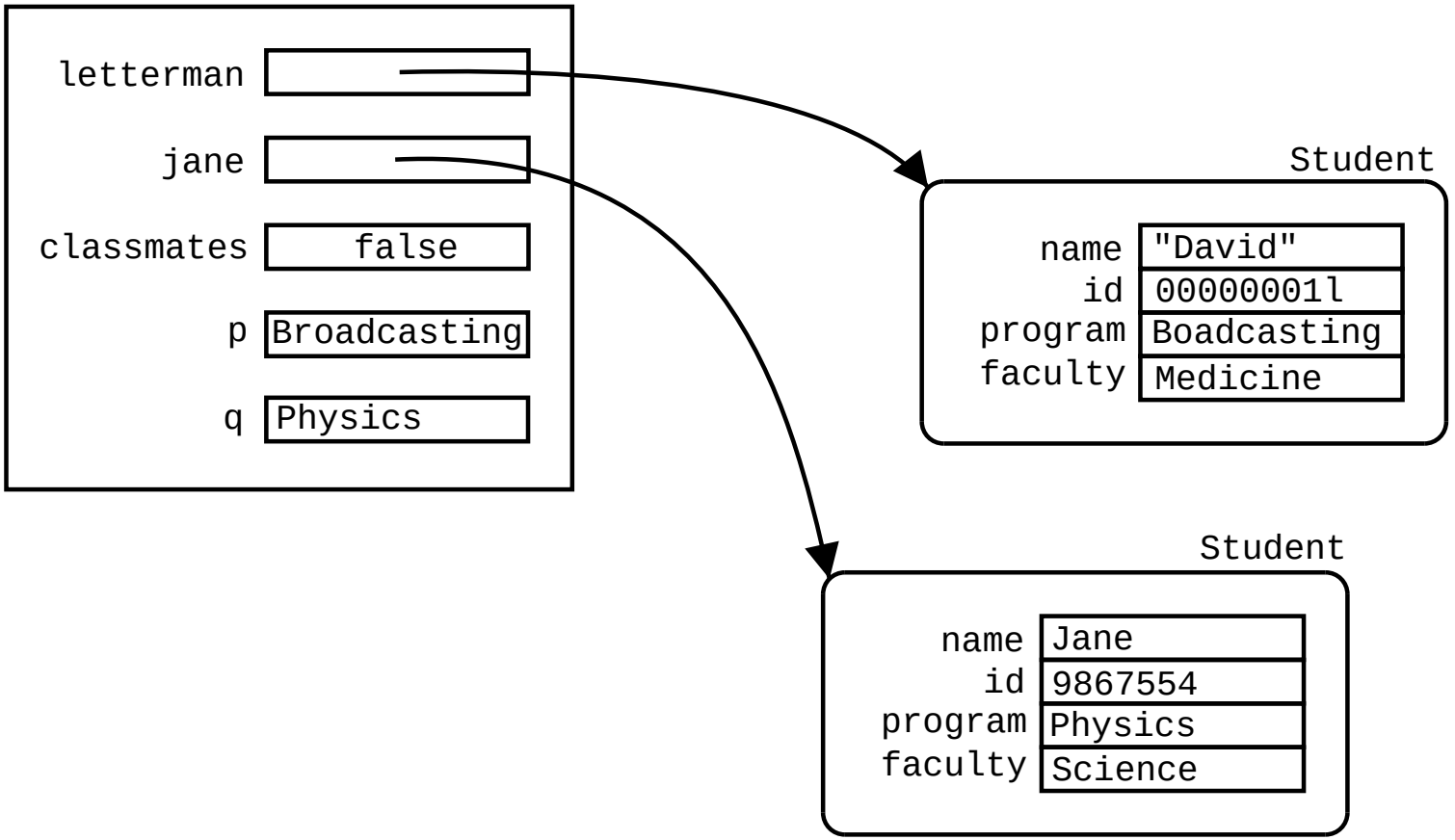
Example


letterman
jane
classmates
p Broadcasting
q Physics
Student
name "David"
id 00000001l
program Boadcasting
faculty Medicine
Student
name Jane
id 9867554
program Physics
faculty Science

Example

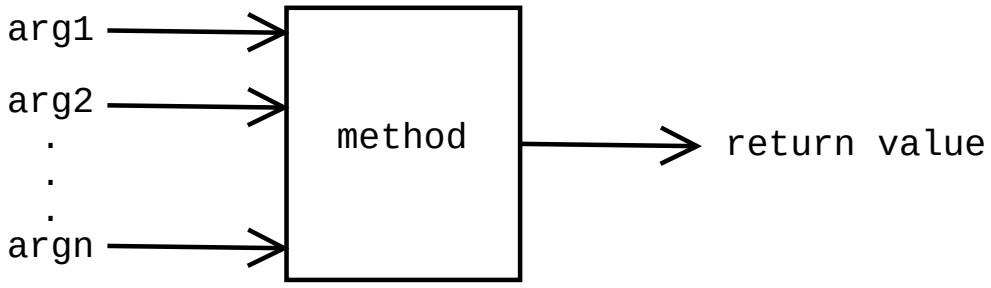
```
Student letterman, jane;
String p, q;
boolean classmates;

letterman = new Student();
jane = new Student();       // Different studen

letterman.set_name(“David”);
letterman.set_id(000000011);

jane.set_name(“Jane”);
jane.set_id(9867554);

letterman.set_prog_and_faculty(“Broadcasting”,
                                “Medicine”);
jane.set_prog_and_faculty(“Physics”, “Science”)

p = letterman.get_program();
q = jane.get_program();

if (p.equals(q)) classmates = true; else class
```

Example


letterman
jane
classmates
false
p
Broadcasting
q
Physics
Student
name
"David"
id
00000001l
program
Boadcasting
faculty
Medicine
Student
name
Jane
id
9867554
program
Physics
faculty
Science

Methods as functions

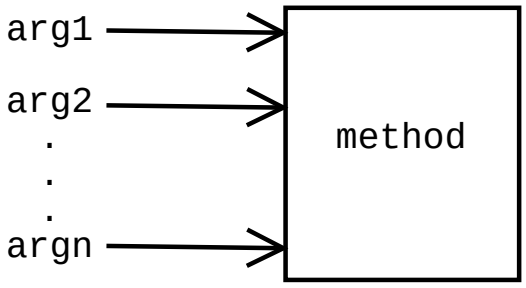
- Methods can be viewed as a "black box" with inputs and outputs:



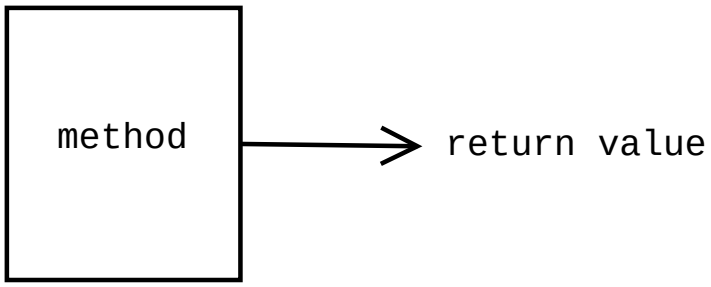
- There are three kinds of methods:
  - Mutators: Modify the state of objects,
  - Accessors: Return information about the object,
  - Constructors: Initialize a newly created object.

Method types

- Mutators are usually `void` methods, which do not return anything, but modify the state of the object:



- Accessor methods may only return values without expecting any arguments as input:



Constructors

- Special methods, whose syntax is given by

```
class_name(list_of_arguments)
{
    statements;
}
```

- For example:

```
public class Student {
    //...
    Student(String n, long i)
    {
        name = n;
        id = i;
    }
    //...
}
```

Constructors (contd.)

- A constructor method gets executed when a new object of the class gets created using the new keyword. Therefore, the general syntax for the expression used to create objects is:

```
new class_name(list_of_actual_arguments);
```

- For example

```
Student al;
al = new Student(“Alan Turing”, 110011223331);
```

Software clients

- A *client* of a class C (or software component in general) is any other class (or software component) which uses C.

- For example:

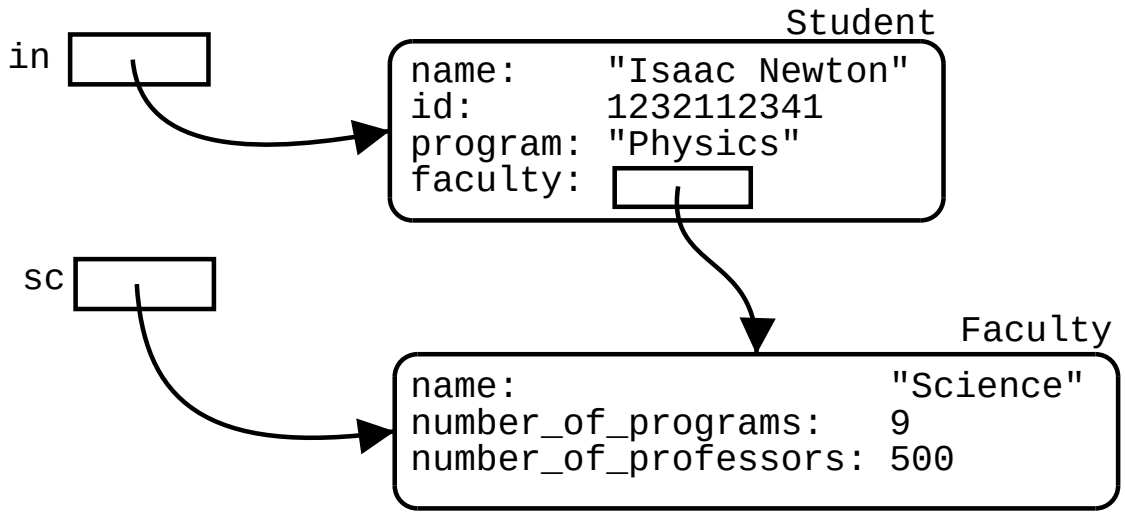
```
public class StudentDatabase {
  public static void main(String[] args)
  {
    Student dave = new Student(“David Hilbert”,
                               1223334444);
    Student bert = new Student(“Bertrand Russell”,
                               1111222334);
    dave.set_prog_and_faculty(“Math”,“Science”);
    //...
  }
}
```

Classes are data types

```
public class Faculty
{
    String name;
    int number_of_programs, number_of_professors;
    //...
}

public class Student
{
    String name;
    long id;
    String program;
    Faculty faculty;
    //...
    void set_prog_and_faculty(String p, Faculty f)
    {
        program = p;
        faculty = f;
    }
    //...
}
```

```
public class StudentDatabase
{
  public static void main(String[] args)
  {
    Faculty sc = new Faculty();

    sc.name = "Science";

    Student in = new Student("Isaac Newton",
                             1232112341);

    in.set_prog_and_faculty("Physics", sc);

    //...
    System.out.println(sc.name);
    System.out.println(in.name);
  }
}
```

Object structure in memory




```
in.set_prog_and_faculty("Physics", new Faculty());
```

doesn't create the variable `sc`, but then, the `Faculty` object cannot be shared between different `Student` objects.

Scope

- Different classes can have attributes and methods which have the same names.

- For example given the following class definition

```
public class C {
    int a;
    //...
}
```

the variables `x.a` and `y.a` are different memory locations in the following client:

```
public class D {
    void m()
    {
        C x = new C();
        C y = new C();
        x.a = 3;
        y.a = 5;
    }
}
```

Scope (contd.)

- This also applies if the attributes are in different classes:

```
public class C {
  int a;
  // ...
}
public class E {
  int a;
  // ...
}
public class D {
  void m()
  {
    C x = new C();
    E y = new E();
    x.a = 3;
    y.a = 5;
  }
}
```

Scope (contd.)

- The *scope* of a variable, a parameter, an attribute or a method is the part of the program that can access that variable, attribute or method.

- The scope of a parameter of a method is the body of the method.

- Variables declared in the body of a method are called *local variables*, which means that their scope is only the body of the method.

- The direct scope of an attribute of a class or a method is the class itself (e.g. the direct scope of `id` is the `Student` class.)

- However, the indirect scope of an attribute of a class or a method is the rest of the program (e.g. the `id` attribute can be accessed by other clients with the expression `var.id`, where var is of type `Student`.)

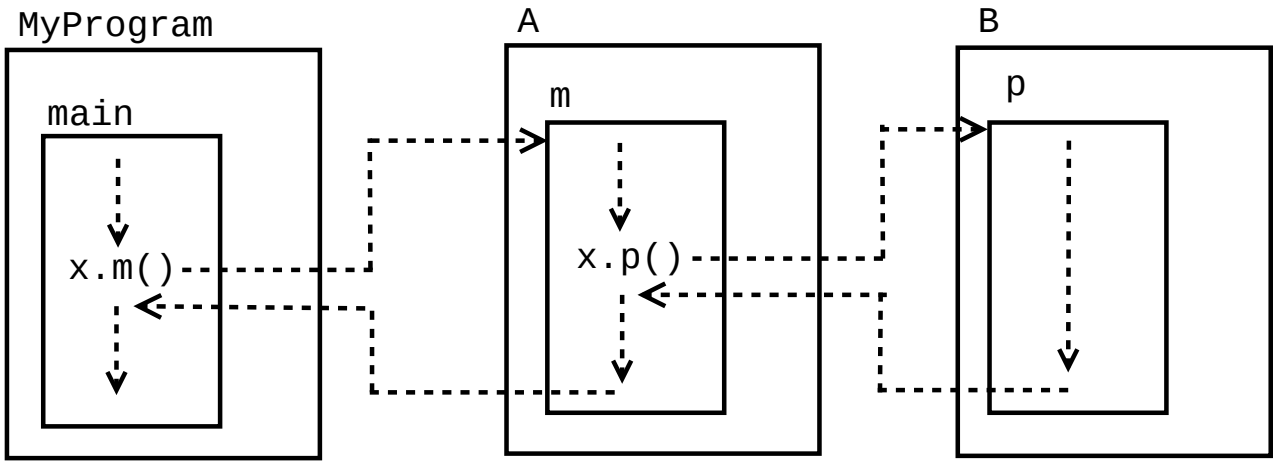
Program Structure

```
public class MyProgram {
  public static void main(String[] args)
  {
    //...
  }
}

public class A {
  //...
}

public class B {
  //...
}
```

Method invocation: control flow

```
public class MyProgram {
  public static void main(String[] args)
  {
    A x = new A();
    x.m();
    System.out.println(“Main done”);
  }
}
public class A {
  void m()
  {
    B x = new B();
    x.p();
    System.out.println("m done");
  }
}
public class B {
  void p()
  {
    System.out.println(“Do something”);
  }
}
```

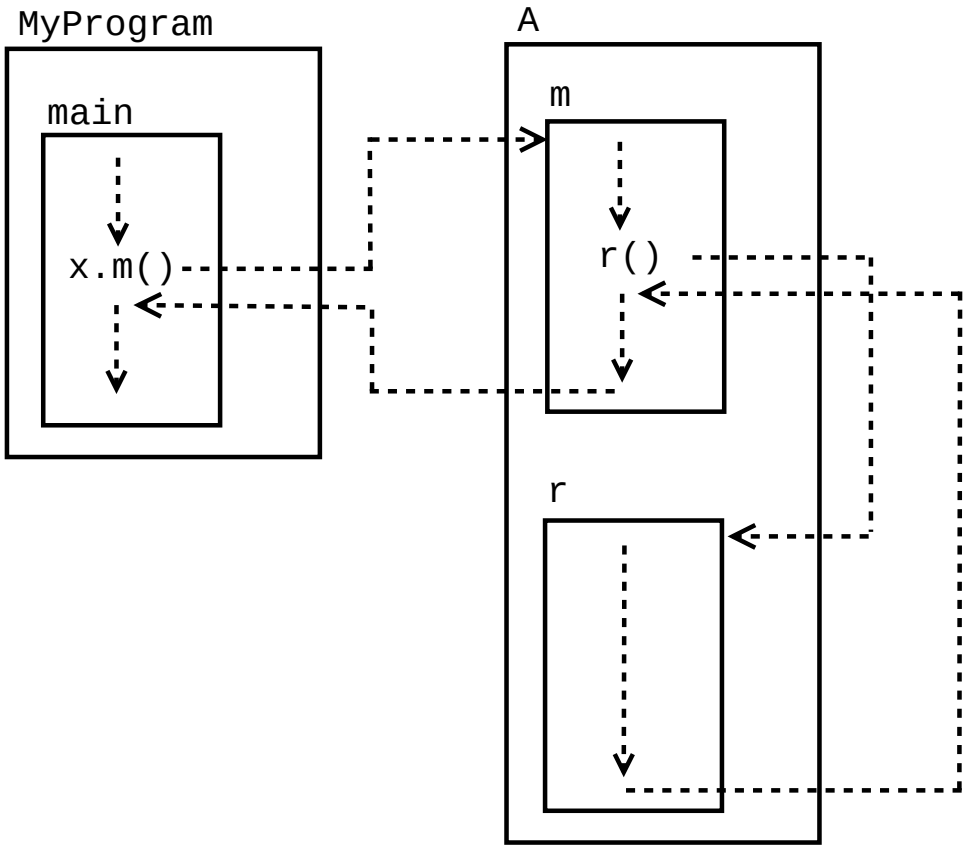
Method invocation: control flow



Method invocation: control flow; “this”

```
public class MyProgram {
  public static void main(String[] args)
  {
    A x = new A();
    x.m();
    System.out.println(“Done”);
  }
}
public class A {
  void m()
  {
    r();    // Equivalent to this.r();
  }
  void r()
  {
    System.out.println("Do something");
  }
}
```

Method invocation: control flow



Method invocation: parameter passing

- A *frame* is a space in memory which stores a set of variables. It can be viewed as a table containing the memory locations for each variable in the set.

- Suppose that a method is declared as follows:

```
type method(type1 param1, type2 param2,
            ..., typen paramn)
{
    statements;
}
```

- A method call of the form

```
variable.method(arg1, arg2, ..., argn)
```

...where *arg1*, *arg2*, ..., *argn* are expressions with type matching the types as appear in the method declaration, is executed by

First: evaluating each of the arguments *arg1*, *arg2*, ..., *argn* from left to right,

Second: creating a *frame*, reserving space for all the parameters of the method, and local variables declared in the body of the method. The frame also contains a pointer to the object refered to by the *variable*.

Third: in that frame, perform the assignments *param1* = *arg1*; *param2* = *arg2*; ...; *paramn* = *argn*;

Fourth: "jumping" to the body of the method and executing the *statements* in order. The calling method is suspended while the called method is executed.

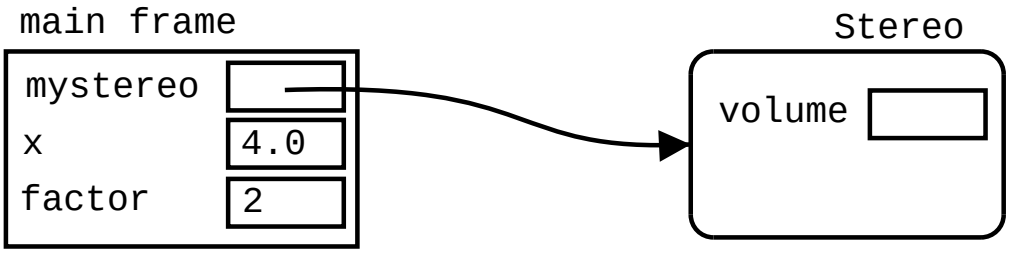
Fifth: when the end of the method is reached, or a `return` statement is reached, stop the method, the frame is discarded, and return to the calling method. The calling method is then resumed in the instruction immediatly after the method call.

Method invocation: Example

```
public class Stereo {
    double volume;
    void set_volume(double v)
    {
        volume = v;
    }
    double get_volume()
    {
        return volume;
    }
}
public class SoundSystem {
    public static void main(String[] args)
    {
        Stereo mystereo = new Stereo();
        double x, factor = 2;
        System.out.println("Testing...");
        x = 4.0;
        mystereo.set_volume(x*factor);
        System.out.println(mystereo.get_volume());
    }
}
```

Method invocation: Memory structure

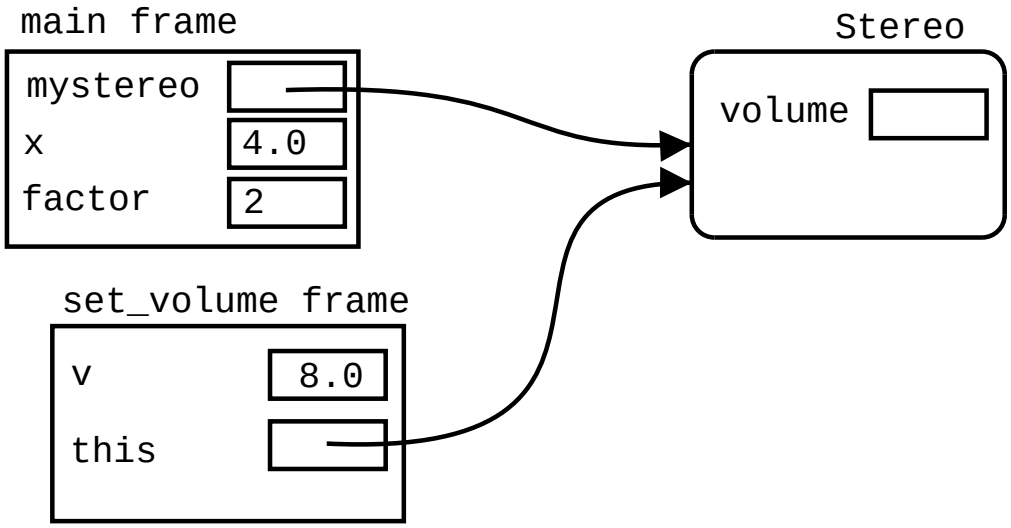
Before calling `mystereo.set_volume(x*factor)`



First its arguments (`x*factor`) are evaluated:
Evaluating `x*factor` in the main frame results in `8.0`

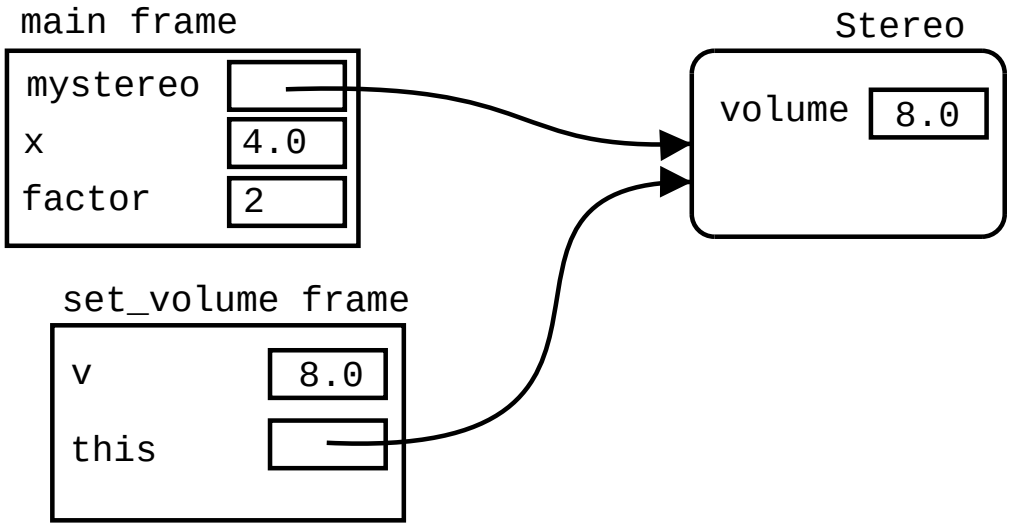
Method invocation: Memory structure

A frame for set_volume is created, and
the argument is assigned to the parameter: `v = 8.0;`



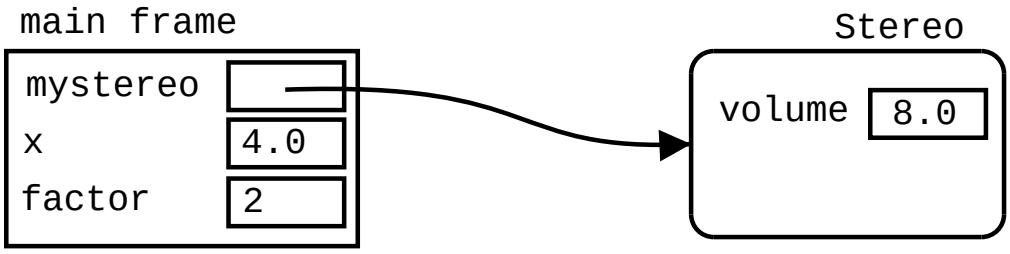
Method invocation: Memory structure

The current method (main) is suspended, and
the body of the called method (set_volume) is executed
in the context of the current frame (the set_volume frame):



Method invocation: Memory structure

Finally the called method frame is discarded, and computation of the calling method (main) is resumed in the instruction immediately after the method call.



The end