Announcements

- Assignment 3 has been posted.

Statements

- Variable declaration

  ```
  type variable;
  ```

- Assignment

  ```
  variable = expression;
  ```

- Method invocation

  ```
  objectreference.methodname(parameters);
  ```

  or

  ```
  classname.methodname(parameters);
  ```

- Conditional

  ```
  if (condition) block;
  ```

  or

  ```
  if (condition) block1; else block2;
  ```

- Loop

  ```
  while (condition) block;
  ```

Program structure

- A java program is made of *classes*
- Classes are made of
  - *attributes* (variable declarations,) and
  - *methods*
- Methods are made of statements

Program structure

```
// File: A.java
public class A {
  // ...
}

// File: B.java
public class B {
  // ...
}
```

Program structure

```
public class A {
  int x;

  void f()
  {
    // ...
  }

  void g()
  {
    // ...
  }
}
```

Program structure

```
public class A {
  int x;

  void f()
  {
    x++;
  }

  void g()
  {
    System.out.println(x);
    x--;
  }
}
```

Program structure

```
public class A {
  int x;

  x++;
  void f()
  {
    x++;
  }

  void g()
  {
    System.out.println(x);
    x--;
  }
}
```

Program structure

```
// File: A.java
public class A {
  // ...
}

// File: B.java
public class B {
  // ...
}

// File: C.java
public class C {
  public static void main(String[] args)
  {
    //...
  }
}
```

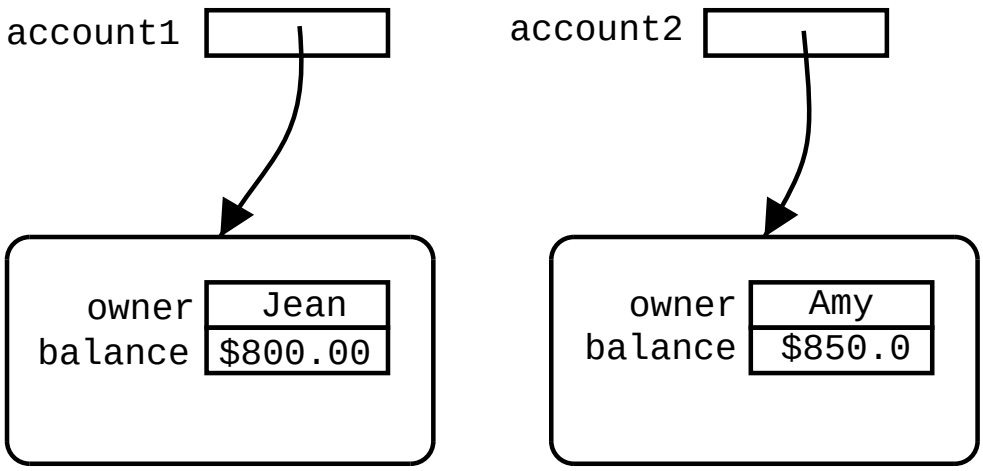
Classes

- Classes have a dual role in Java:
  - They are modules
  - They are the data-type of *objects*
- Information in a Java program is represented by either
  - Primitive data (e.g. numbers, booleans)
  - Objects (composite data)
- An *object* is a composite piece of data which can be applied certain actions or operations:
  - An object is "made up" of other (simpler) pieces of data (primitive or objects)
  - An object is a group of data "glued" toghether that can be treated a s a unit, a single piece of data
  - An object can "react" to operations we appy to it

Objects and Classes

- A bank account has:
  - owner
  - balance
- Given a bank account we can:
  - deposit
  - withdraw

Objects and Classes



Objects and Classes

- Objects have a type
- The type of an object is a *class*
- A class describes:
  - the structure of its objects (attributes)
  - and its operations (methods)
- A class is ***not*** the same as an object
- A class is like the "blueprint" of a family of objects
- An object is a particular *instance* of a class

Objects and Classes

- To be able to use objects we need:
  - Define some class or classes
  - A mechanism to create objects of a defined class
  - A mechanism to apply operations to these objects

Class definition

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

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

Objects and Classes

- Defining a class:

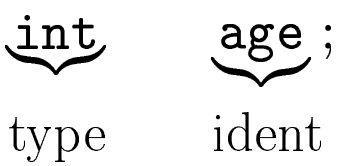
```
public class BankAccount
{
    String owner;
    double balance;

    void withdraw(double amount)
    {
      // ...
    }

    void deposit(double amount)
    {
      // ...
    }
}
```

Classes and Objects

- Declaring a variable:

  type identifier ;

- It is the same for primitive types



  as for non-primitive types (classes)

Classes and Objects

- Declaring a variable does not create any objects
- To *create objects* we use the `new` operator

```
account1 = new BankAccount("Jean");
```

- To *apply operations to objects* we use the *dot* operator:

```
account1.deposit(200.00);
```

- You cannot apply methods without first creating objects

Classes and Objects

- To *create objects* we use the `new` operator

```
objectvariable = new ClassName(parameters);
```

- To *apply operations to objects* we use the *dot* operator:

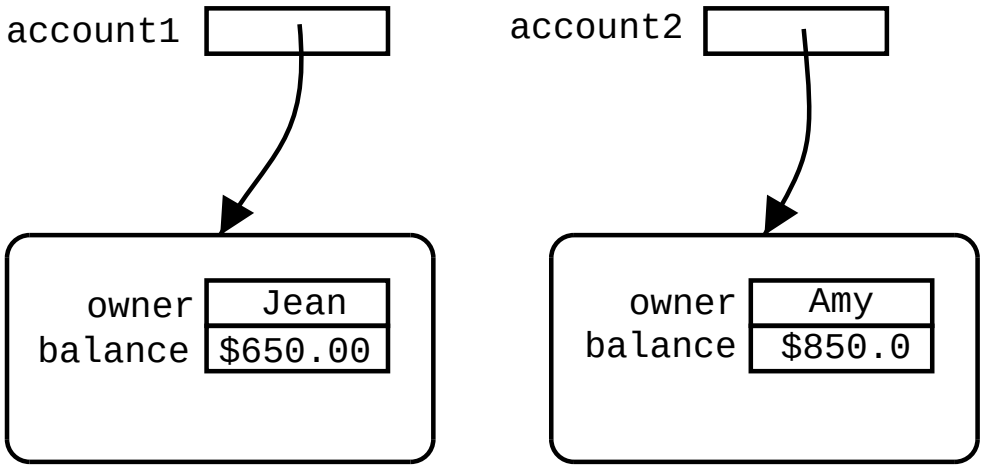
```
objectvariable.method(parameters);
```

Objects and Classes

account1 . withdraw (150.00);

object method parameters

- Applying a method to an object affects only the object it is being applied to.



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;
    }
    // ...
}
```

Example: Stereo

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

    while (!radio_on) current_song++;  //WRONG!

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

Objects and classes

- A class is a "type" of objects. Objects are the values of a class.
- A class is defined by the attributes shared by all its objects, and by its methods
- The attributes of a class represent those characteristics which all objects of the class *have*: e.g. every student has a *name* and an *id*. Hence, `name` and `id` can be attributes of a `Student` class.
- The methods of a class represent the operations that can be performed on objects of that class, they define how an object in the class reacts to "messages" sent to it by other objects: e.g. the method `play` in the `Stereo` class, defines how all stereo objects react to the message "play".

Objects and classes

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

- These translate into code as "class definitions"

```
public class ClassName
{
  Attribute definitions

  Method definitions
}
```

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
```

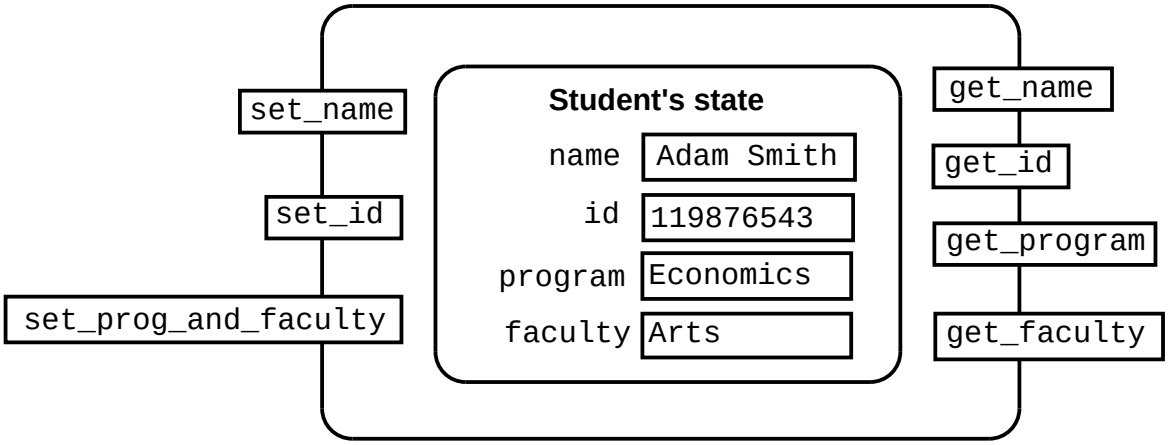
- The parameters are the inputs to the method, to be provided by other methods in a method call.

Example

```
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.
```

An object of the Student class

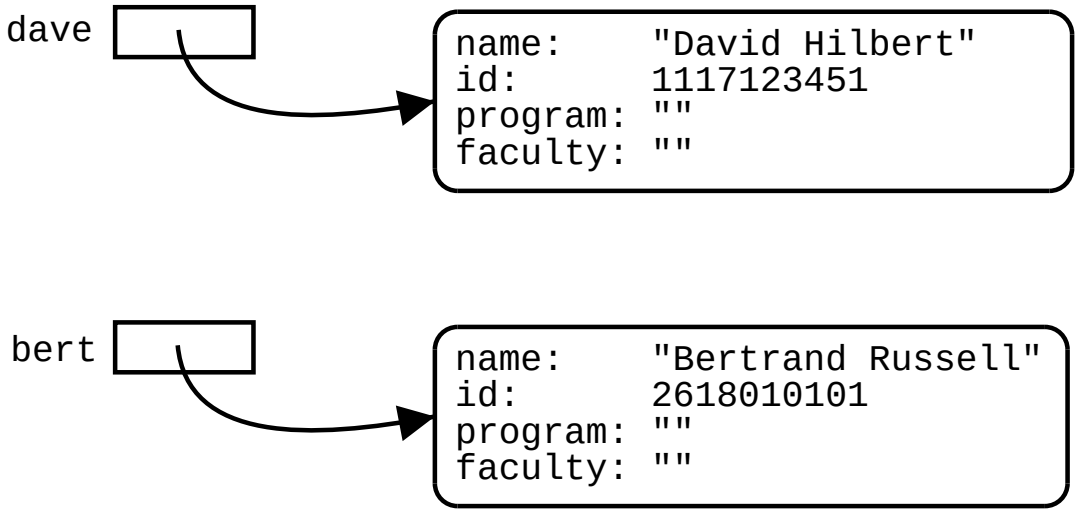


Objects are not classes

- A class can be thought of as a data type. Its values are objects.
- 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.

Individual identity of objects

Student
+name: String +id: long +program: String +faculty: String
+set_name(n:String): void *+set_id(n:long): void* *+set_prog_and_faculty(p:String,f:String): void* *+get_name(): String* *+get_id(): long* *+get_program(): String* *+get_faculty(): String*



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

  ```
  Student dave;

  dave = new Student();
  ```

- The two can be done in one line:

  ```
  Student bert = new Student();
  ```

Accessing attributes

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

  variable.*attribute*

  ...but only if the attribute exists in the class of the variable.

- Example:

```
dave.name = "David Hilbert";
dave.id = 111712345l;
System.out.println(dave.name);
System.out.println(dave.id);

bert.name = "Bertrand Russell";
bert.id = 261801010l;
System.out.println(bert.name);
System.out.println(bert.id);
```

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):

  ```
  variable.method_name(arguments)
  ```

  where the type of `variable` 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.

Sending messages (contd.)

- For example:

```
bert.set_prog_and_faculty(“Philosophy”, “Arts”);

dave.set_id(009876543);
```

- A method call

  ```
  a.m(b, c, d);
  ```

  could be interpreted as “sending the message `m` to the object `a` with arguments `b`, `c`, and `d`.”

Example

```
public class BankAccount
{
    String owner;
    double balance;

    void withdraw(double amount)
    {
      if (amount <= balance)
        balance = balance - amount;
    }

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

Example

```
public class BankExample {
  public static void main(String[] args)
  {
    BankAccount a1;
    double x;

    a1 = new BankAccount();
    a1.owner = “John”;
    a1.balance = 0.0;
    a1.deposit(200.0);
    x = a1.balance;
    System.out.println(x);
  }
}
```

Example

```
public class BankExample {
  public static void main(String[] args)
  {
    BankAccount a1;
    double x;

    a1.owner = ‘‘John’’;
    a1.balance = 0.0;
    a1.deposit(200.0);
    x = a1.balance;
    System.out.println(x);
  }
}
```

Example

```
public class BankExample {
  public static void main(String[] args)
  {
    BankAccount a1, a2;
    double x, y;

    a1 = new BankAccount();
    a1.owner = “John”;
    a1.balance = 0.0;
    a2 = new BankAccount();
    a2.owner = “Marie”;
    a2.balance = 100.0;
    a1.deposit(200.0);
    a2.withdraw(50.0);
    x = a1.balance;
    y = a2.balance;
    System.out.println(x);
    System.out.println(y);
  }
}
```

Method parameters

The following class

```
class SomeClass {
  void someMethod(int parameter)
  {
    //...do something
  }
}
```

is ***not*** the same as

```
class SomeClass {
  void someMethod()
  {
    int parameter;
    //...do something
  }
}
```

Method parameters

- Parameters are information provided to a method by another method somewhere else in the program.

```
SomeClass obj1;
obj1 = new SomeClass();
obj1.someMethod(73);  //This works with the first
```

- Inputs are not only given by the user. The input to a method (its parameters) are provided by other methods.

Method parameters

```
public class BankAccount
{
    String owner;
    double balance;

    void withdraw(double amount)
    {
      if (amount <= balance)
        balance = balance - amount;
    }

    void deposit()
    {
      double amount = Keyboard.readDouble();
      balance = balance + amount;
    }
}
```

Example

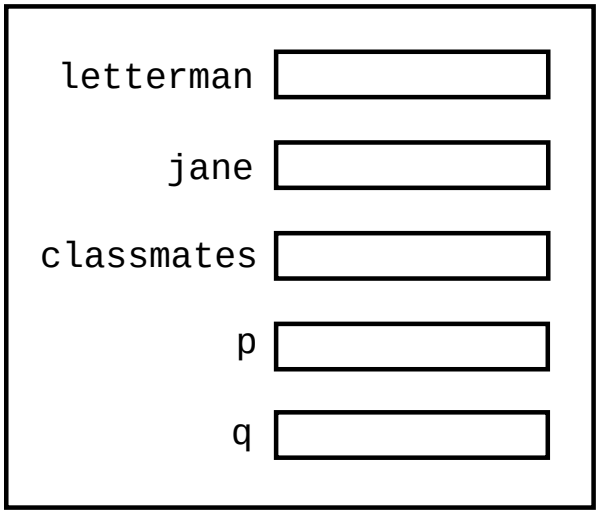
```
// in a file called Student.java
public class Student
{
    String name;
    long id;
    String program;
    String faculty;

    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.
```

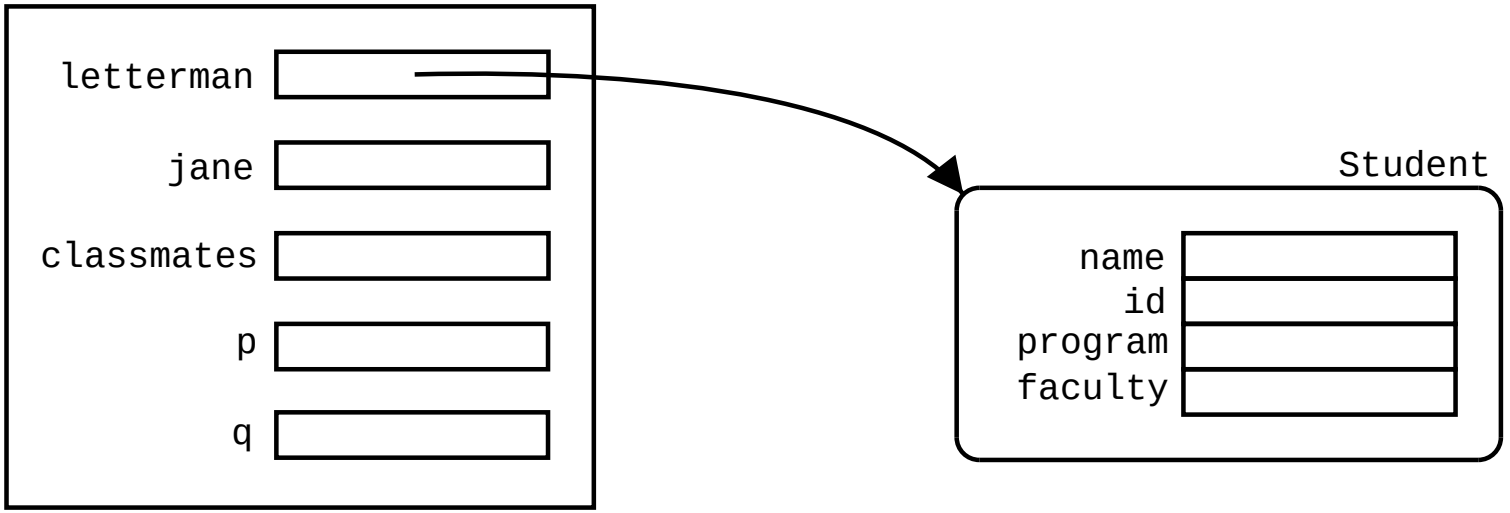
Example

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

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

letterman.set_name(“David”);
letterman.set_id(00000011);

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
```

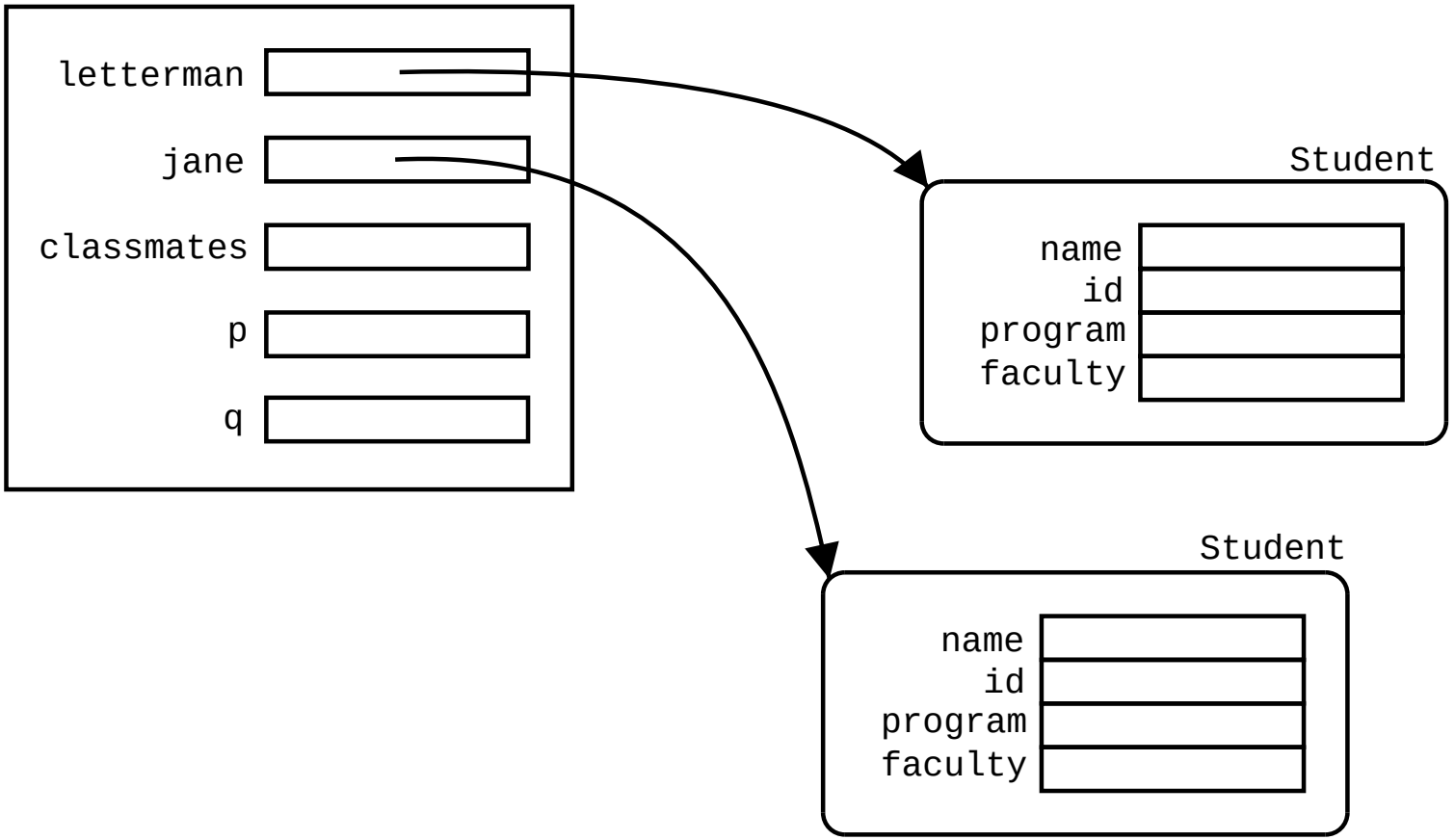
Example



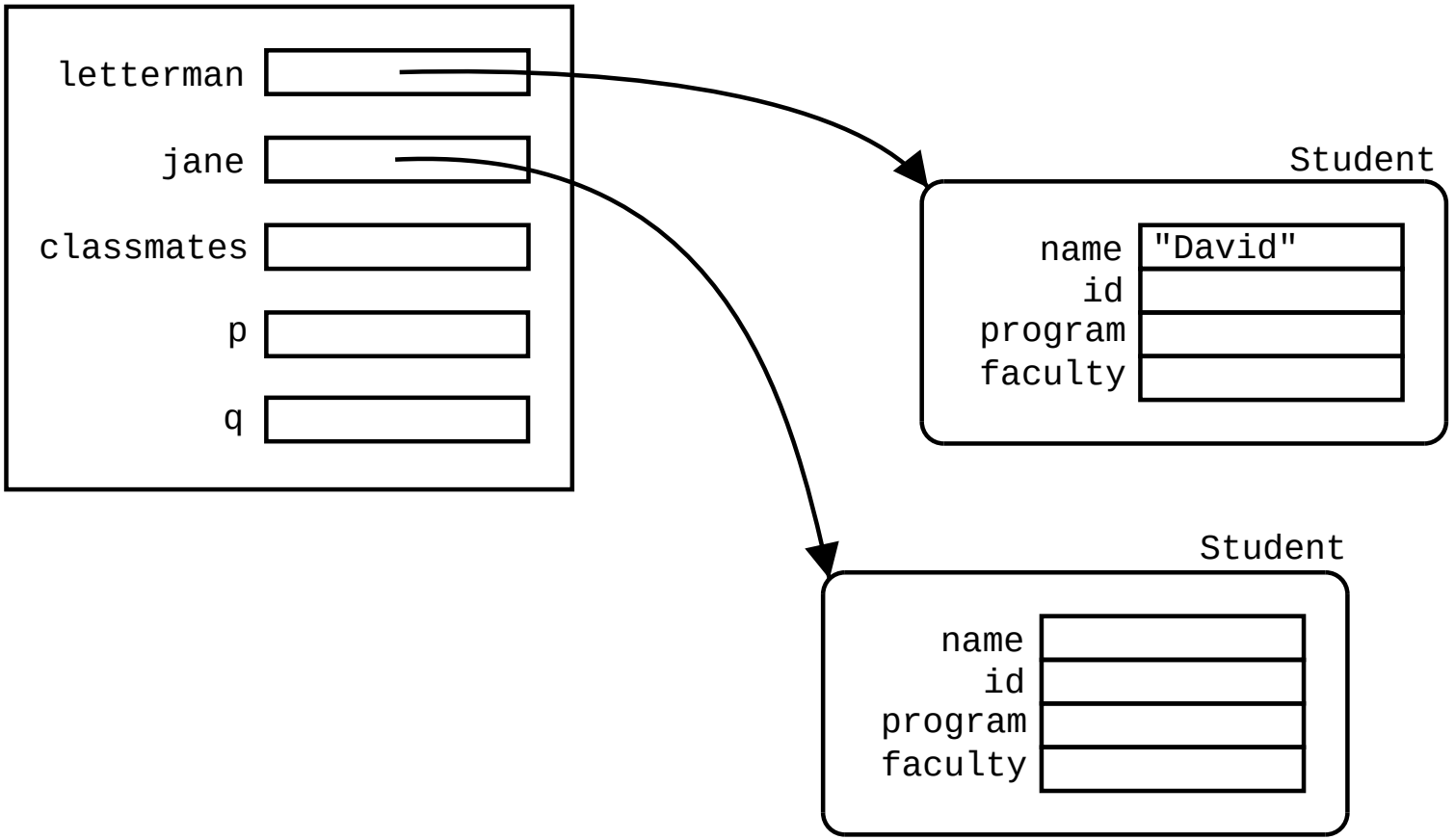
Example


letterman
jane
classmates
p
q
Student
name
id
program
faculty

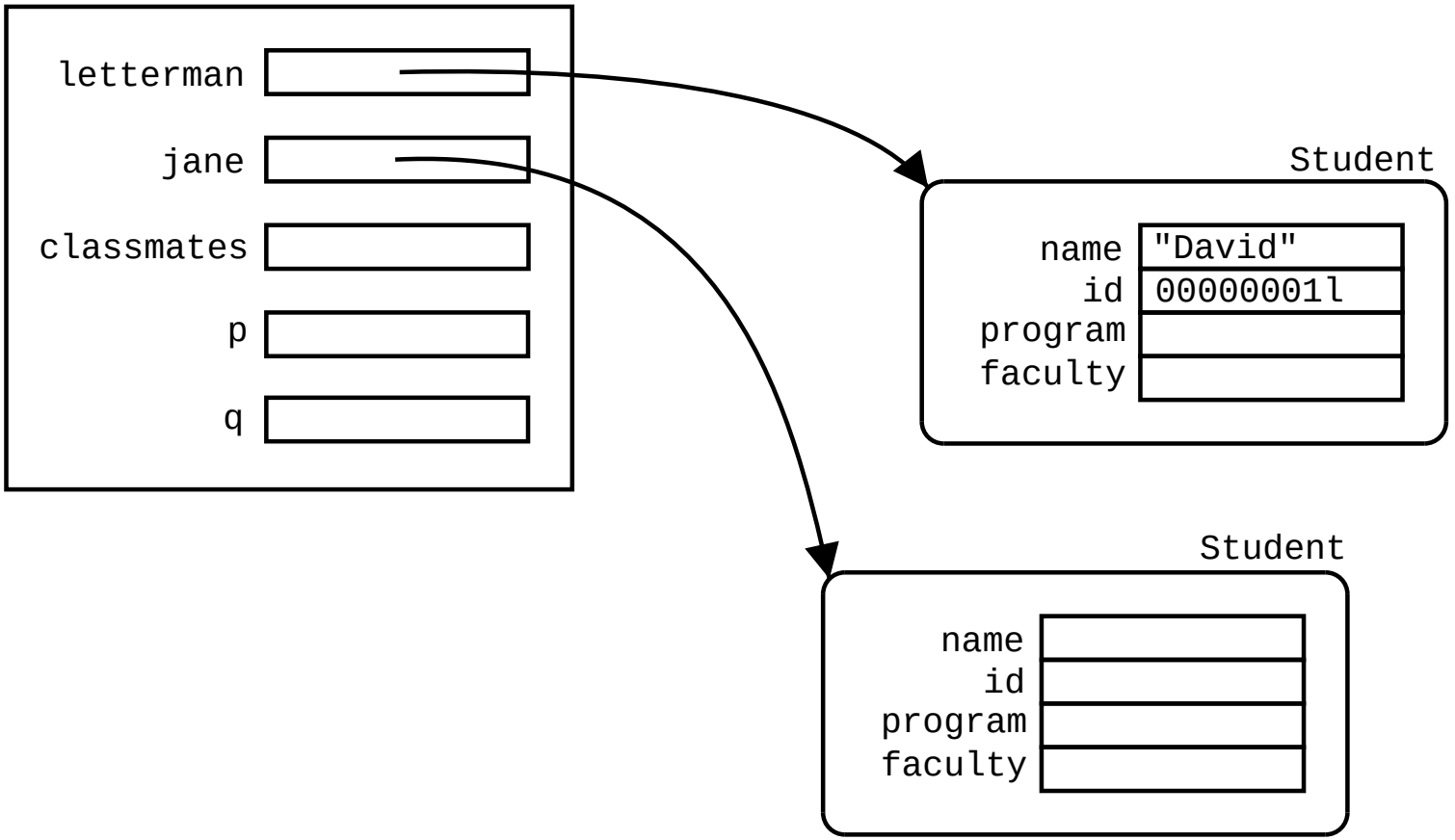
Example


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

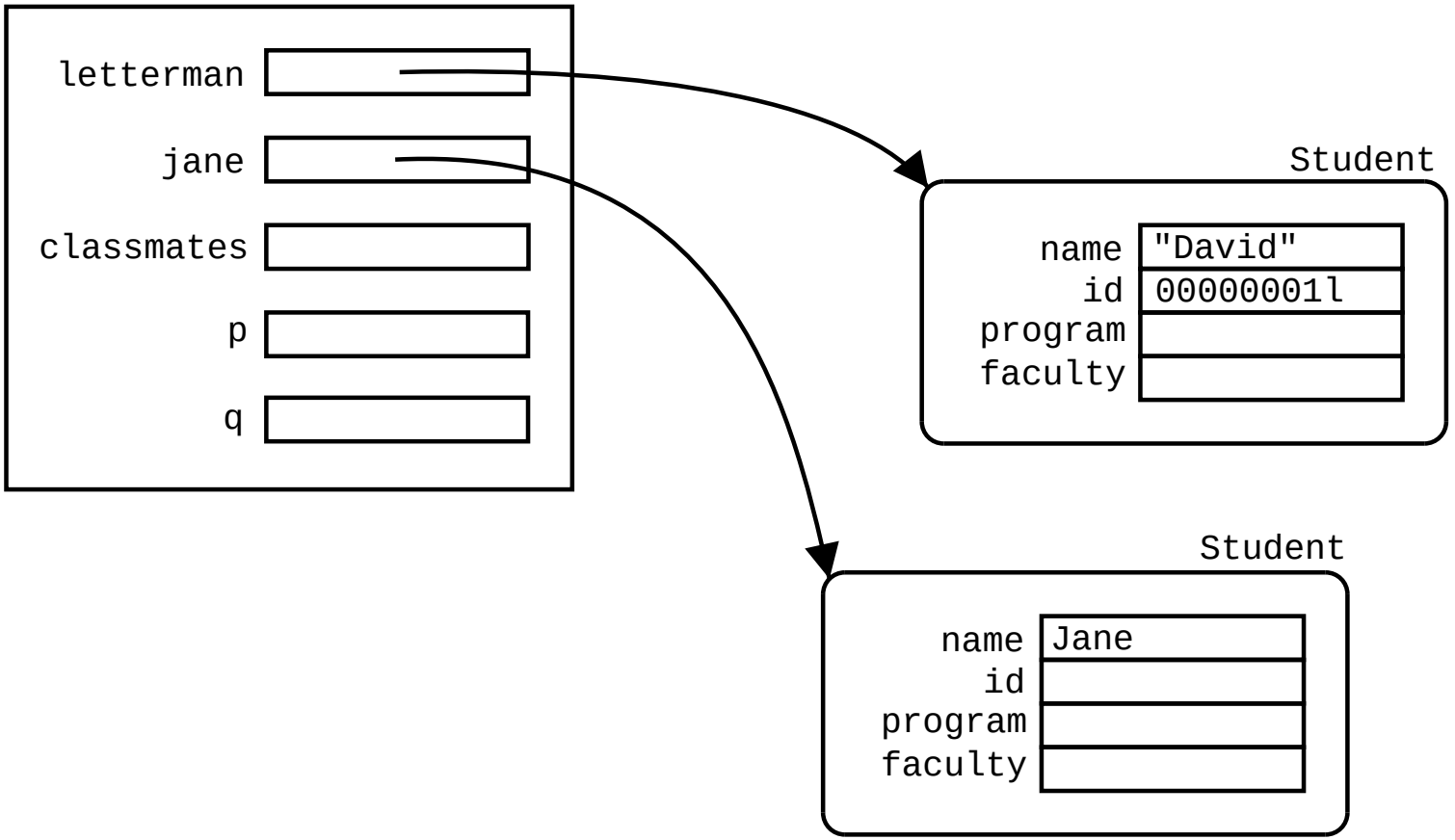
Example


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

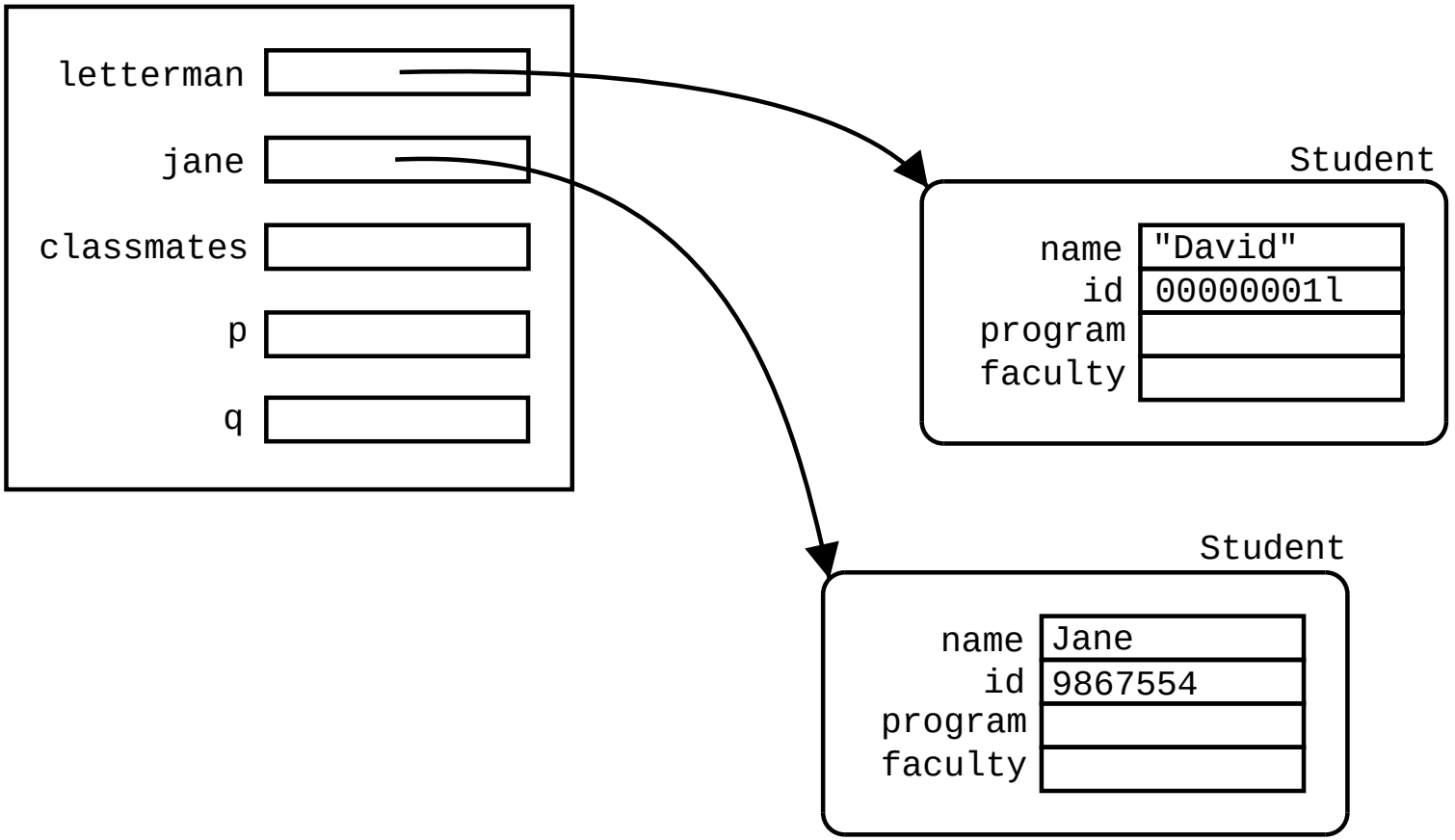
Example


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

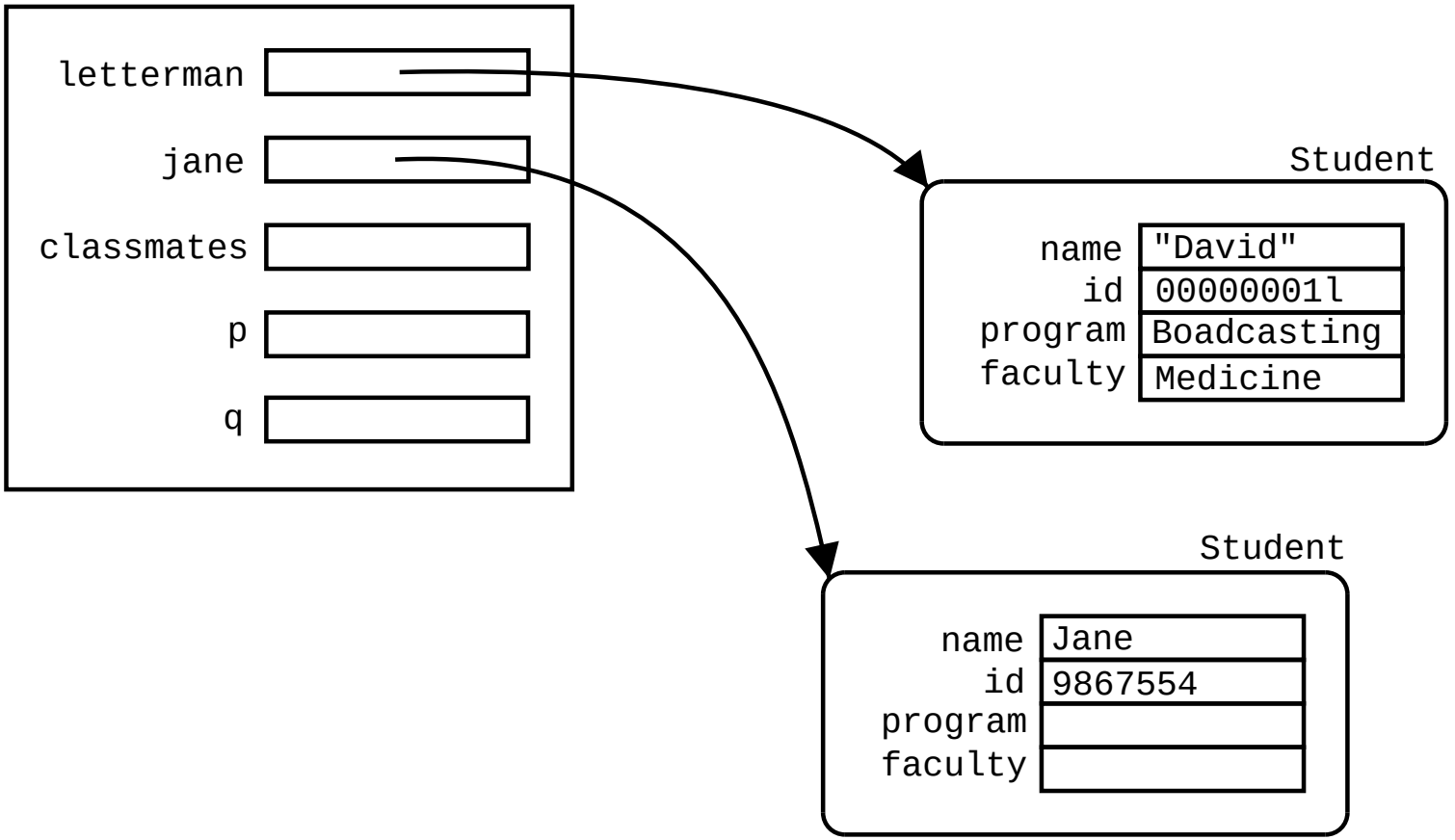
Example


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

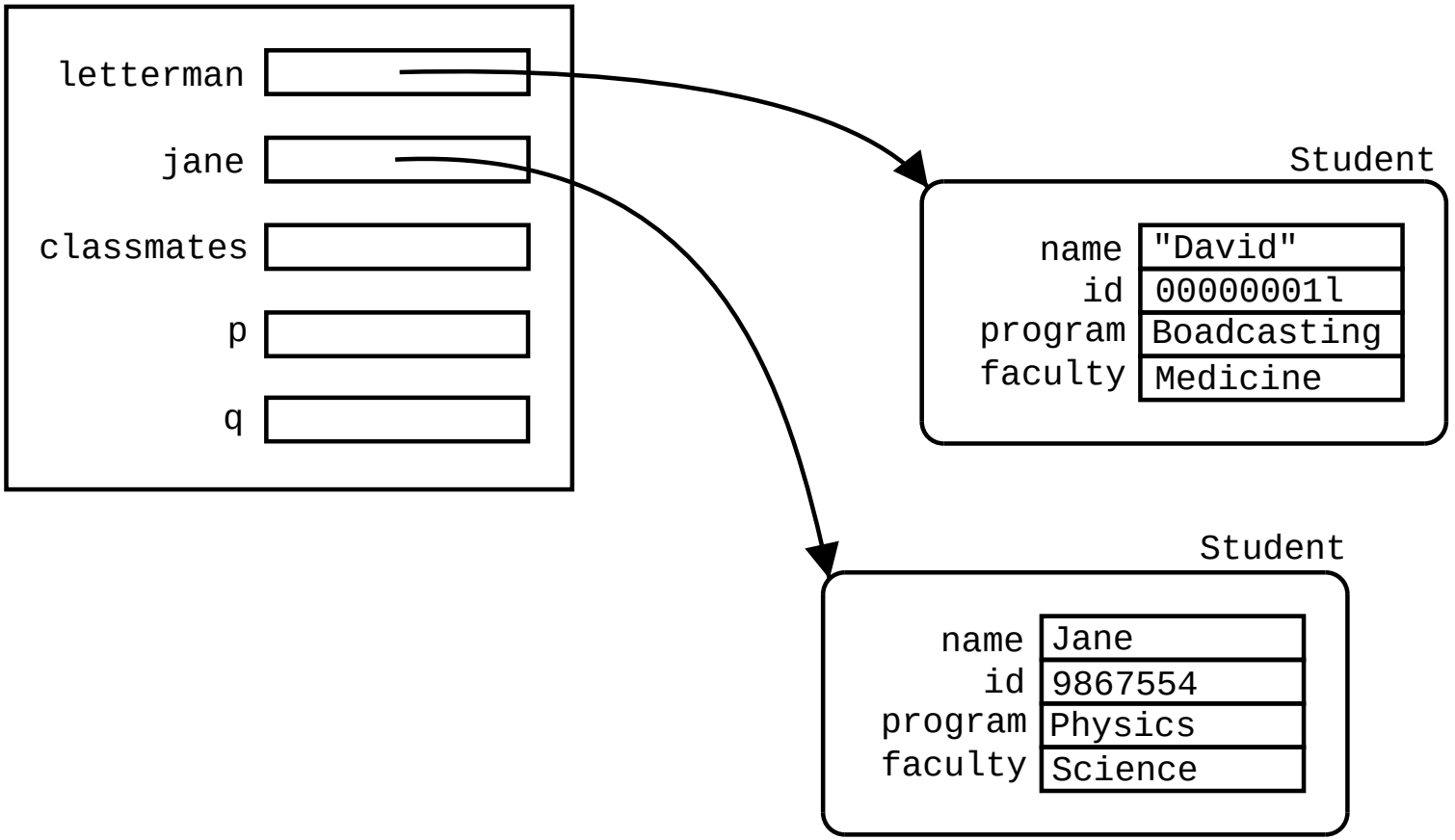
Example


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

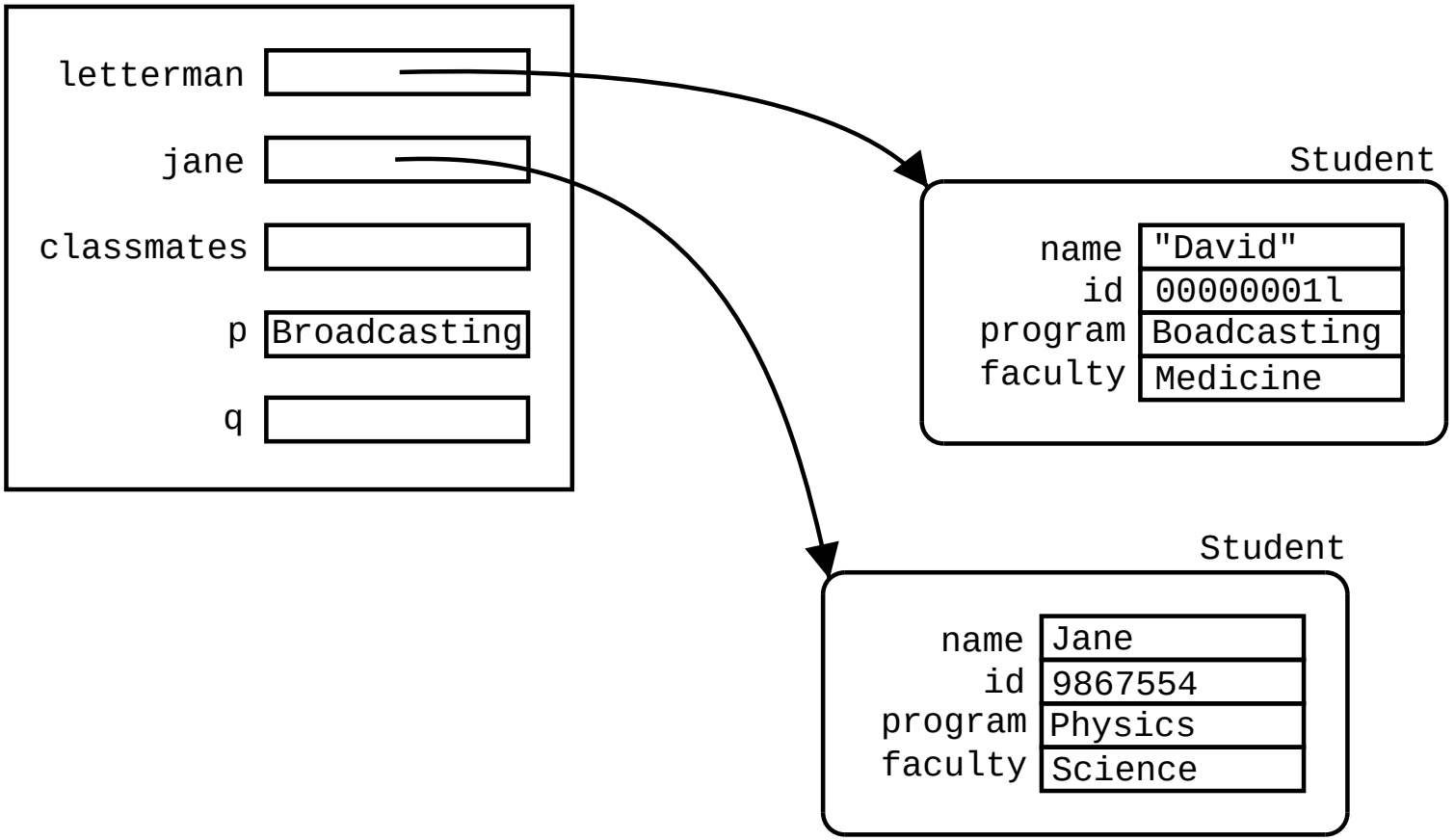
Example


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

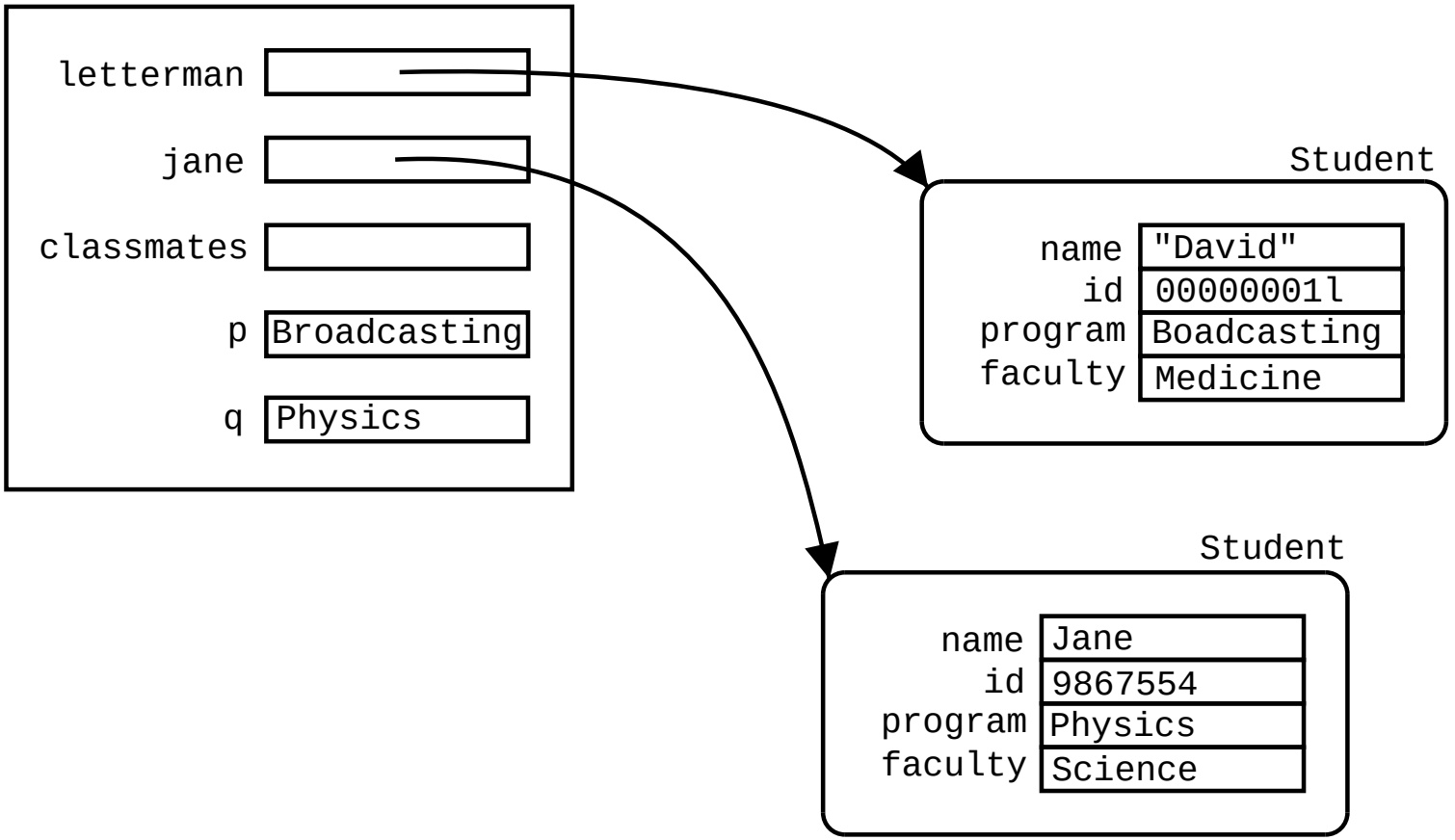
Example


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

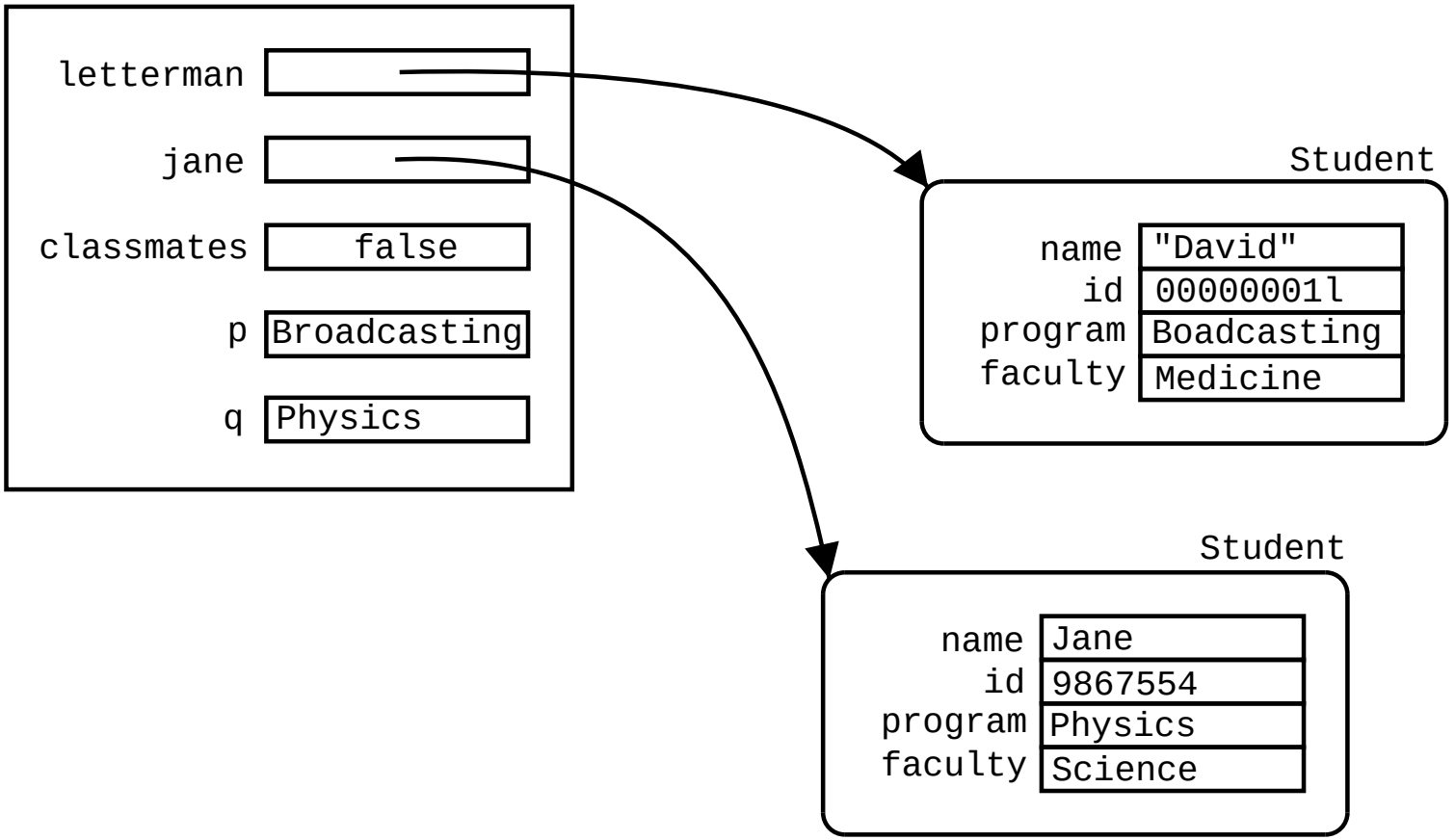
Example


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


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


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

Method calls in context

- There are two forms of method calls:
  - Method call as a statement
  - Method call as an expression
- A method call is a statement if its return type is `void`, otherwise it is an expression.
- If a method call is an expression, it must appear in a context that allows expressions, such as:

  A. the right hand-side of an assignment:

  ```
  long n = dave.get_id();
  String s = dave.get_program();
  ```

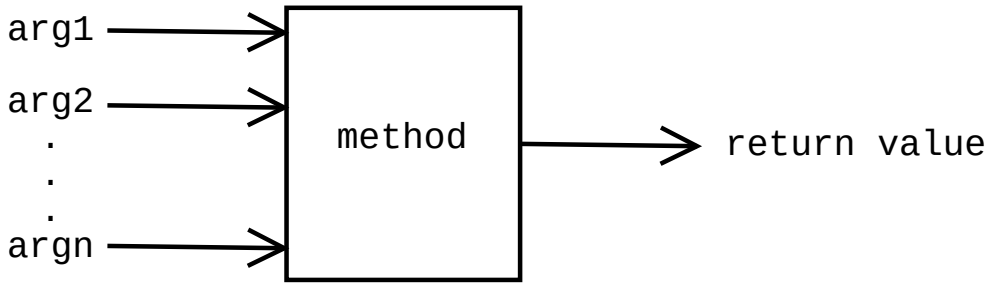
  B. ...or, the argument of another method:

  ```
  System.out.println(dave.get_id());
  bert.set_id(dave.get_id());
  ```

- But the types **must** match!

Methods as functions

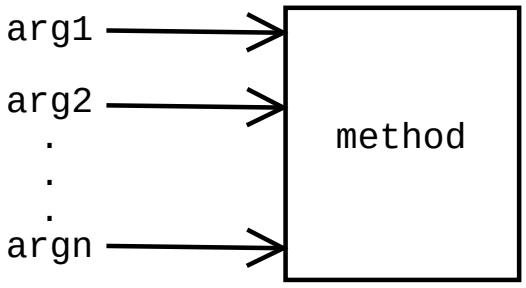
- Methods can be viewed as a "black box" with inputs and outputs:



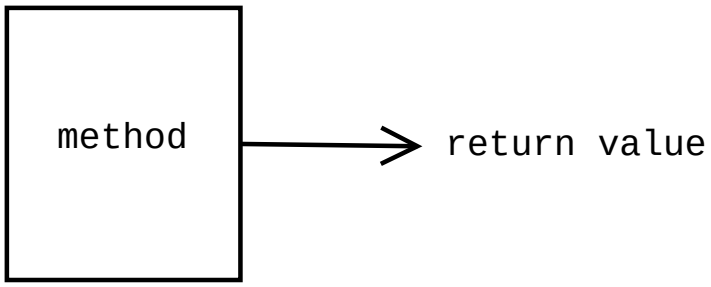
- There are three kinds of methods:
  - Mutators: Modify the state of objects,
  - Accessors: Return information about the object,
  - Constructors: Initialize a newly created object.

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:



The end