Statements

- Variable declaration

```
type variable;
```

- Assignment

```
variable = expression;
```

- Conditional

```
if (condition) { stmts; }
```

or

```
if (condition) { stmts1; } else { stmts2; }
```

- Loop

```
while (condition) { stmts; }
```

- Method invocation

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

or

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

Object-Oriented Programming

- Java is an *object-oriented* programming language
- The fundamental notion is that of an *object*
- Objects represent entities of a problem (possibly real-world entities)
- A program defines objects that interact with each other

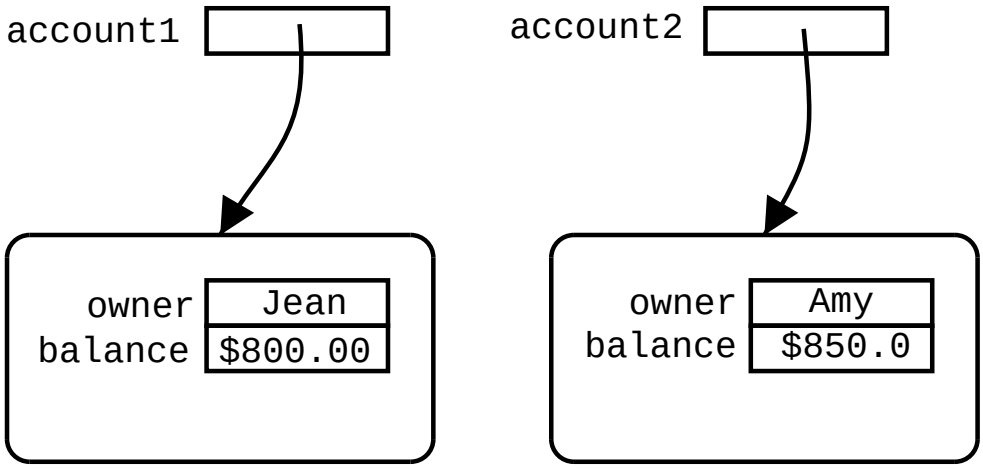
Objects

- An *object* is a *composite* and *reactive* piece of data
  - A piece of *data*: an object is data, it can be treated as a *unit*, a single piece of data
  - *Composite*: an object is a *group* of data
  - *Reactive*:
    * an object can *react* to *messages* sent to it
    * we can ask an object to perform a task
    * we can apply operations on an object

Objects

- A bank account *has*: (data)
  - owner
  - balance
- Given a bank account we *can*: (operations/behaviour)
  - deposit
  - withdraw

Objects


account1
account2
owner
balance
Jean
$800.00
owner
balance
Amy
$850.0

Objects and data-types

- All values have a data-type, e.g.:
  - The type of 1729 is int
  - The type of -1.618 is double
  - The type of false is boolean
- Objects are values
- Therefore, all objects have a data type

Objects and Classes

- 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

Classes

- Classes have a dual role in Java:
  - They are the data-type of *objects*
  - They are modules
- A single class alone doesn't do anything ...
  - A class is useful in a context of other classes

Program structure

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

Class definition

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

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

Program structure

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

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

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

Program structure

```
public class A
{
  int x;

  void f()
  {
    // ...
  }

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

Program structure

```
public class A
{
  int x;        // Attribute or field

  void f()
  {
    // ...
  }

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

Program structure

```
public class A
{
  int x;

  void f()      // Method
  {
    // ...
  }

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

Program structure

```
public class A
{
  int x;

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

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

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

Objects and Classes

- Defining a class:

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

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

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

Objects and Classes

- Declaring a variable:

  type identifier ;

- It is the same for primitive types

  ```
  int    age;
  ```
  type ident

  as for non-primitive types (classes)

  ```
  BankAccount    account1;
  ```
  type identifier

Objects and Classes

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

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

- 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:

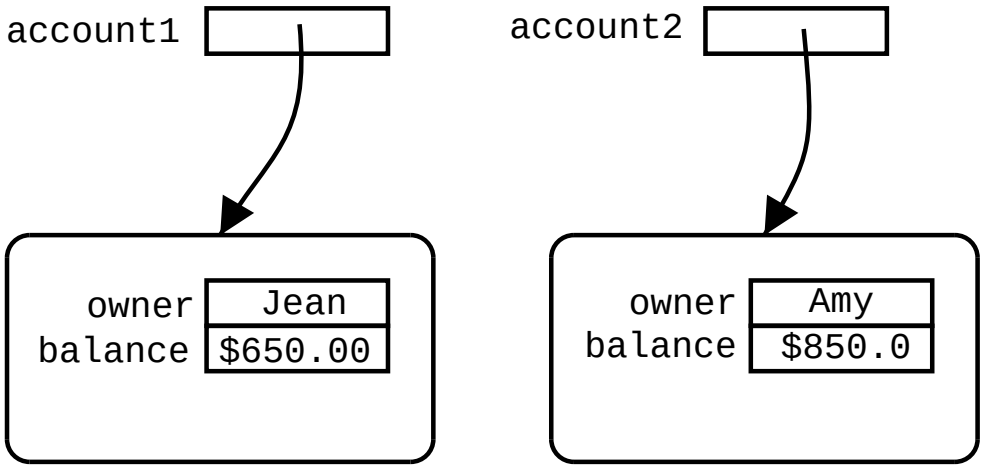
```
objectvariable.method(parameters);
```

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.



Objects and Classes

- Defining a class:

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

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

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

Objects and classes

- A single class alone doesn't do anything ...
- A class is useful in a context of other classes

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
    }
}
```

- Note: only one class in a program has a `main` method

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        BankAccount account2;
        account2 = new BankAccount();
        account2.deposit(150.0);
    }
}
```

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
    }
}
```

Objects and Classes

- Each object has its own separate *identity*, its own individual *state*
- The *state* of an object is the current value of its attributes
- The state of an object can change:
  - when we ask the object to do something
  - ... therefore, the methods of the object's class are responsible for changes to the object's state

Objects and Classes

- Defining a class:

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

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

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

Anatomy of methods

- A method is normally written as

```
void method_name (parameters)
{
  // body
}
```

  where parameters is a (possibly empty) list of variable declarations

```
type1 param1, type2 param2, ..., typen paramn
```

  and body is a list of statements

- The body of the method is executed *only* when the method is *invoked*

Anatomy of methods

- If a method can return a result value, it has the syntax

```
type method_name (parameters)
{
  // body
  return expression;
}
```

  where *expression* is of type *type*

Parameters

- *Parameters*: variables that receive information necessary to execute a method

- Information flow:
  - when a method is invoked,
  - the caller *passes* information to the method in the form of *arguments*
  - and the method receives that information in its parameters

Objects and Classes

- Defining a class:

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

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

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

Objects and Classes

- Defining a class:

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

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

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

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

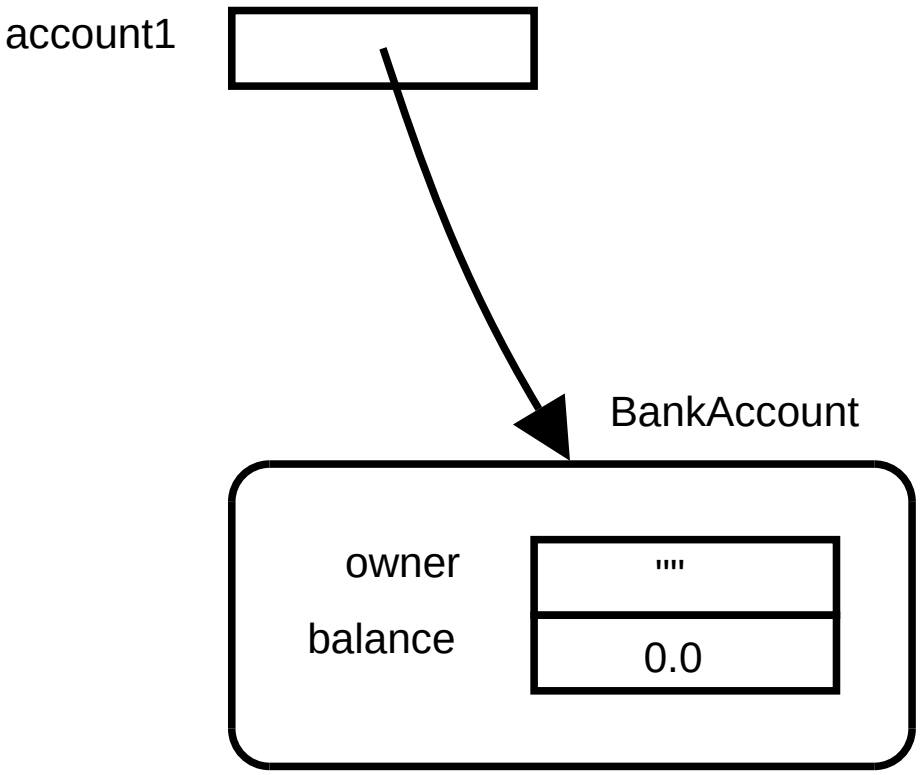
```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        System.out.println("Done");
    }
}
```

Objects and Classes



Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

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

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

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

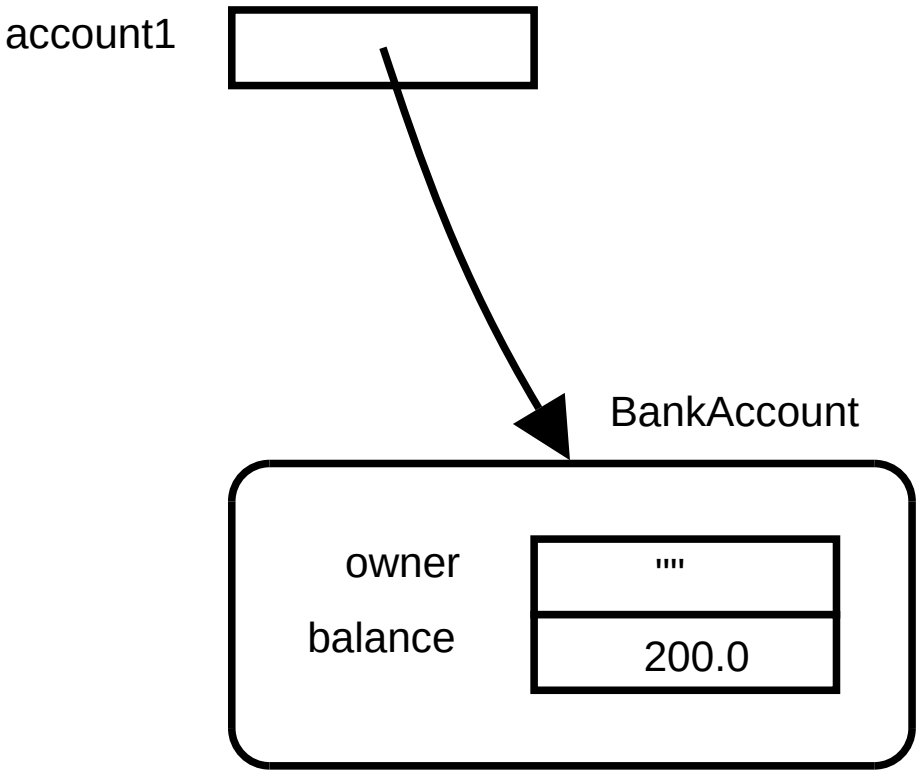
Objects and Classes

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

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

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

Objects and Classes



Objects and Classes

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

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

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

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
        System.out.println(“Done”);
    }
}
```

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
        System.out.println(“Done”);
    }
}
```

Objects and Classes

Objects and Classes

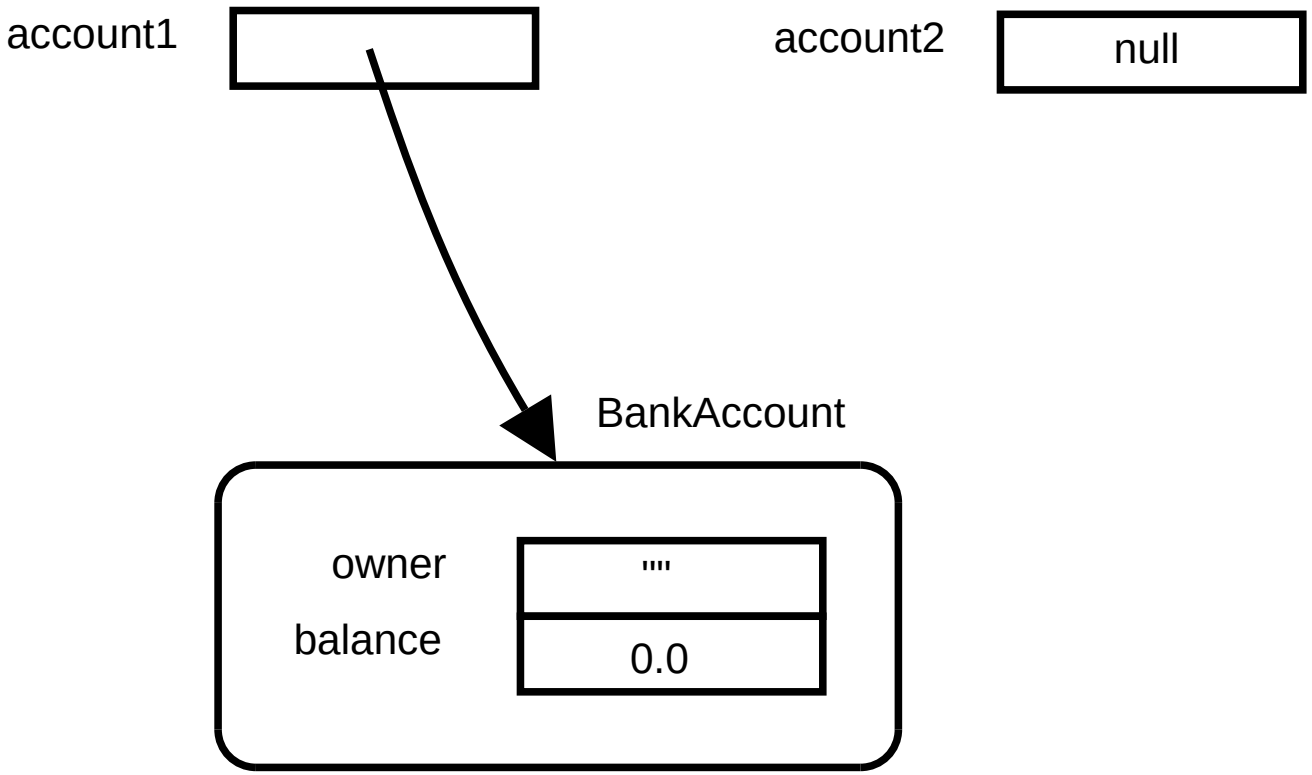
```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
        System.out.println(“Done”);
    }
}
```

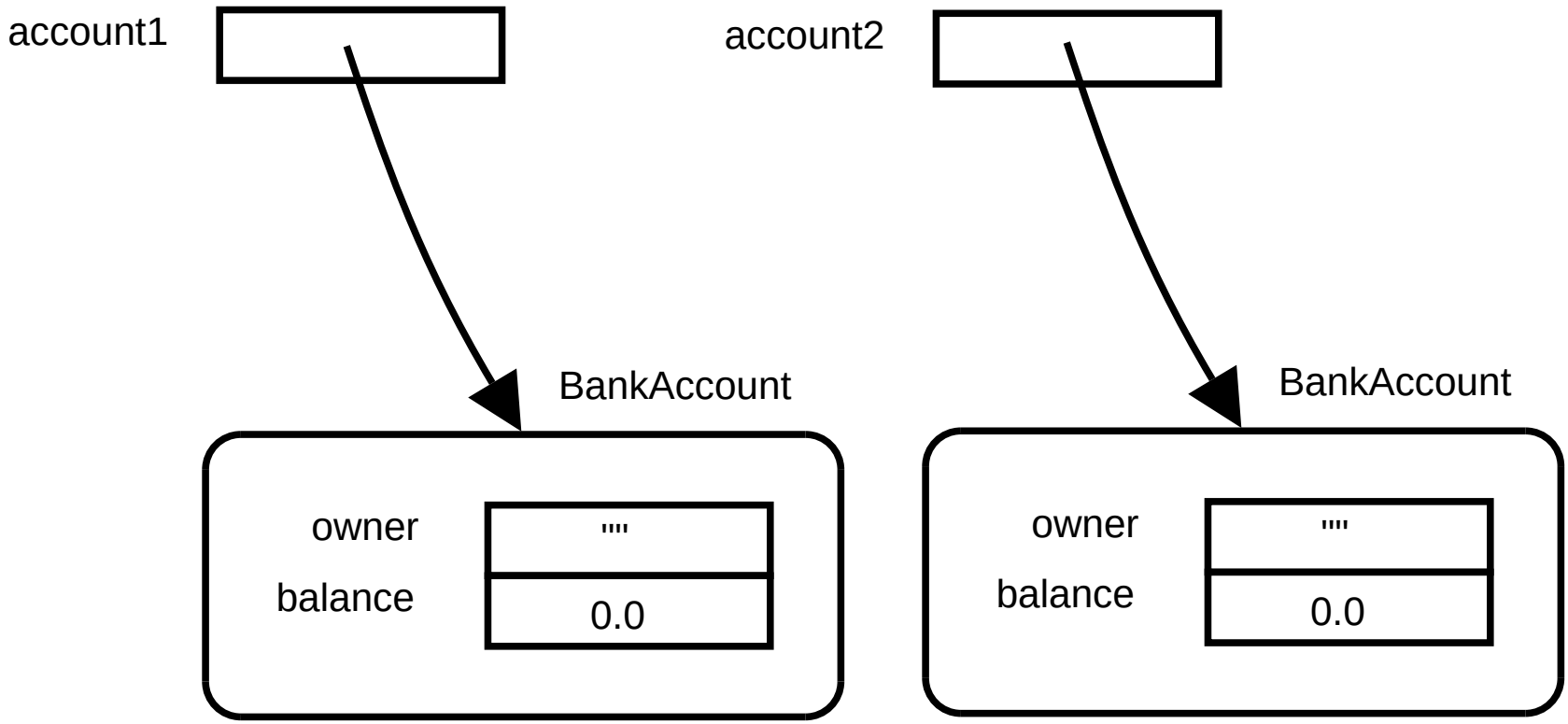
Objects and Classes



Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
        System.out.println("Done");
    }
}
```

Objects and Classes



Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

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

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

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

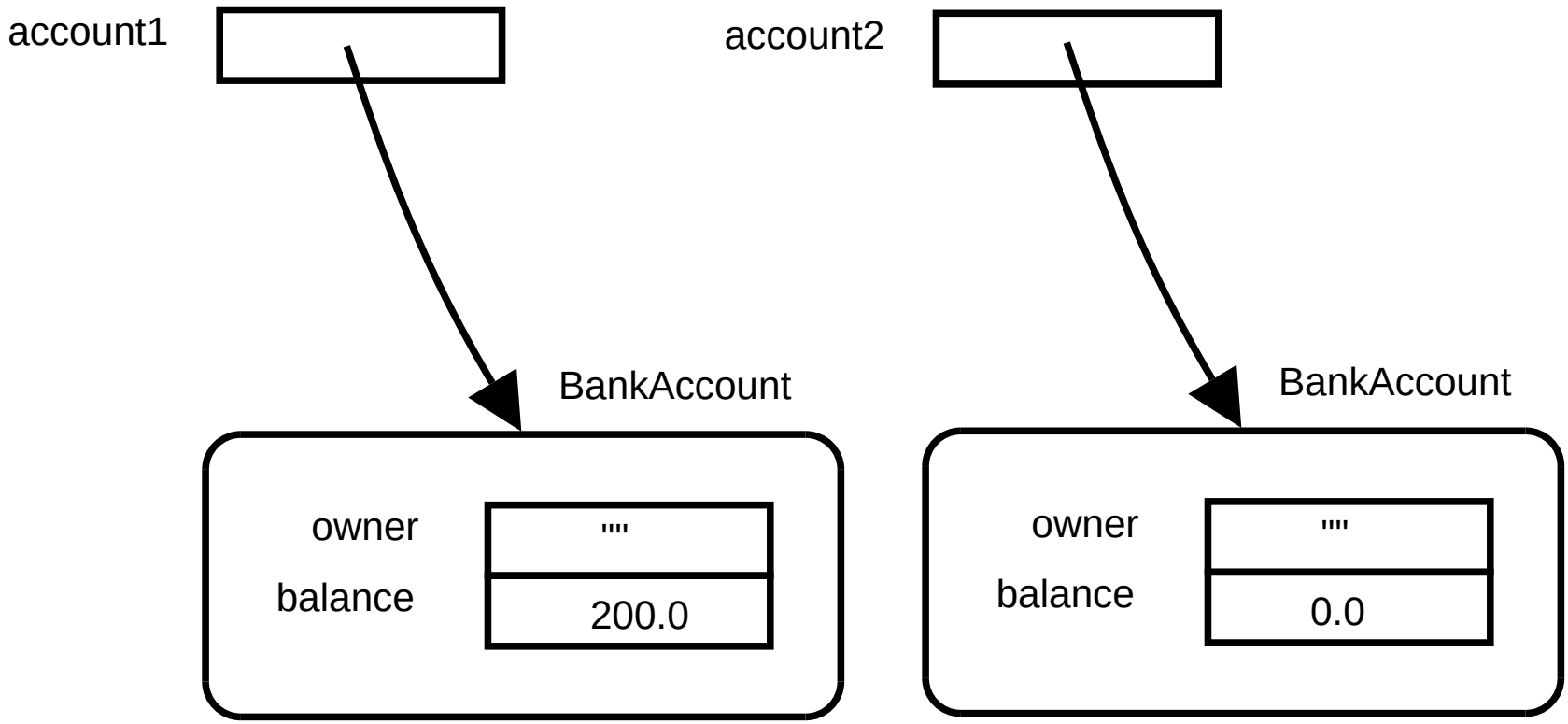
Objects and Classes

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

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

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

Objects and Classes



Objects and Classes

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

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

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

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
        System.out.println("Done");
    }
}
```

Objects and Classes

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

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

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

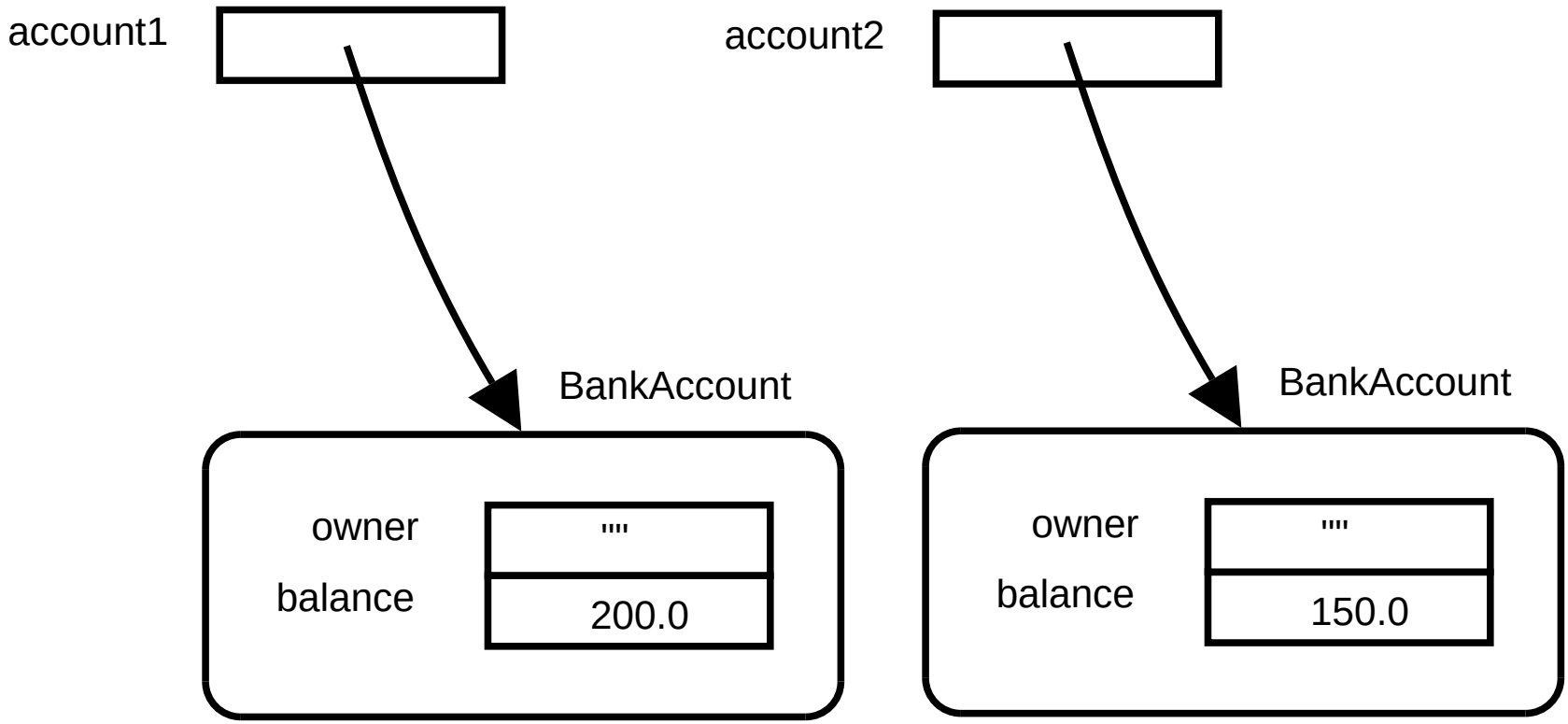
Objects and Classes

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

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

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

Objects and Classes



Objects and Classes

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

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

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

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        BankAccount account2;
        account1 = new BankAccount();
        account2 = new BankAccount();
        account1.deposit(200.0);
        account2.deposit(150.0);
        System.out.println("Done");
    }
}
```

Objects and classes

- Rules on using objects:
  - Before applying methods to an object, the object has to exist (it must be created)
  - If a method is applied to an object, then:
    * the method must be defined in the object's class
    * the number of arguments passed must be the same as the number of parameters expected
    * the types of arguments passed must match the types of the parameters, in the same order

Objects and classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1.deposit(200.0);
        System.out.println("Done");
    }
}
```

Objects and classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1.deposit(200.0);    // ERROR!
        System.out.println("Done");
    }
}
```

- Before applying methods to an object, the object has to exist (it must be created)

Objects and classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.payInterest(200.0);
        System.out.println("Done");
    }
}
```

Objects and classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.payInterest(200.0);  // ERROR
        System.out.println("Done");
    }
}
```

- If a method is applied to an object, it must be defined in the object's class

Objects and classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit();
        System.out.println("Done");
    }
}
```

Objects and classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit();  // ERROR
        System.out.println("Done");
    }
}
```

- If a method is applied to an object, it must have the same number of arguments as the number of expected parameters

Objects and classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(true);
        System.out.println("Done");
    }
}
```

Objects and classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(true);  // ERROR
        System.out.println("Done");
    }
}
```

- If a method is applied to an object, the types of the arguments must match the types of the parameters, in the right order

Direct access

- Normally we interact with an object through it's class' methods...
- so methods are responsible for updating an object's state (if necessary) by modifying its attributes
- ... but we can access the object's attributes directly (although it is not a good idea,) with the syntax:

  objectreference`.`*attribute*

  as long as `attribute` is defined in the object's class

Direct access

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.owner = ‘‘Albert’’;
        account1.deposit(200.0);
        System.out.print(‘‘’’+account1.owner);
        System.out.println(‘‘ has ’’+account1.balance);
    }
}
```

Direct access

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.owner = ‘‘Albert’’;
        account1.deposit(200.0);
        System.out.print(‘‘’’+account1.owner);
        System.out.println(‘‘ has ’’+account1.balance);
    }
}
```

Constructors

- The *constructor* of a class is a special method which is executed when a new instance of the class is created
- It has a special syntax

```
ClassName (parameters)
{
  // body
}
```

- that is, its name is the same as the class name, and
- it has no return type
- It is used to initialize the state of the object being created

Constructors

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

    BankAccount ()
    {
        owner = “No one”;
        balance = 0.0;
    }

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

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

Constructors

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        System.out.print("Done");
    }
}
```

Constructors

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount(“Jane”, 100.0);
        account1.deposit(200.0);
        System.out.print(“Done”);
    }
}
```

Constructors

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

    BankAccount (String who, double qty)
    {
        owner = who;
        balance = qty;
    }

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

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

The end