4. Defining Your Own Classes Part1 (1/2)

27 Aug 2015

Objectives

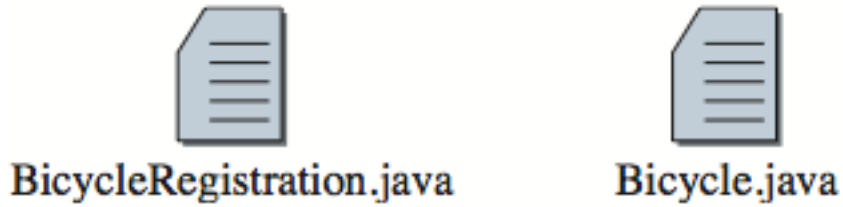
- Define a class with multiple methods and data members.
- Define a class with constructor.
- Pass both primitive data and objects to a method.
- Define and use value-returning methods.

1st Example:Defining and Using a Class

- Suppose we want to develop a program that tracks the bicycles by assigning an owner name.

1st Example:Defining and Using a Class

- For this sample program, we have created two classes: BicycleRegistration Class (Main Class) and Bicycle Class. So there are 2 source files for this program.



1st Example:Defining and Using a Class

- Bicycle Class (Programmer-Defined Class)

```
class Bicycle {

    // Data Member
    private String ownerName;

    //Constructor: Initialzes the data member
    public Bicycle( ) {
        ownerName = "Unknown";
    }

    //Returns the name of this bicycle's owner
    public String getOwnerName( ) {
        return ownerName;
    }

    //Assigns the name of this bicycle's owner
    public void setOwnerName(String name) {

        ownerName = name:
    }
}
```

1st Example:Defining and Using a Class

- BicycleRegistration Class (Main Class)

```
class BicycleRegistration {

    public static void main(String[] args) {

        Bicycle bike1, bike2;
        String owner1, owner2;

        bike1 = new Bicycle( );  //Create and assign values to bike1
        bike1.setOwnerName("Adam Smith");

        bike2 = new Bicycle( );  //Create and assign values to bike2
        bike2.setOwnerName("Ben Jones");

        //Output the information
        owner1 = bike1.getOwnerName( );
        owner2 = bike2.getOwnerName( );

        System.out.println(owner1 + " owns a bicycle.");
        System.out.println(owner2 + " also owns a bicycle.");
    }
}
```

1st Example:Defining and Using a Class

- The dependency diagram between the two classes is as follows:

1st Example:Defining and Using a Class

- Compile and run the BicycleRegistration class.

```
javac BicycleRegistration.java
```

```
java BicycleRegistration
```

- When this program is executed, we get the following output

```
Adam Smith owns a bicycle.
Ben Jones also owns a bicycle.
```

1st Example:Defining and Using a Class

- The use of the Bicycle class instead of the standard classes. We create a Bicycle object bike2 by calling the 'new' operator, and we assign the name of its owner by executing

```
Bicycle bike1, bike2;

...

bike2 = new Bicycle( );
bike2.setOwnerName("Ben Jones");
```

1st Example:Defining and Using a Class

- To get the name of the owner of bike2 and assign the returned value to a variable, we write

```
String owner2;
...
owner2 = bike2.getOwnerName();
```

1st Example:Defining and Using a Class

- Bicycle Class: Here's a template for the Bicycle class declaration

```
class Bicycle {
    //data members
    //methods
}
```

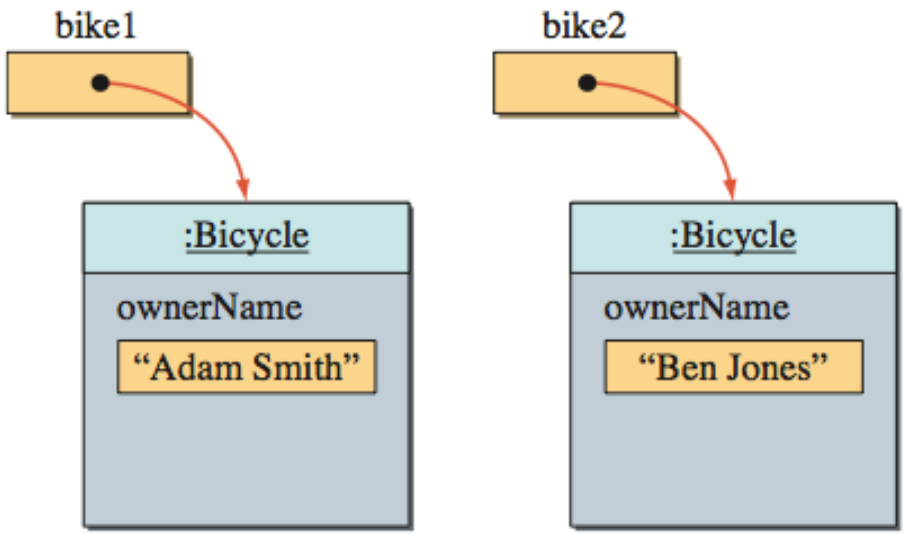
1st Example:Defining and Using a Class

- Data Member: Here's how we define the 'instance' data member 'ownerName' of the Bicycle class

```
class Bicycle {

    private String ownerName;

    //definitions for the constructor,
    //getOwnerName, and setOwnerName methods come here
}
```

1st Example:Defining and Using a Class

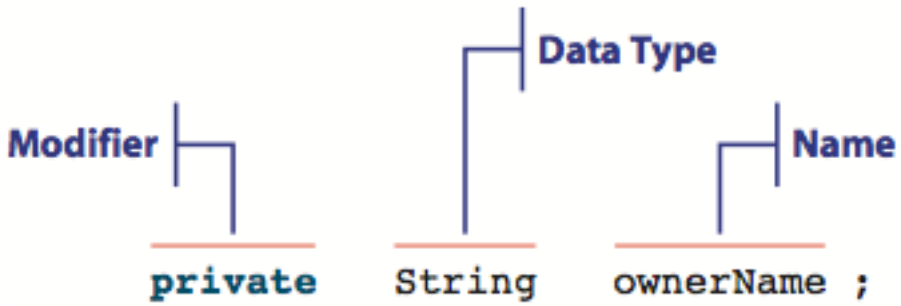
– We defined an instance variable as the data member, each instance of the class will have its own copy.



1st Example:Defining and Using a Class

– The syntax for the data member declaration is

```
<modifier-list> <data type> <name> ;
```



1st Example:Defining and Using a Class

- Methods: The syntax for defining a method is

```
<modifiers> <return type> <method name> ( <parameters> ) {
    <statements>
}
```

1st Example:Defining and Using a Class

– 3 Methods of Bicycle Class

Method	Parameter	Description
`Bicycle`	None	Initializes the owner's name to `Unassigned`.
`getOwnerName`	None	Returns the owner's name.
`setOwnerName`	Name of the owner (string)	Assigns the bicycle owner's name to the passed value.

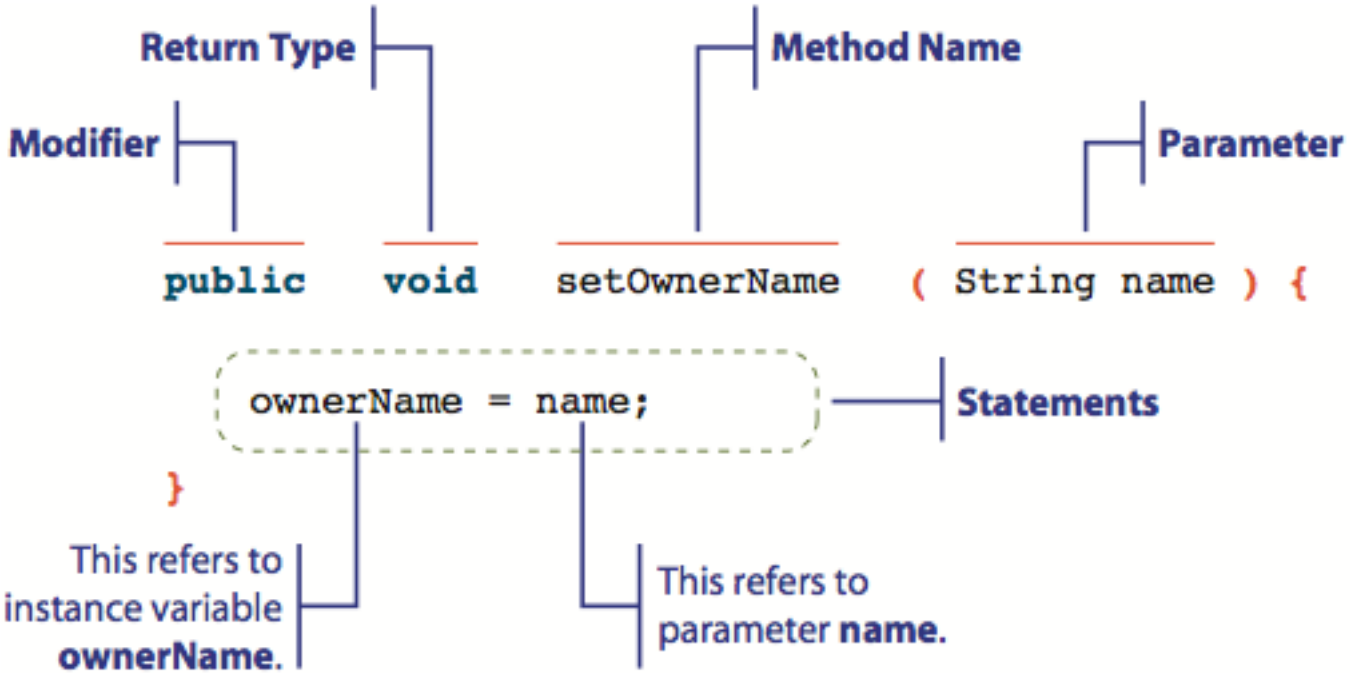
1st Example:Defining and Using a Class

1. setOwnerName() method is defined as follows:

```
public void setOwnerName(String name) {
    ownerName = name;
}
```

1st Example:Defining and Using a Class

– The components in the general syntax in setOwnerName() method:



1st Example:Defining and Using a Class

- A method that does not return a value, is declared as ‘void’. It is called a void method.
- setOwnerName() method is used to set value for data member ‘ownerName’, so this is void method.

- A method that sets a property of an object (such as sets owner of a bicycle) is called a ‘mutator’ or ‘setter’. The setOwnerName() method is a mutator.

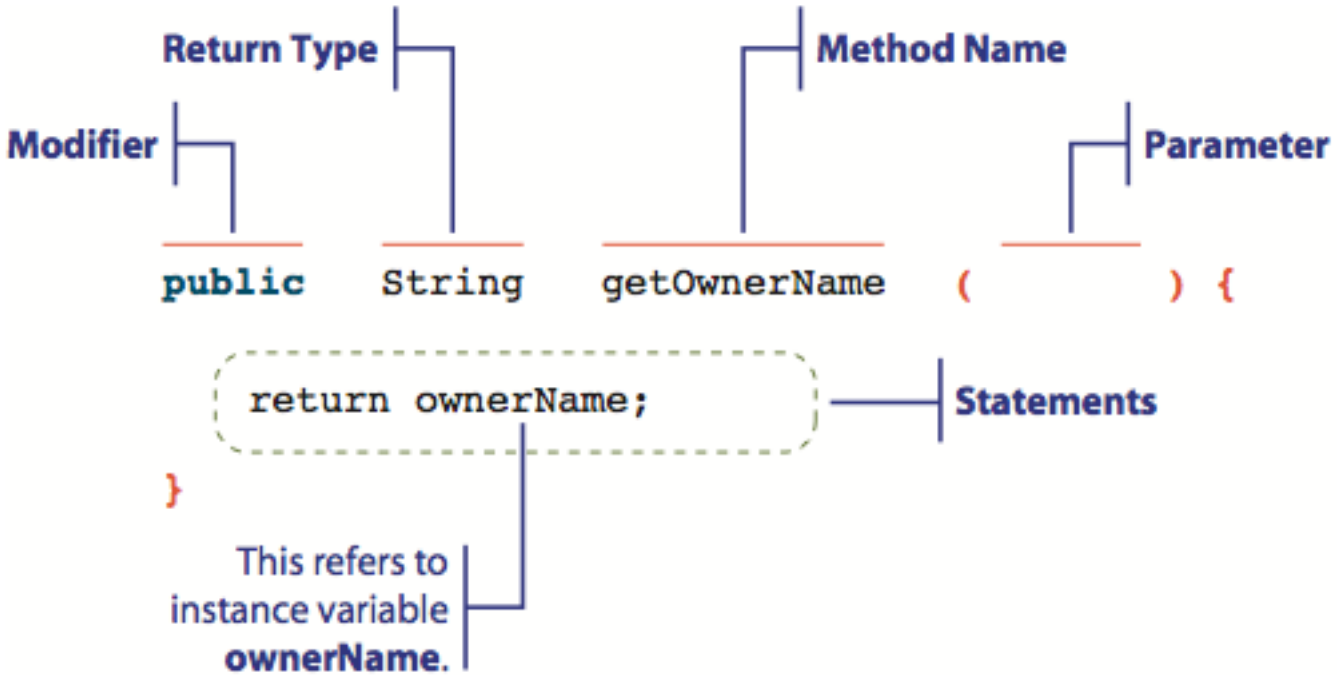
1st Example:Defining and Using a Class

2. getOwnerName() method is defined as follows:

```
public String getOwnerName( ) {
    return ownerName;
}
```

1st Example:Defining and Using a Class

– The components in the general syntax in getOwnerName() method:



1st Example:Defining and Using a Class

- A method that return a value is a value-returning method. When this method is called, it returns a value to the caller. A value returning method must include a return statement of the format

```
return <expression> ;
```

- getOwnerName() method returns a String value of instance variable 'ownerName', so its return type is declared as String.

1st Example:Defining and Using a Class

– A method that returns information about an object (such as who is the owner of a bicycle) is called an ‘accessor’ or ‘getter’. The getOwnerName() method is an accessor.

1st Example:Defining and Using a Class

3. Bicycle() method is a special method called a ‘constructor’ defined as follows:

```
public Bicycle( ) {
    ownerName = "Unassigned";
}
```

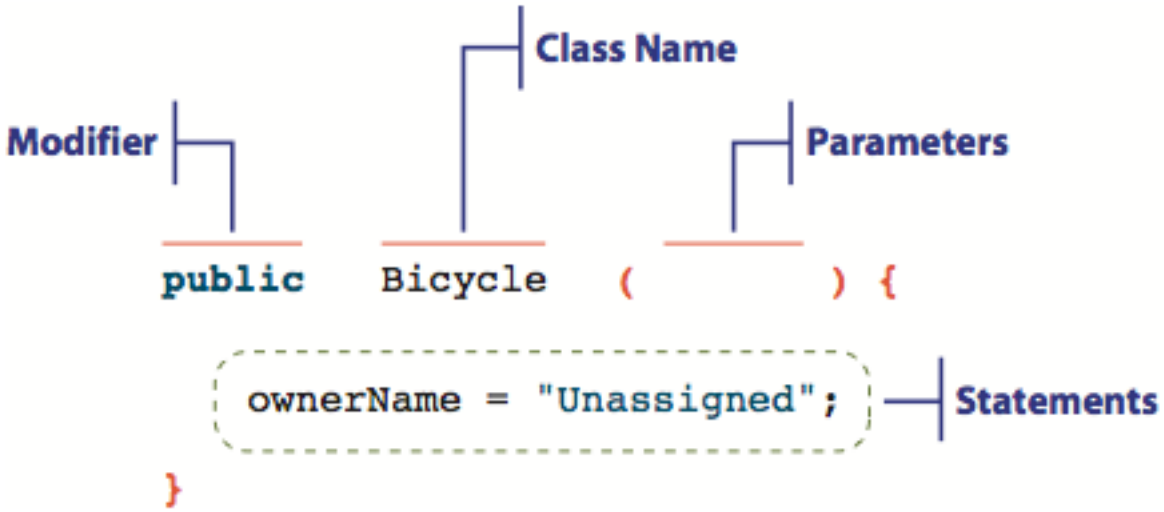
1st Example:Defining and Using a Class

– A constructor is a special method that is executed when a new instance of the class is created, that is, when the 'new' operator is called. It follows the general syntax

```
public <class name> ( <parameters> ) {
    <statements>
}
```

1st Example:Defining and Using a Class

– The components in the general syntax in Bicycle() constructor:



1st Example:Defining and Using a Class

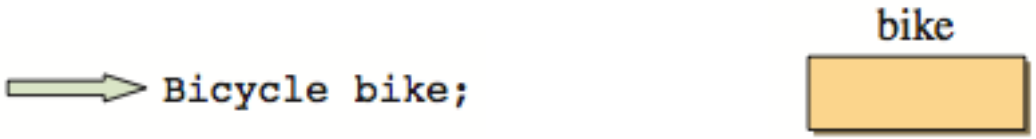
- Notice that a constructor does not have a return type and, consequently, will never include a return statement.

- The modifier of a constructor does not have to be public, but non-public constructors are rarely used.

- The purpose of the Bicycle() constructor is to initialize the data member of Bicycle Class.

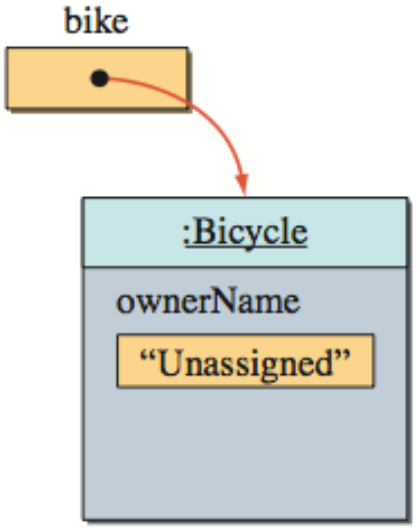
1st Example:Defining and Using a Class

– A sequence of state-of-memory diagrams after executing the constructor and the setOwnerName() method of the Bicycle class.

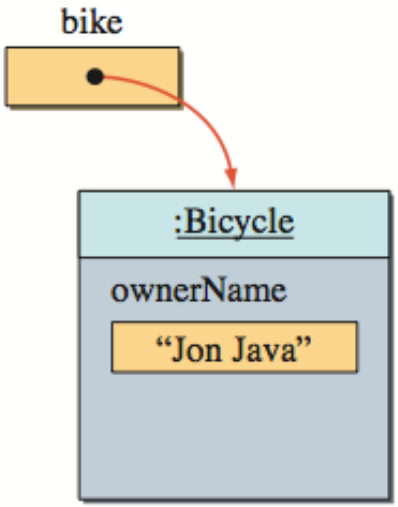


1st Example:Defining and Using a Class

```
Bicycle bike;

bike = new Bicycle( );
```

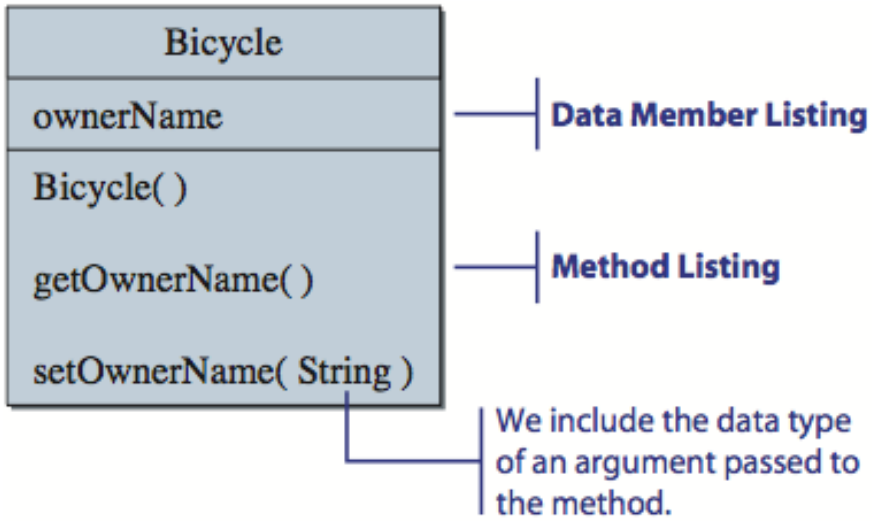



```
Bicycle bike;

bike = new Bicycle( );

bike.setOwnerName("Jon Java");
```



1st Example:Defining and Using a Class

– We stated earlier that the Bicycle class has 3 methods. However, a constructor is distinct from other "regular" methods, so it is more common to state that the Bicycle class has 1 constructor and 2 methods.



1st Example:Defining and Using a Class

– In listing the data members and methods of a class, we will use the following convention: data members first, then the constructor, and finally the methods.

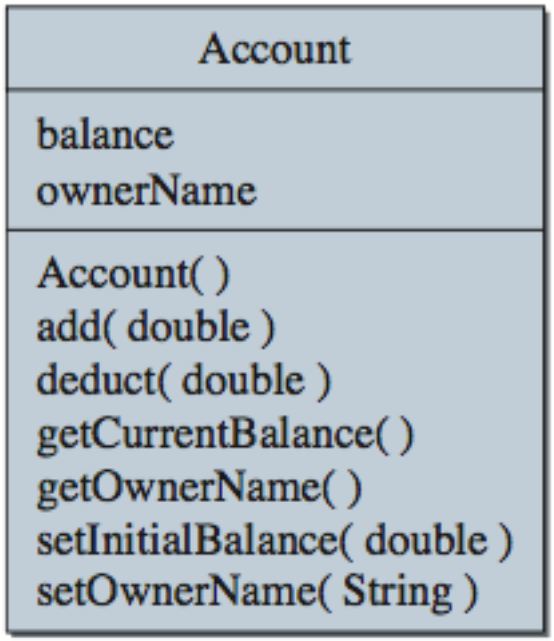
```
class <class name>  {

    // data members

    // constructor

    // methods

}
```

2nd Example:Defining and Using Multiple Classes

- The second sample program is composed of 3 classes (we are not counting the standard classes).

2nd Example:Defining and Using Multiple Classes

- In this example, we will define a new class named Account. An Account object has the owner name (String) and the balance (double).



2nd Example:Defining and Using Multiple Classes

- Account Class

```
class Account {

    // Data Members
    private String ownerName;

    private double balance;

    //Constructor
    public Account( ) {
        ownerName = "Unassigned";
        balance = 0.0;
    }
```

2nd Example:Defining and Using Multiple Classes

- Account Class (cont.)

```
//Adds the passed amount to the balance
public void add(double amt) {
    balance = balance + amt;
}

//Deducts the passed amount from the balance
public void deduct(double amt) {
    balance = balance - amt;
}
```

2nd Example:Defining and Using Multiple Classes

- Account Class (cont.)

```
//Returns the current balance of this account
public double getCurrentBalance( ) {
    return balance;
}

//Returns the name of this account's owner
public String getOwnerName( ) {

    return ownerName;
}
```

2nd Example:Defining and Using Multiple Classes

- Account Class (cont.)

```
    //Sets the initial balance of this account
    public void setInitialBalance(double bal) {
        balance = bal;
    }

    //Assigns the name of this account's owner
    public void setOwnerName(String name) {

        ownerName = name;
    }
}
```

2nd Example:Defining and Using Multiple Classes

- SecondMain Class (Main Class)

```
class SecondMain {
    //This sample program uses both the Bicycle and Account classes

    public static void main(String[] args) {

        Bicycle bike;
        Account acct;

        String  myName = "Jon Java";
```

2nd Example:Defining and Using Multiple Classes

- SecondMain Class (Main Class) (cont.)

```
bike = new Bicycle( );
bike.setOwnerName(myName);

acct = new Account( );
acct.setOwnerName(myName);
acct.setInitialBalance(250.00);

acct.add(25.00);
acct.deduct(50);
```

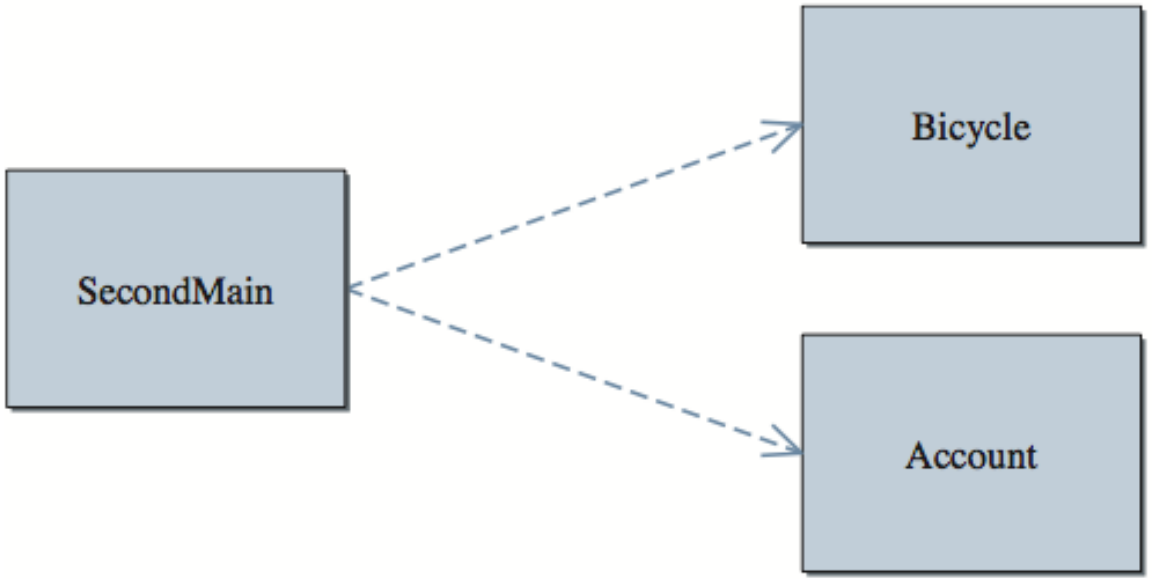
2nd Example:Defining and Using Multiple Classes

- SecondMain Class (Main Class) (cont.)

```
        //Output some information
        System.out.println(bike.getOwnerName() + " owns a bicycle and");
        System.out.println("has $ " + acct.getCurrentBalance() +
                                                    " left in the bank");

    }
}
```

2nd Example:Defining and Using Multiple Classes

- The dependency diagram between the two classes is as follows:



2nd Example:Defining and Using Multiple Classes

- Output:

```
Jon Java owns a bicycle and
has $ 225.0 left in the bank
```

Matching Arguments and Parameters

- Consider the Demo class that includes a ‘compute’ method. This method has three parameters—2 int and 1 double.

```
class Demo {
   ...

   public void compute(int i, int j, double x) {

      //method body
      //the actual statements in the body
      //are irrelevant to the discussion

   }

   ...
}
```

Matching Arguments and Parameters

- When we call the ‘compute’ method, we must pass three values. The values we pass must be assignment-compatible with the corresponding parameters.
  - For example, it is not okay to pass a double value to an int parameter.

Matching Arguments and Parameters

- Here are some valid calls from the main method:

```
class MyMain {

    public static void main(String[] arg) {

        Demo demo = new Demo();

        int i, k, m;

        i = 12;
        k = 10;
        m = 14;

        demo.compute(3, 4, 5.5);

        demo.compute(i, k, m);

        demo.compute(m, 20, 40);
    }
}
```

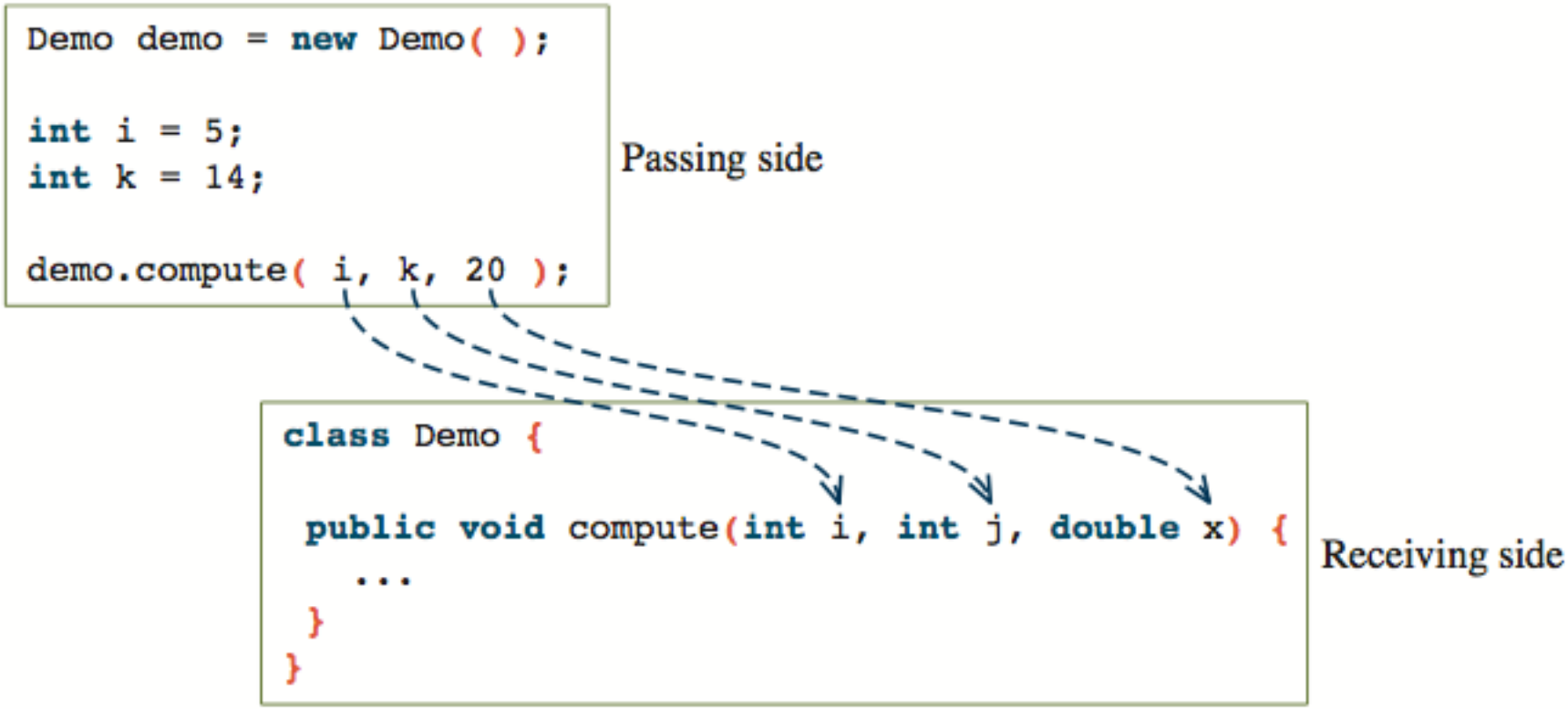
Matching Arguments and Parameters

- Argument: is a value we pass to a method, and the value is assigned to the corresponding parameters.

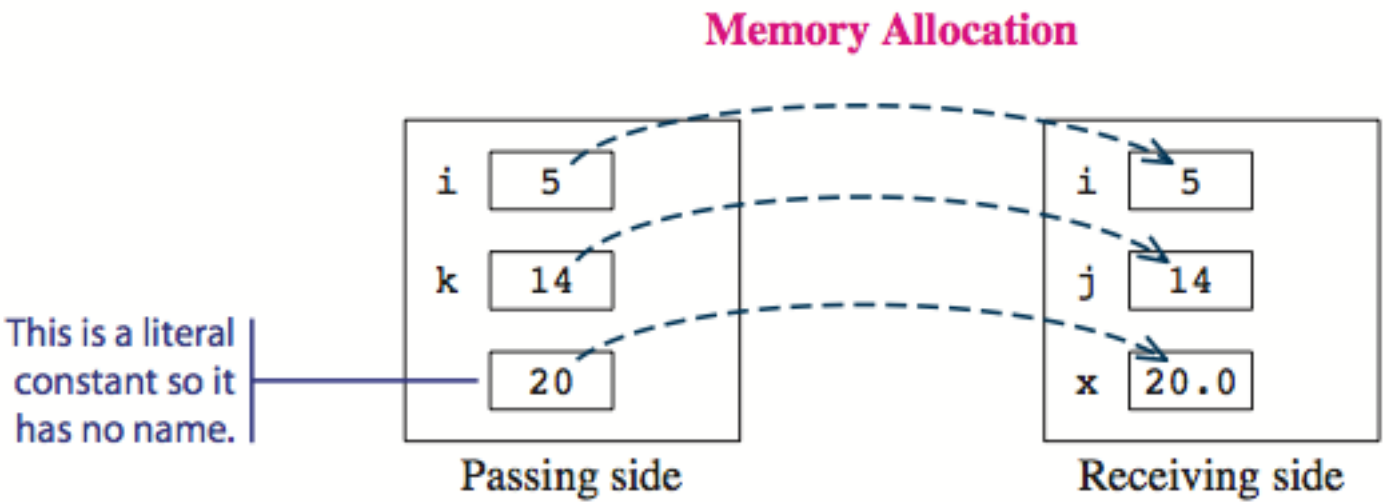
- Parameter: is a placeholder in the called method to hold the value of a passed argument.

Matching Arguments and Parameters

- Arguments and Parameters mapping

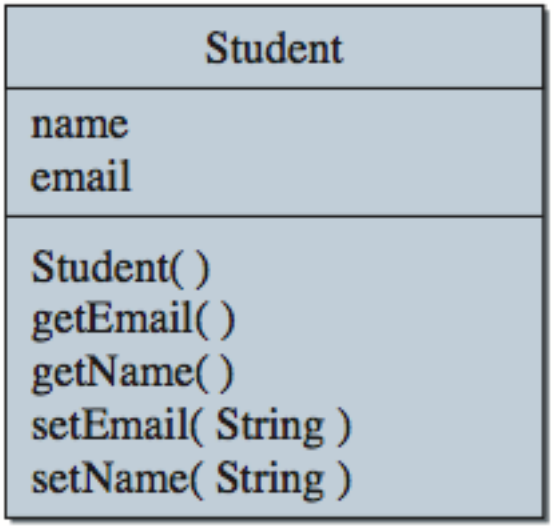


Matching Arguments and Parameters



Passing Objects to a Method

- In this section, we study how to pass an object when calling a method.
- First, we define the Student class. A Student object has a name (String) and an email (String).



Passing Objects to a Method

- Student Class

```
class Student {

    //Data Members
    private String name;

    private string email;

    //Constructor
    public Student( ) {
        name = "Unassigned";
        email = "Unassigned";
    }

    //Returns the email of this student
    public String getEmail( ) {

        return email;
    }
```

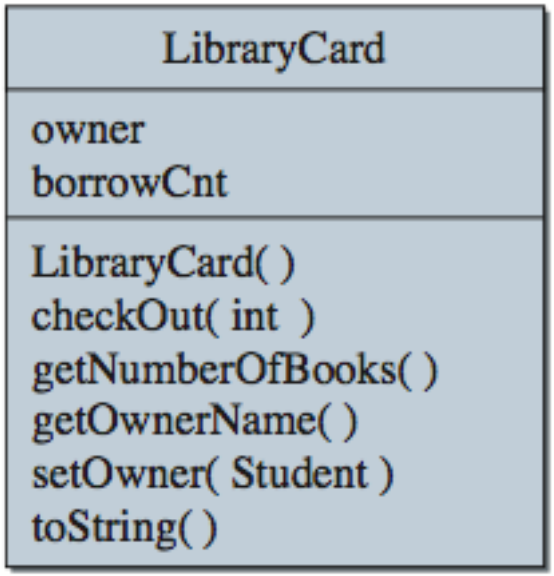
Passing Objects to a Method

- Student Class (cont.)

```
    //Returns the name of this student
    public String getName( ) {

        return name;
    }

    //Assigns the email of this student
    public void setEmail(String address) {

        email = address;
    }

    //Assigns the name of this student
    public void setName(String studentName) {

        name = studentName;
    }
}
```

Passing Objects to a Method

- Then we define the LibraryCard class. A LibraryCard object is owned by a Student, and it records the number of books being checked out.



Passing Objects to a Method

- LibraryCard Class

```
class LibraryCard {

    // Data Members

    //student owner of this card
    private Student owner;

    //number of books borrowed
    private int borrowCnt;

    //Constructor
    public LibraryCard( ) {
        owner = null;
        borrowCnt = 0;
    }
```

Passing Objects to a Method

- LibraryCard Class (cont.)

```
//numOfBooks are checked out
public void checkOut(int numOfBooks) {
    borrowCnt = borrowCnt + numOfBooks;
}

//Returns the number of books borrowed
public int getNumberOfBooks( ) {
    return borrowCnt;
}

//Returns the name of the owner of this card
public String getOwnerName( ) {
    return owner.getName( );
}
```

Passing Objects to a Method

- LibraryCard Class (cont.)

```
    //Sets owner of this card to student
    public void setOwner(Student student) {
        owner = student;
    }

    //Returns the string representation of this card
    public String toString( ) {
        return  "Owner Name:      " + owner.getName( ) + "\n" +
                "       Email:    " + owner.getEmail( ) + "\n" +
                "Books Borrowed: " + borrowCnt;
    }
}
```

Passing Objects to a Method

- In LibraryCard() constructor we initialize the data member ‘owner’ to null. The value of null means that ‘owner’ is pointing to no object.

```
//student owner of this card
private Student owner;

//number of books borrowed
private int borrowCnt;


//Constructor
public LibraryCard( ) {
    owner = null;
    borrowCnt = 0;
}
```

Passing Objects to a Method

- The setOwner() method must be called to assign a Student object. The method accepts a Student object as its parameter and sets the data member ‘owner’ to this Student object.

```
//Sets owner of this card to student
public void setOwner(Student student) {
    owner = student;
}
```

Passing Objects to a Method

- The getOwnerName() method returns the name of the owner. Because the data member owner refers to a Student object, we can get the name of this student by calling its getName() method. It is defined as

```
public String getOwnerName( ) {

   return owner.getName( );
}
```

Passing Objects to a Method

- The toString() method is a method that returns a string representation of an object.

```
//Returns the string representation of this card
public String toString( ) {
    return  "Owner Name:      " + owner.getName( ) + "\n" +
            "       Email:    " + owner.getEmail( ) + "\n" +
            "Books Borrowed: " + borrowCnt;
}
```

Passing Objects to a Method

- The power of passing an object to a method, for example, suppose a single student owns two library cards. Then we can make the data member owner of two LibraryCard objects to refer to the same Student object.

Passing Objects to a Method

- Librarian Class (Main Class)

```
class Librarian {

    public static void main(String[] args) {

        Student     student;
        LibraryCard card1, card2;

        student = new Student( );
        student.setName("Jon Java");
        student.setEmail("jj@javauniv.edu");
```

Passing Objects to a Method

- Librarian Class (cont.)

```
card1 = new LibraryCard( );
card1.setOwner(student);
card1.checkOut(3);

card2 = new LibraryCard( );
card2.setOwner(student); //the same student is the owner
                         //of the second card, too
```

Passing Objects to a Method

- Librarian Class (cont.)

```
        System.out.println("Card1 Info:");
        System.out.println(card1.toString() + "\n");

        System.out.println("Card2 Info:");
        System.out.println(card2.toString() + "\n");
    }
}
```

Passing Objects to a Method

- Output:

```
Card1 Info:
Owner Name: Jon Java
     Email: jj@javauniv.edu
Books Borrowed: 3

Card2 Info:
Owner Name: Jon Java
     Email: jj@javauniv.edu
Books Borrowed: 0
```

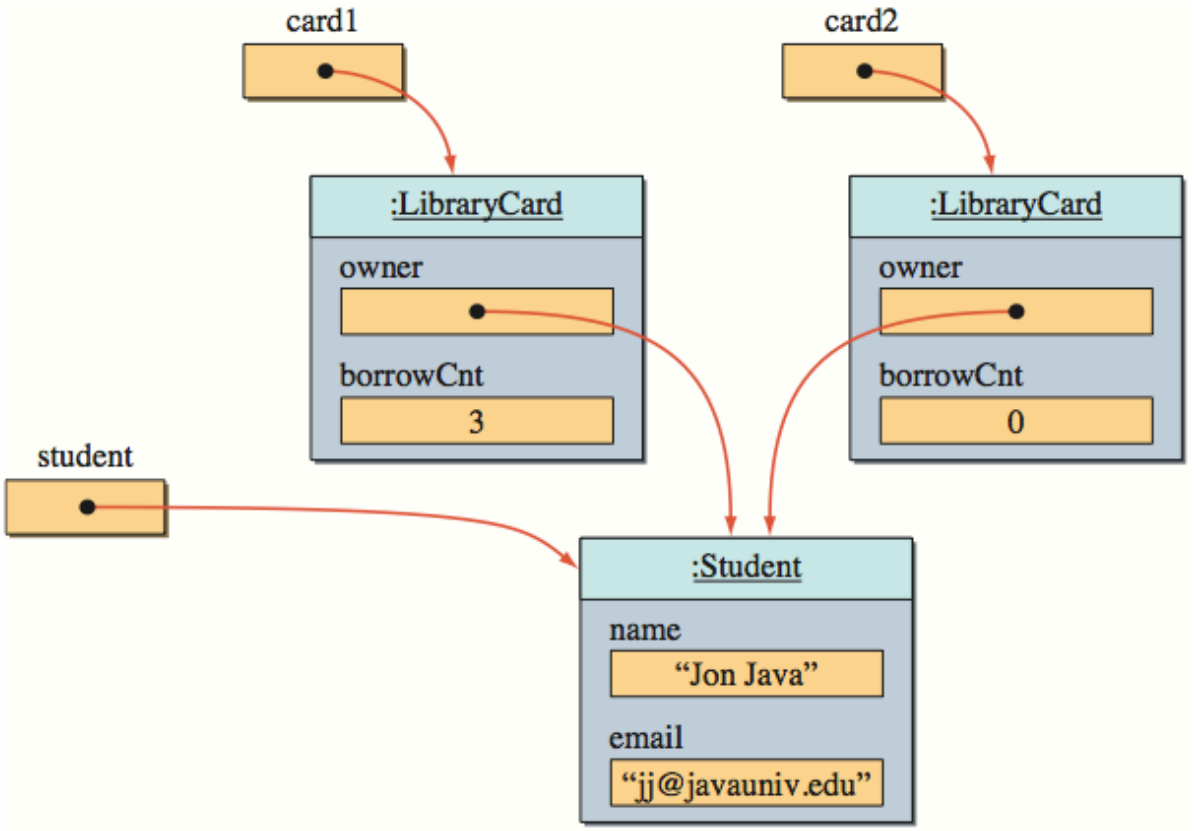
Passing Objects to a Method

- In this program, we create 1 Student object and 2 LibraryCard objects. For each of LibraryCard objects, we pass the same student to setOwner() methods:

```
card1.setOwner(student);
...
card2.setOwner(student);
```

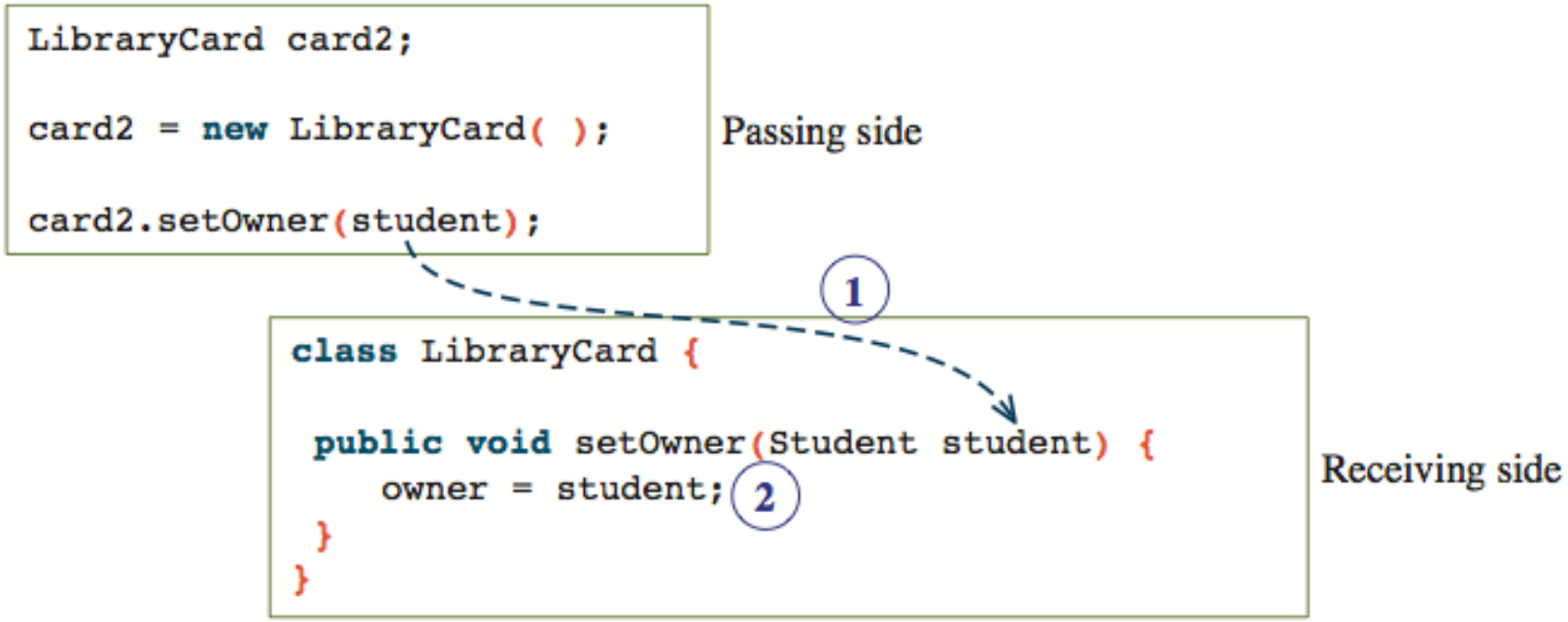
Passing Objects to a Method

- After the setOwner() method of card2 is called in the main() method



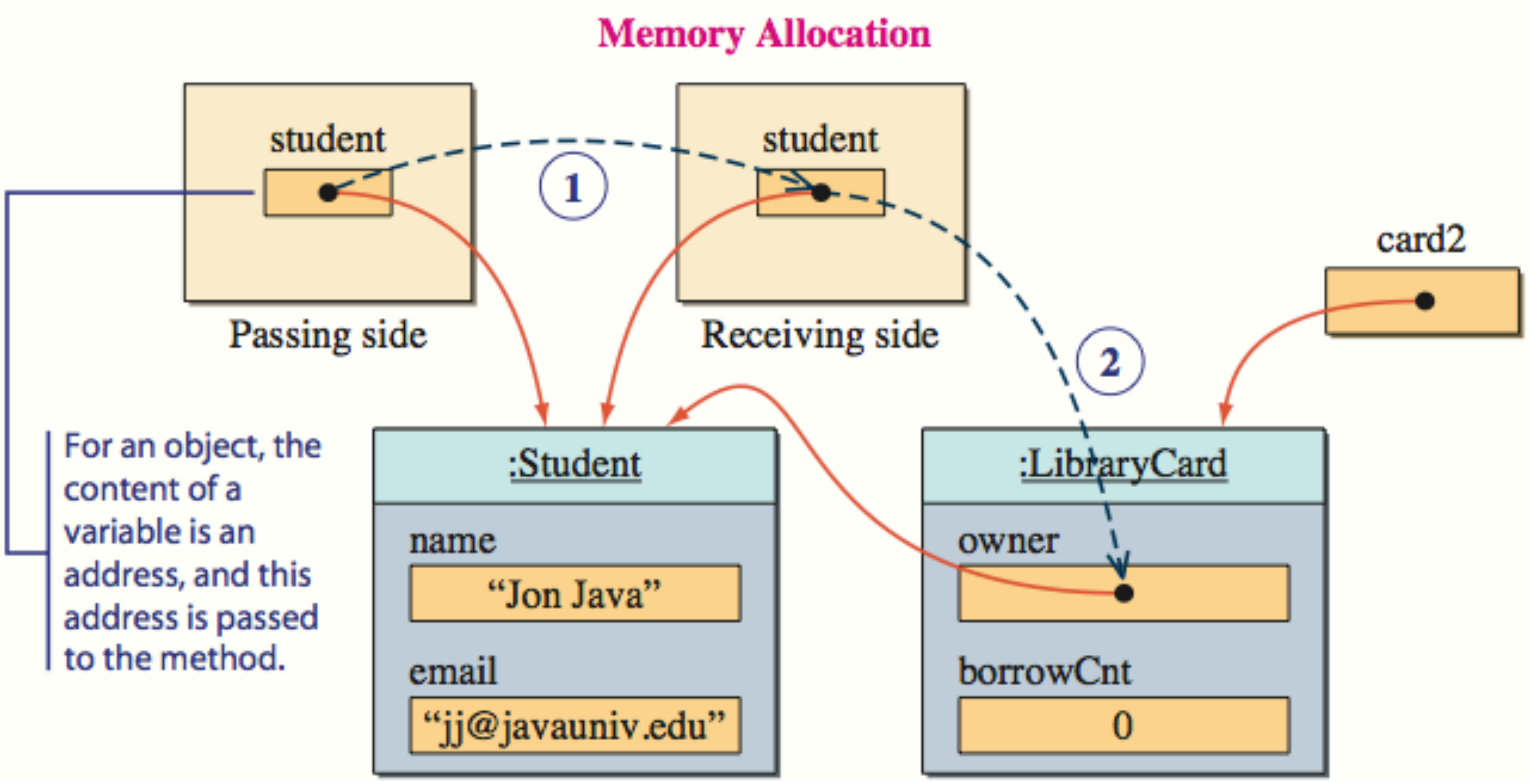
Passing Objects to a Method

- When we pass an object to a method, we are not sending a copy of an object, but rather a reference to the object.



Passing Objects to a Method

- An object is passed as an argument to a method.



Summary

- An object's properties are maintained by a set of data members.
- Methods define the behavior of an object.
- A method may or may not return a value. One that does not return a value is called a 'void method'.

Summary

- A constructor is a special method that is executed when a new object is created. Its purpose is to initialize the object into a valid state.
- A public method that changes a property of an object is called a ‘mutator’ or ‘setter’.
- A public method that retrieves a property of an object is called an ‘accessor’ or ‘getter’.

Reference

- C. Thomas Wu, An Introduction to Object-Oriented Programming with Java, 5th Edition
  - Chapter 4: Defining Your Own Classes-Part 1

Question?