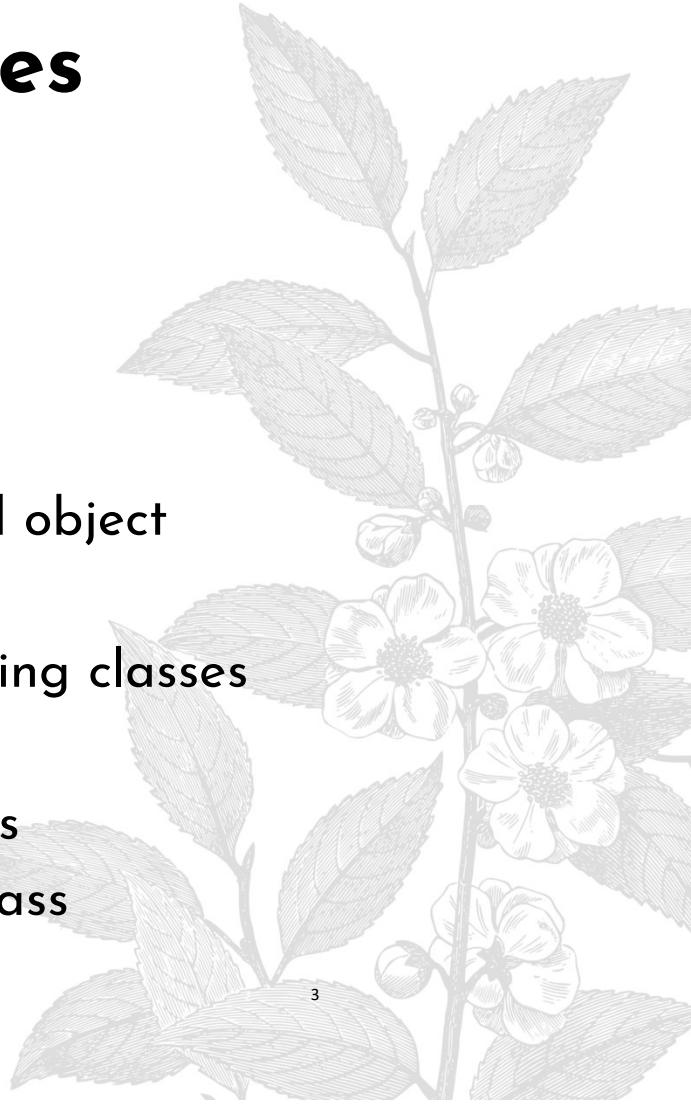
Objects and Classes

Chapter 2 - Objects and Classes

Chapter 2 – Objects and Classes

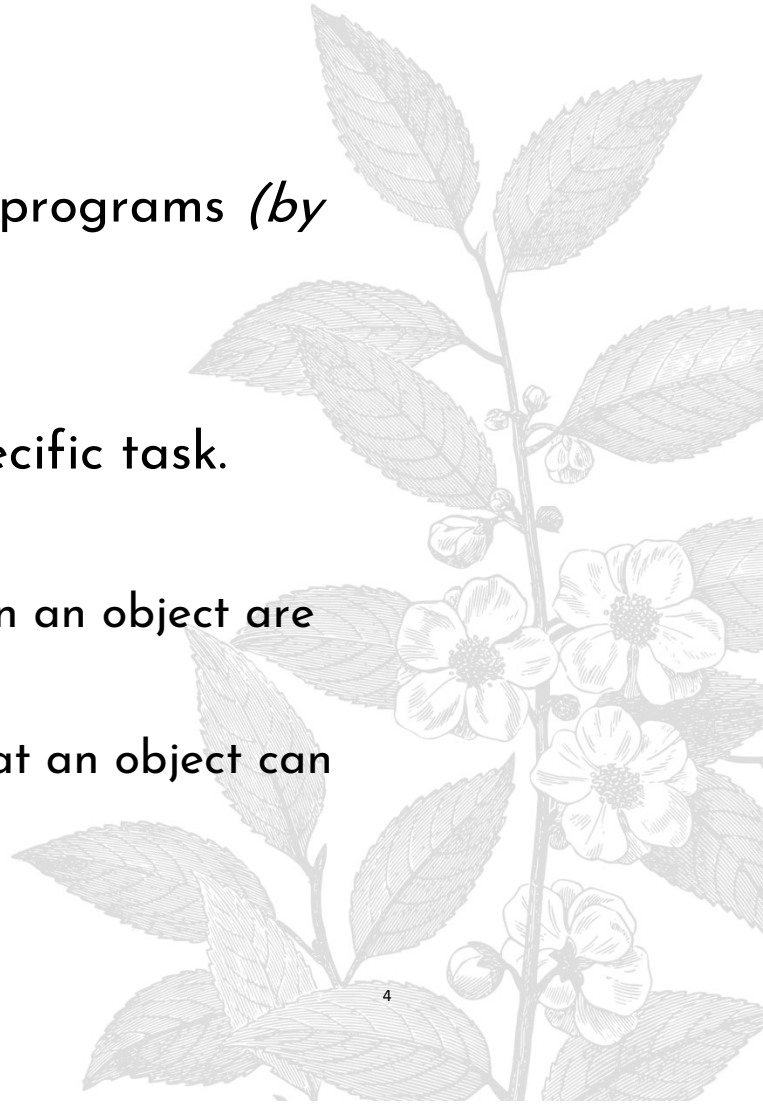
Chapter Goals

- ✓To understand the concepts of classes and objects
- ✓To be able to call methods
- ✓To learn about parameters and return values
- ✓To understand the difference between objects and object references
- ✓To become familiar with the process of implementing classes
- ✓To be able to implement simple methods
- ✓To understand the purpose and use of constructors
- ✓To understand how to use UML Diagrams for a class



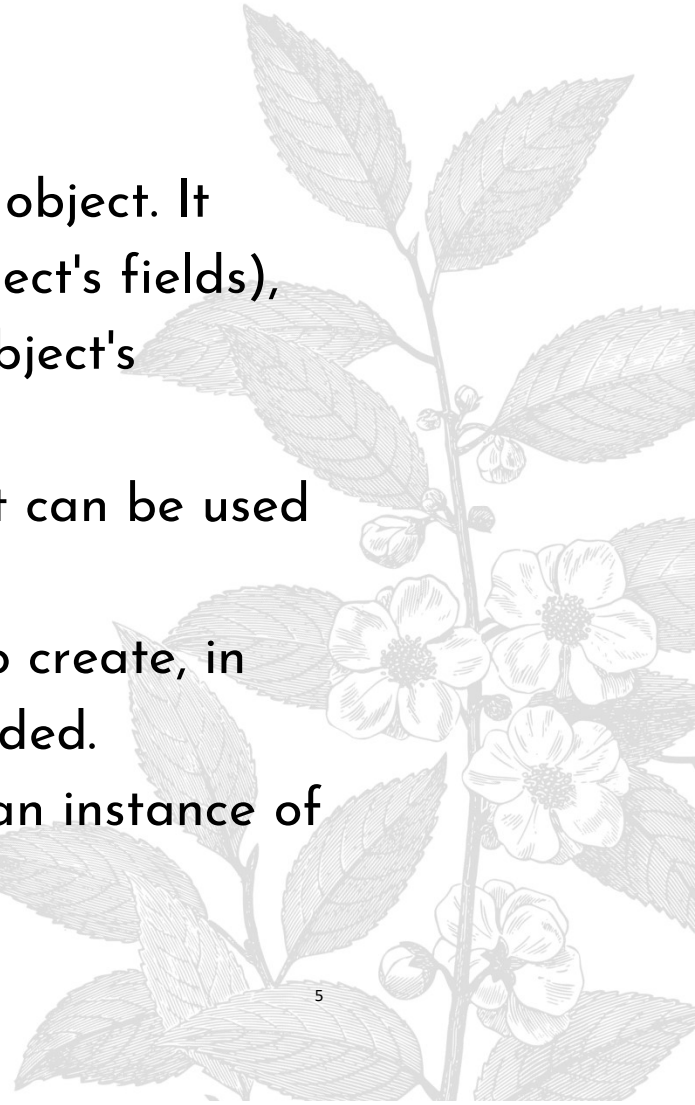
Objects and Classes

- ✓Object: entity that you can manipulate in your programs *(by calling methods)*
- ✓Each object belongs to a class
- ✓An object exists in memory, and performs a specific task.
- ✓Objects have two general capabilities:
  - Objects can store data. The pieces of data stored in an object are known as fields.
  - Objects can perform operations. The operations that an object can perform are known as methods.



Objects and Classes

- ✓A class is code that describes a particular type of object. It specifies the data that an object can hold (the object's fields), and the actions that an object can perform (the object's methods).
- ✓You can think of a class as a code "blueprint" that can be used to create a particular type of object.
- ✓When a program is running, it can use the class to create, in memory, as many objects of a specific type as needed.
- ✓Each object that is created from a class is called an instance of the class.



Objects and Classes

✓Example: System.out belongs to the class PrintStream

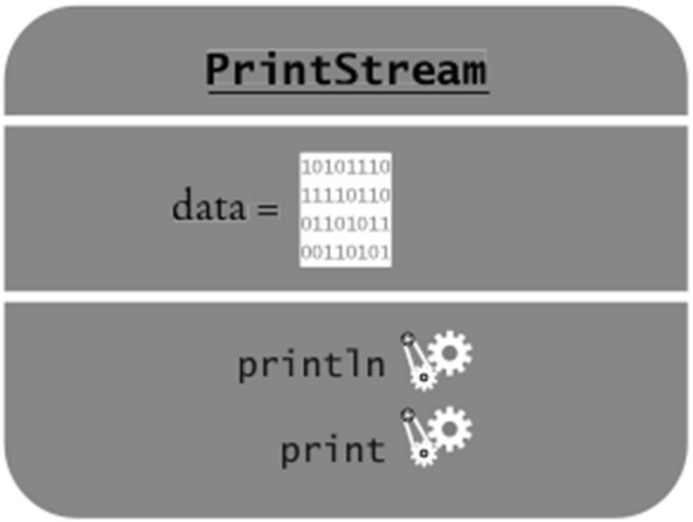


Figure 4 Representation of the `System.out` Object

A Representation of Two String Objects

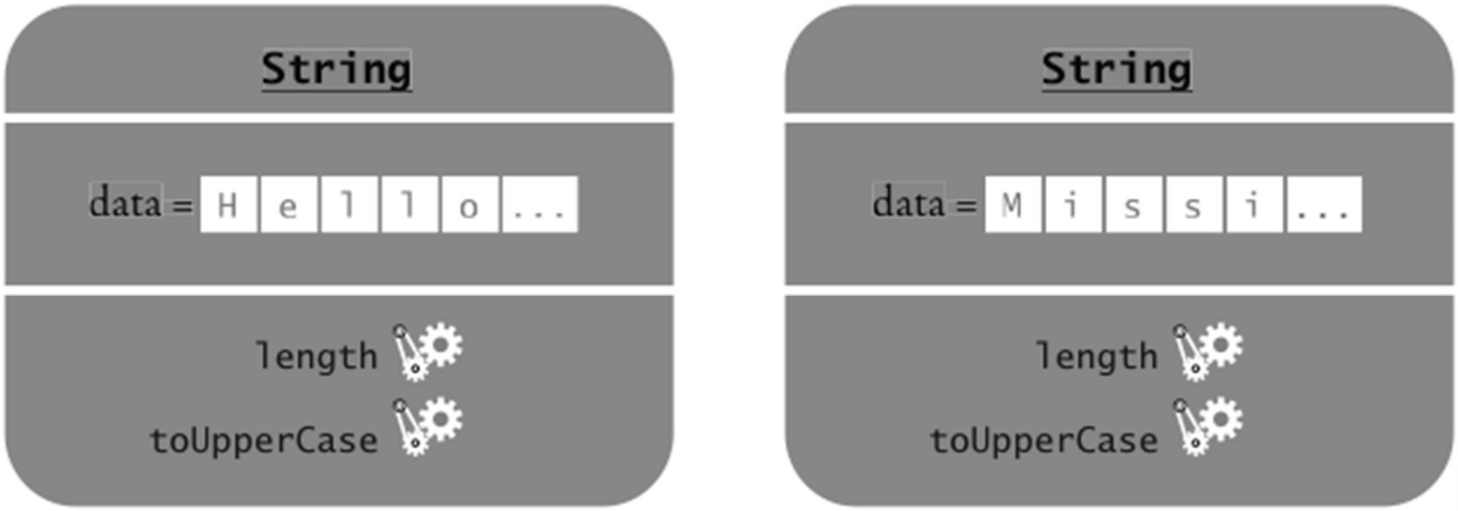
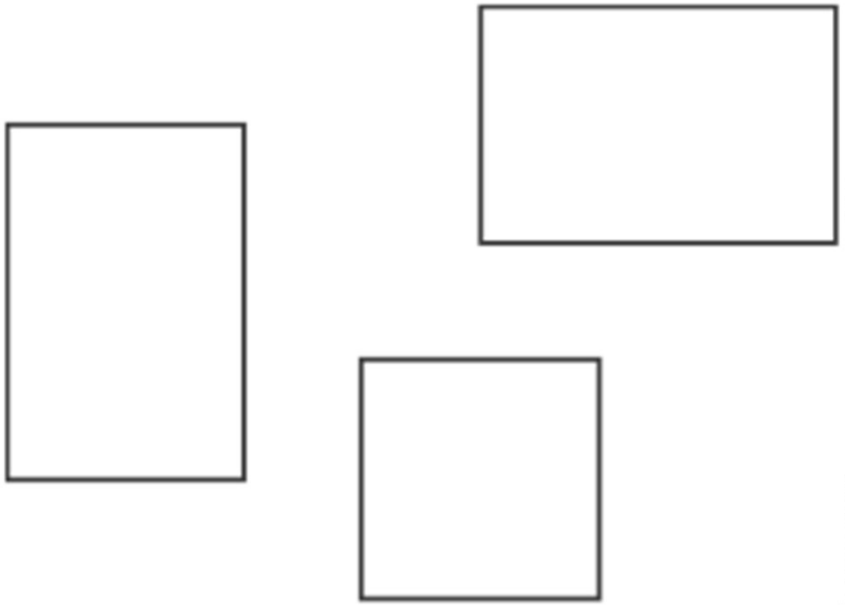


Figure 5 A Representation of Two String Objects

Rectangular Shapes and Rectangle Objects

Figure 10
Rectangular Shapes



Rectangular Shapes and Rectangle Objects

- ✓A Rectangle object isn't a rectangular shape – it is an object that contains a set of numbers that describe the rectangle:

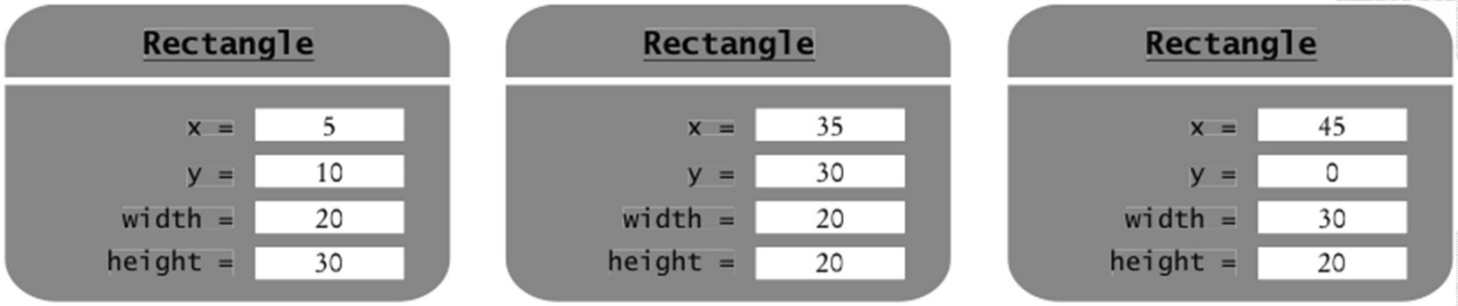
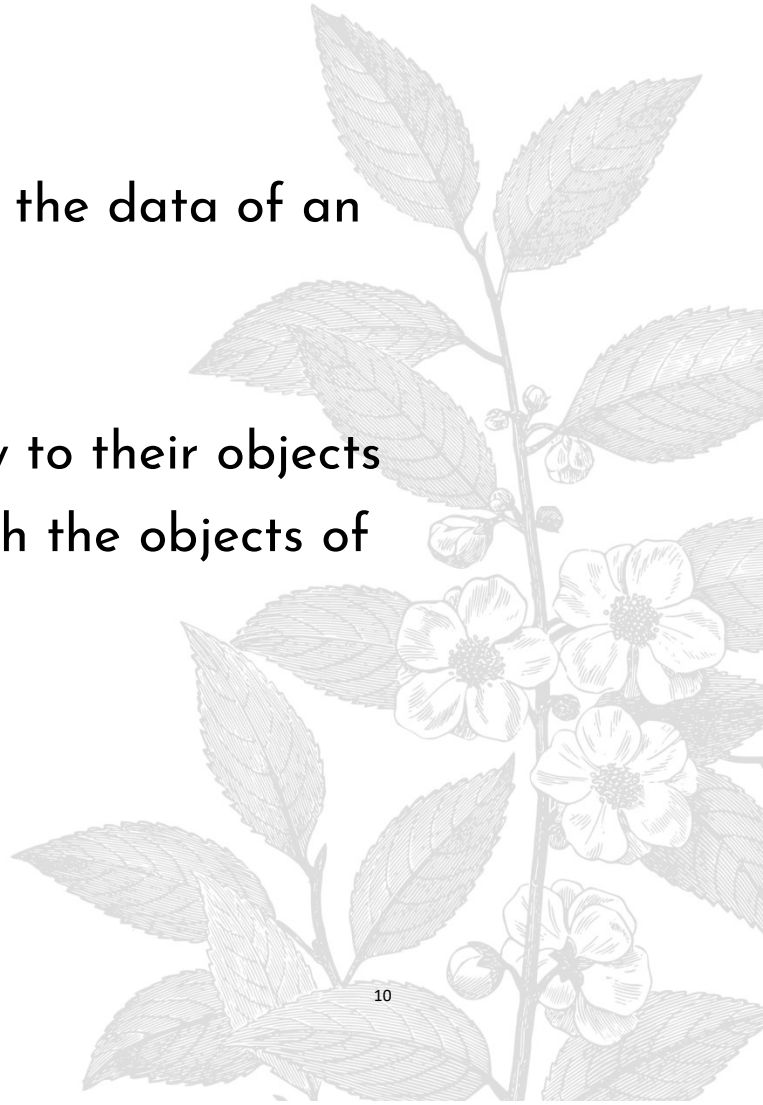


Figure 11 Rectangle Objects

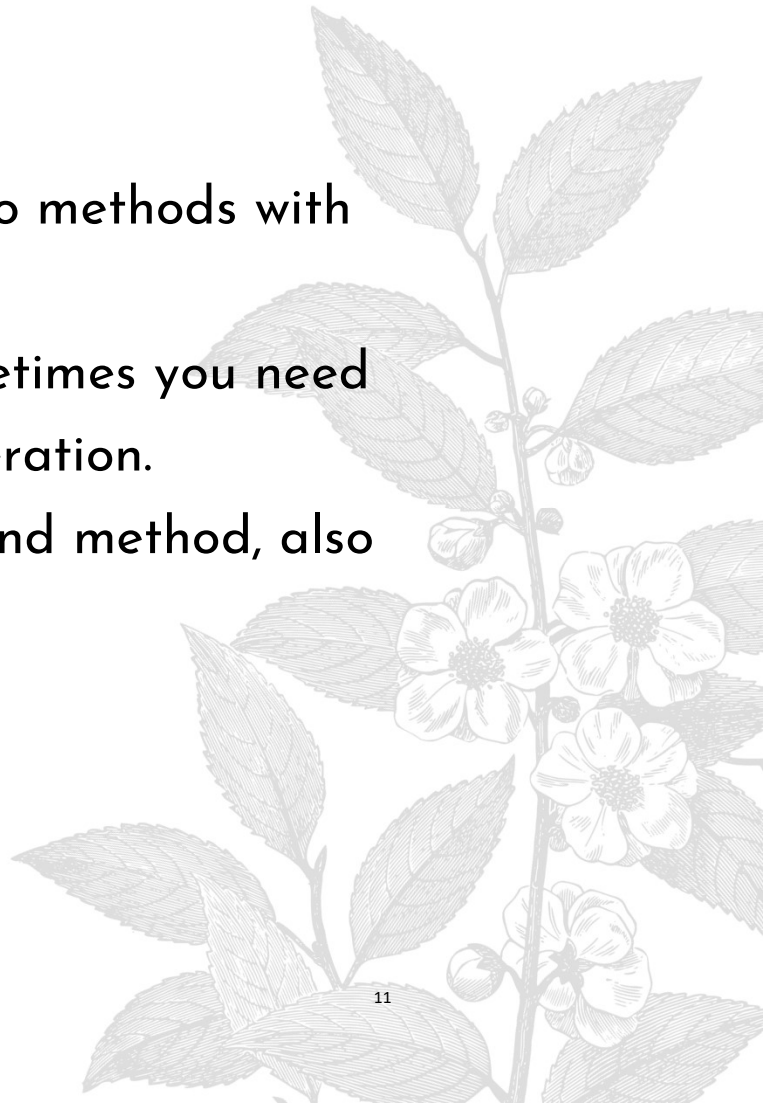
Methods

- ✓**Method**: sequence of instructions that accesses the data of an object
- ✓You manipulate objects by calling its methods
- ✓**Class**: declares the methods that you can apply to their objects
- ✓**Public Interface**: specifies what you can do with the objects of a class



Overloaded Method

- ✓**Overloaded method:** when a class declares two methods with the same name, but different parameters
- ✓Method overloading is important because sometimes you need several different ways to perform the same operation.
- ✓Example: the PrintStream class declares a second method, also called println, as public void println(int output)



Method Signature and Binding

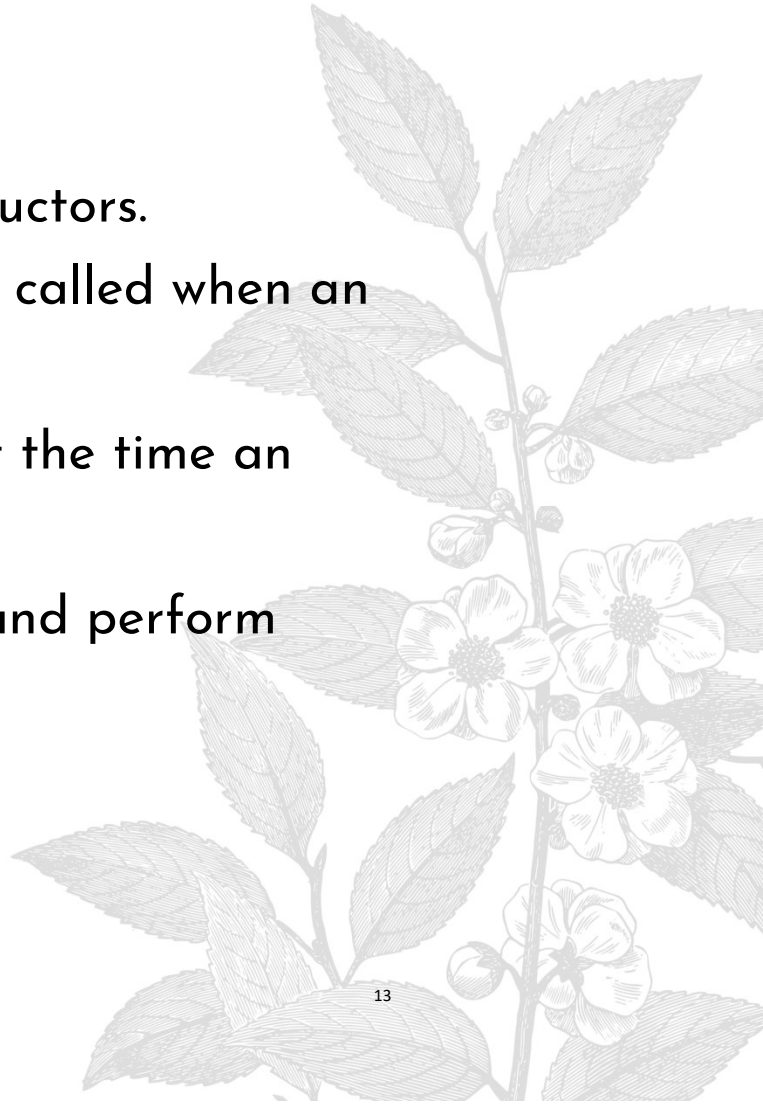
- ✓A method signature consists of the method's name and the data types of the method's parameters, in the order that they appear. The return type is not part of the signature.
- ✓Ex: Signatures of the add methods

  add(int, int)

  add(String, String)
- ✓The process of matching a method call with the correct method is known as binding. The compiler uses the method signature to determine which version of the overloaded method to bind the call to.

Constructors

- ✓Classes can have special methods called constructors.
- ✓A constructor is a method that is automatically called when an object is created.
- ✓Constructors are used to perform operations at the time an object is created.
- ✓Constructors typically initialize instance fields and perform other object initialization tasks.

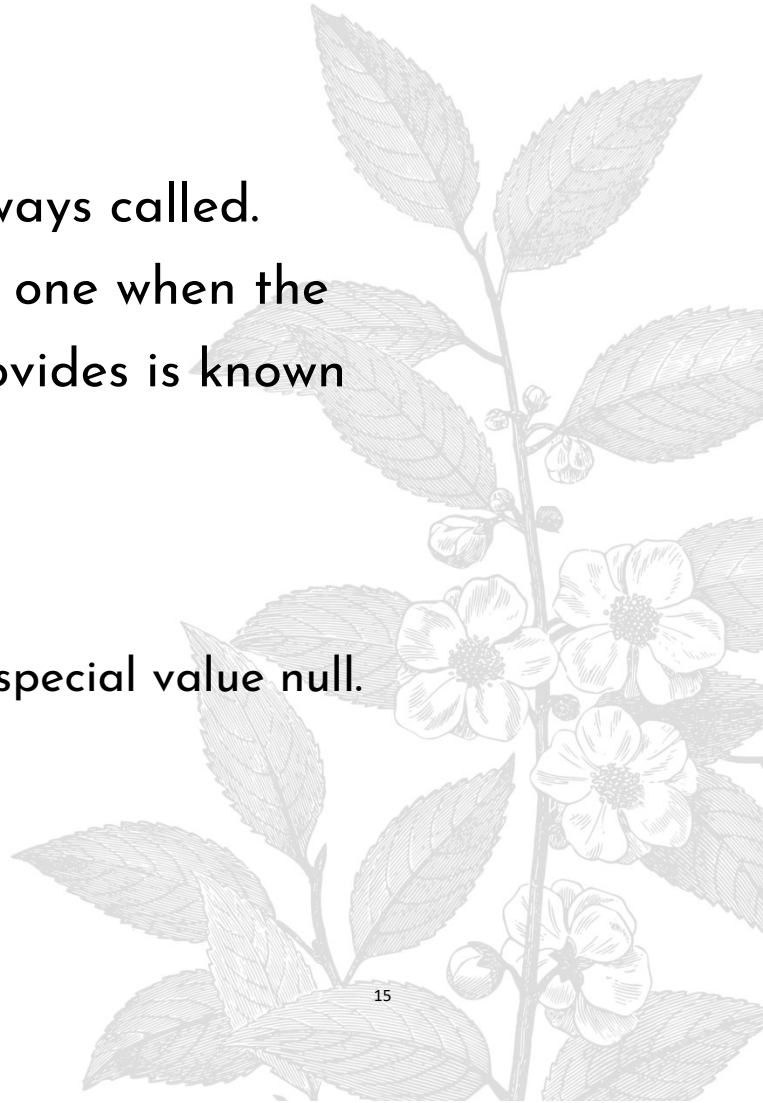


Constructors

- ✓Constructors have a few special properties that set them apart from normal methods.
  - Constructors have the same name as the class.
  - Constructors have no return type (not even void).
  - Constructors are typically public.

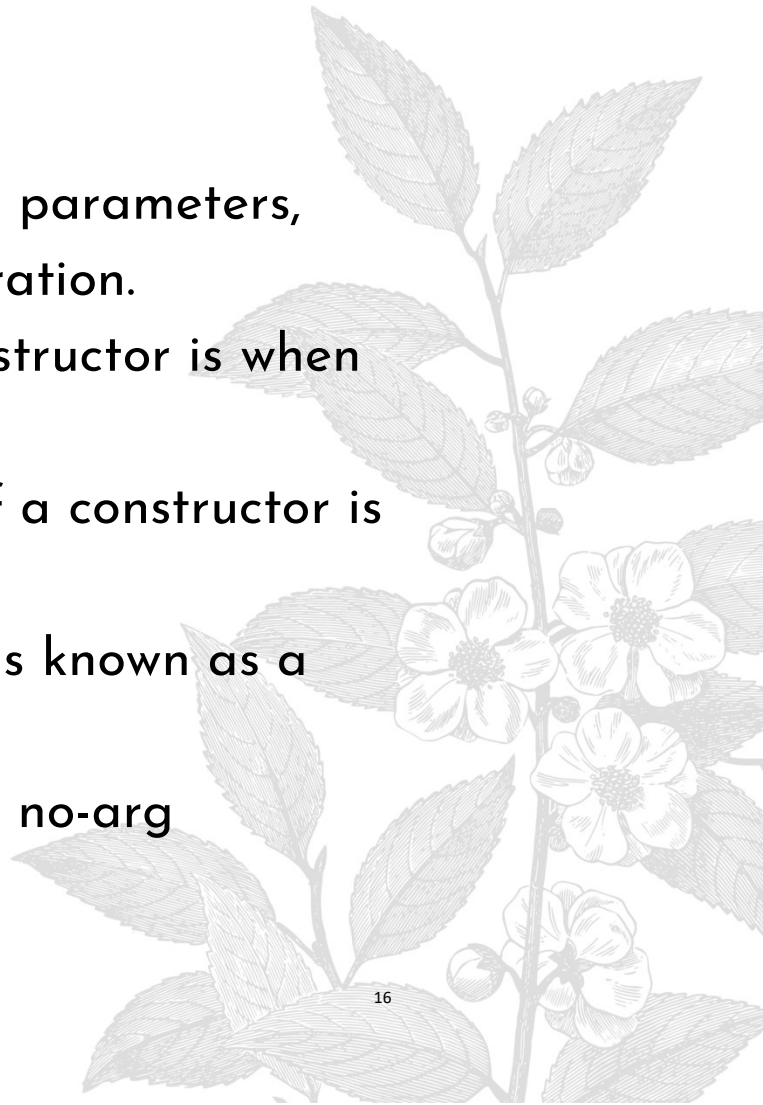
The Default Constructor

- ✓When an object is created, its constructor is always called.
- ✓If you do not write a constructor, Java provides one when the class is compiled. The constructor that Java provides is known as the default constructor.
  - It sets all of the object's numeric fields to 0.
  - It sets all of the object's boolean fields to false.
  - It sets all of the object's reference variables to the special value null.



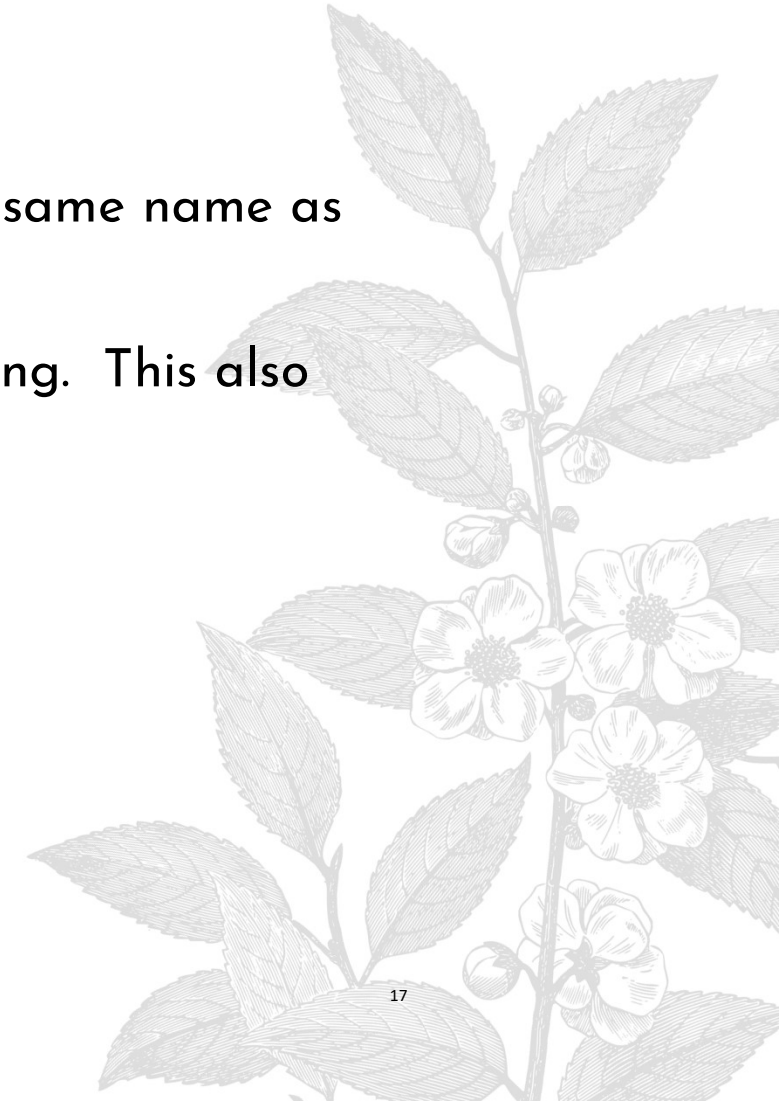
The Default Constructor

- ✓The default constructor is a constructor with no parameters, used to initialize an object in a default configuration.
- ✓The only time that Java provides a default constructor is when you do not write any constructor for a class.
- ✓A default constructor is not provided by Java if a constructor is already written.
- ✓A constructor that does not accept arguments is known as a no-arg constructor.
- ✓The default constructor (provided by Java) is a no-arg constructor.



Overloading Constructors

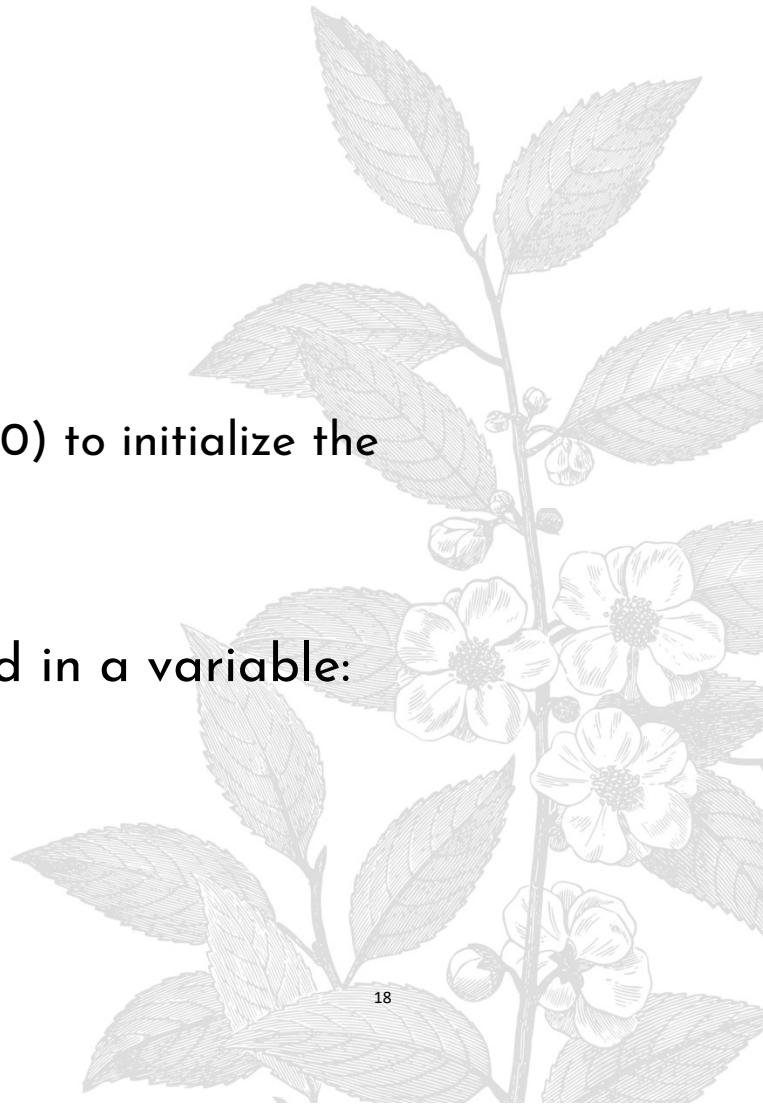
- ✓Two or more methods in a class may have the same name as long as their parameter lists are different.
- ✓When this occurs, it is called method overloading. This also applies to constructors.



Constructing Objects

- ✓new Rectangle(5, 10, 20, 30)
- ✓Detail:
  - The new operator makes a Rectangle object
  - It uses the parameters (in this case, 5, 10, 20, and 30) to initialize the data of the object
  - It returns the object
- ✓Usually the output of the new operator is stored in a variable:

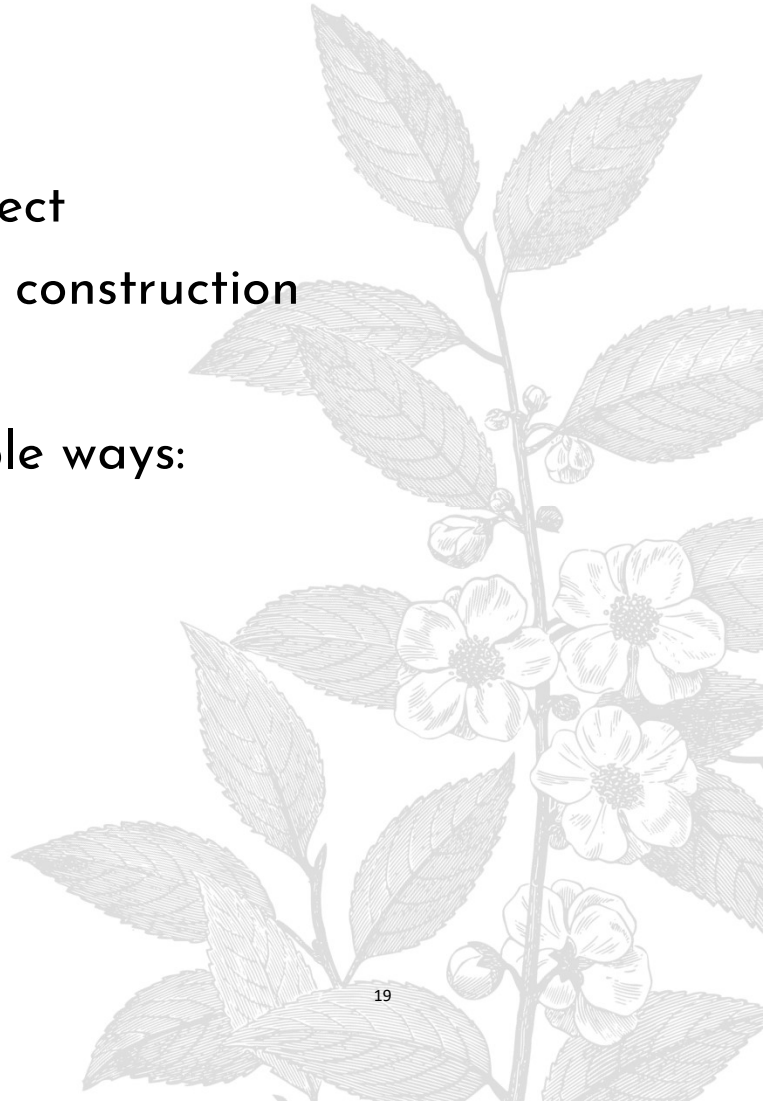
  Rectangle box = new Rectangle(5, 10, 20, 30);



Constructing Objects

- ✓Construction: the process of creating a new object
- ✓The four values 5, 10, 20, and 30 are called the construction parameters
- ✓Some classes let you construct objects in multiple ways:

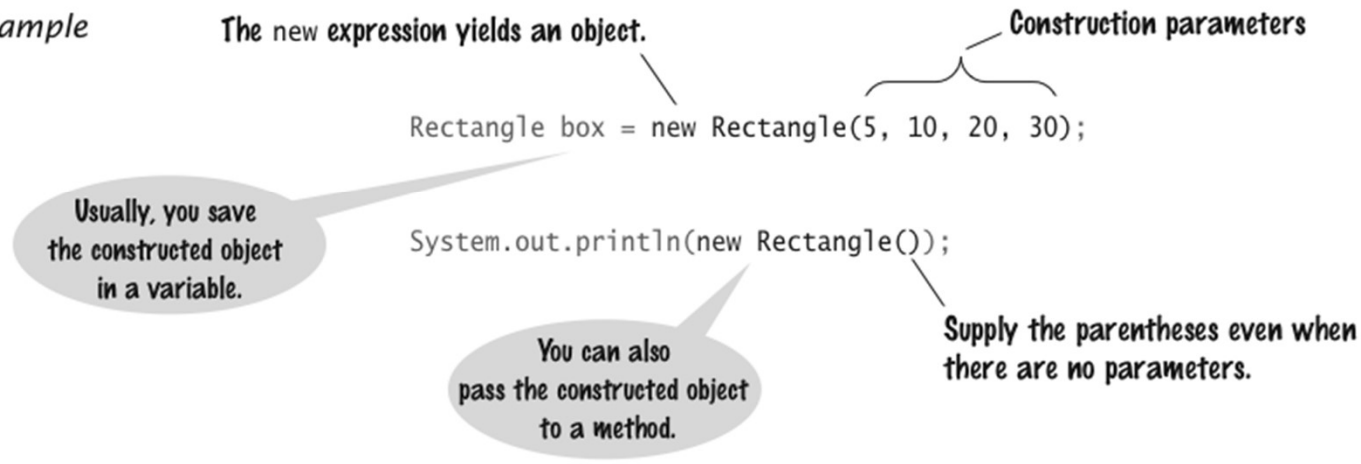
```
new Rectangle()
// constructs a rectangle with its top-left corner
// at the origin (0, 0), width 0, and height 0
```



Syntax 2.3 Object Construction

Syntax new *ClassName*(*parameters*)

Example

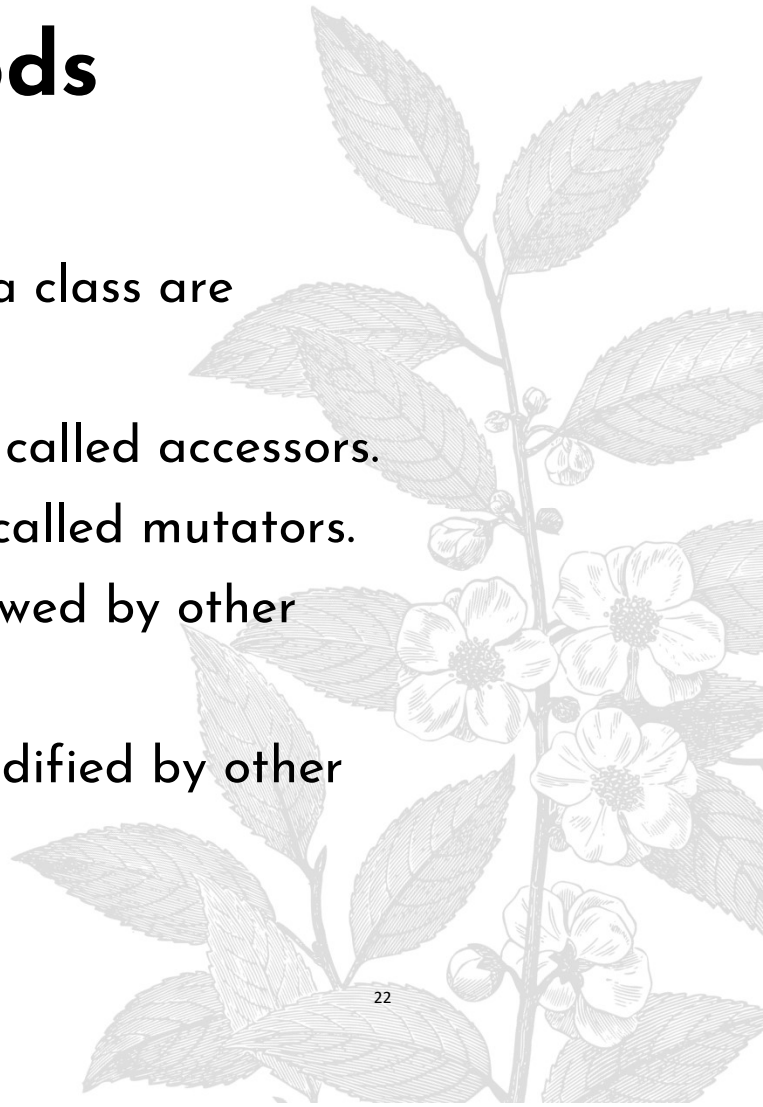


Access Specifiers

- ✓An access specifier is a Java keyword that indicates how a field or method can be accessed.
- ✓public
  - When the public access specifier is applied to a class member, the member can be accessed by code inside the class or outside.
- ✓private
  - When the private access specifier is applied to a class member, the member cannot be accessed by code outside the class. The member can be accessed only by methods that are members of the same class.

Accessor and Mutator Methods

- ✓A method used to control changes to a variable
- ✓Because of the concept of data hiding, fields in a class are private.
- ✓The methods that retrieve the data of fields are called accessors.
- ✓The methods that modify the data of fields are called mutators.
- ✓Each field that the programmer wishes to be viewed by other classes needs an accessor.
- ✓Each field that the programmer wishes to be modified by other classes needs a mutator.



Accessor and Mutator Methods

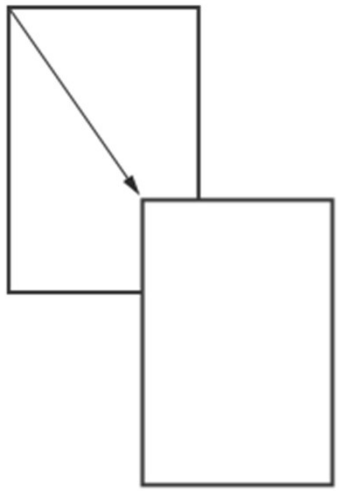
- ✓Accessor method: does not change the state of its implicit parameter:

  double width = box.getWidth();

- ✓Mutator method: changes the state of its implicit parameter:

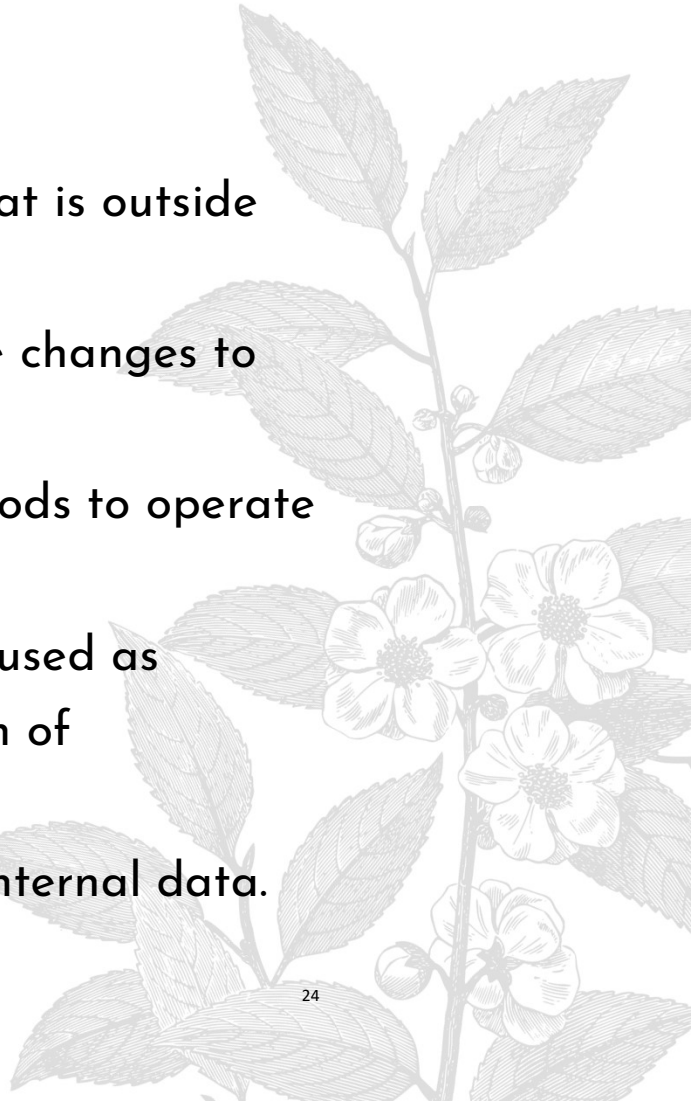
  box.translate(15, 25);

Figure 12
Using the `translate` Method to Move a Rectangle



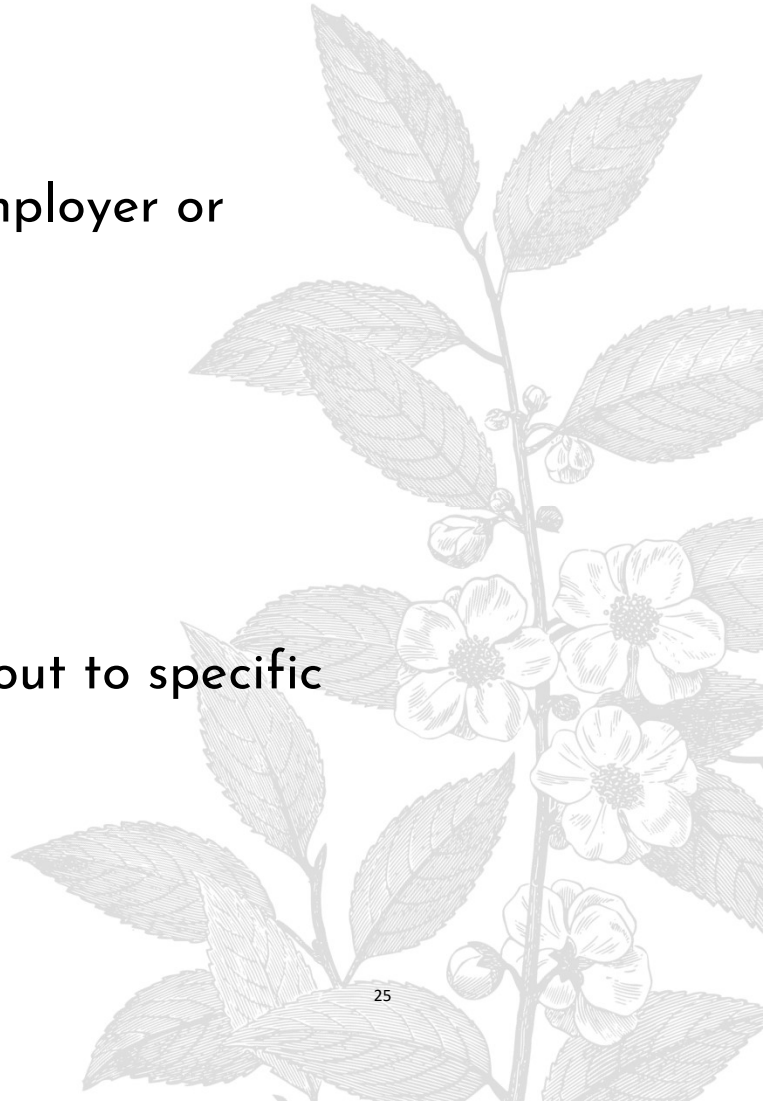
Data Hiding

- ✓An object hides its internal, private fields from code that is outside the class that the object is an instance of (the class).
- ✓Only the class's methods may directly access and make changes to the object's internal data.
- ✓Code outside the class must use the class's public methods to operate on an object's private fields.
- ✓Data hiding is important because classes are typically used as components in large software systems, involving a team of programmers.
- ✓Data hiding helps enforce the integrity of an object's internal data.



Class Layout Conventions

- ✓The layout of a source code file can vary by employer or instructor.
- ✓A common layout is:
  - Fields listed first
  - Methods listed second
  - Accessors and mutators are typically grouped.
- ✓There are tools that can help in formatting layout to specific standards.



Implementing a Test Program

- ✓Provide a tester class.
- ✓Supply a main method.
- ✓Inside the main method, construct one or more objects.
- ✓Apply methods to the objects.
- ✓Display the results of the method calls.
- ✓Display the values that you expect to get.

Object References

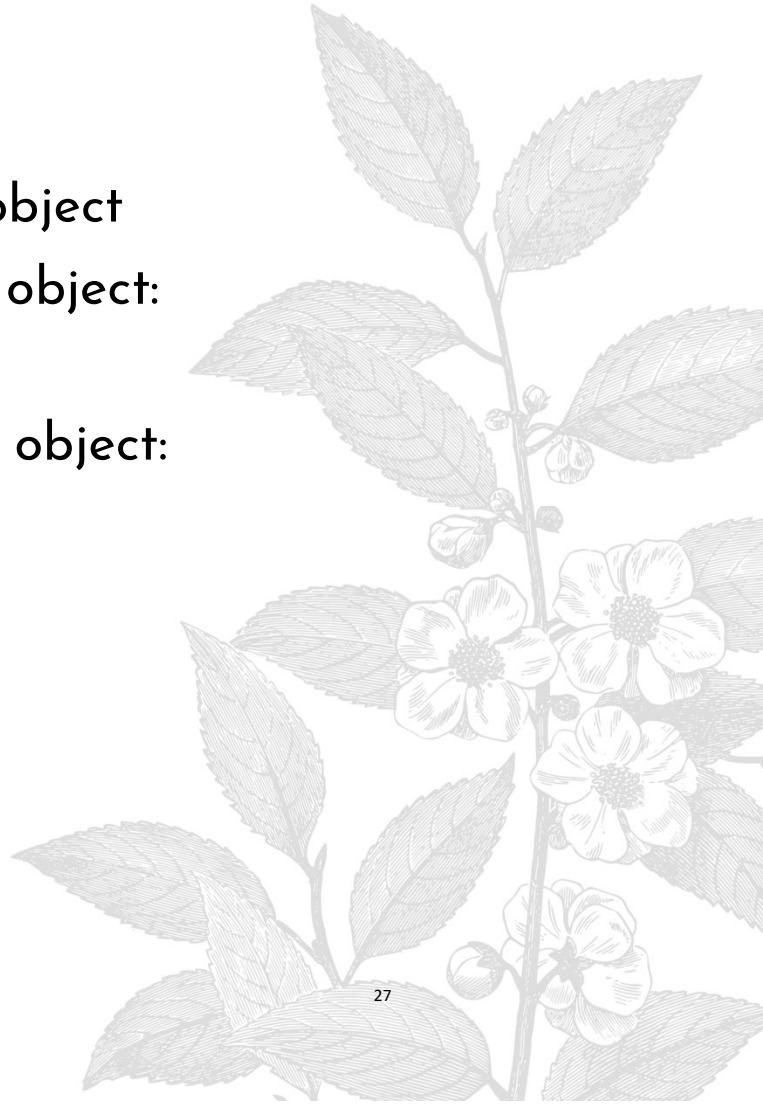
- ✓Object reference: describes the location of an object
- ✓The new operator returns a reference to a new object:

```
Rectangle box = new Rectangle();
```

- ✓Multiple object variables can refer to the same object:

```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
box2.translate(15, 25);
```

- ✓Primitive type variables ≠ object variables



Object Variables and Number Variables

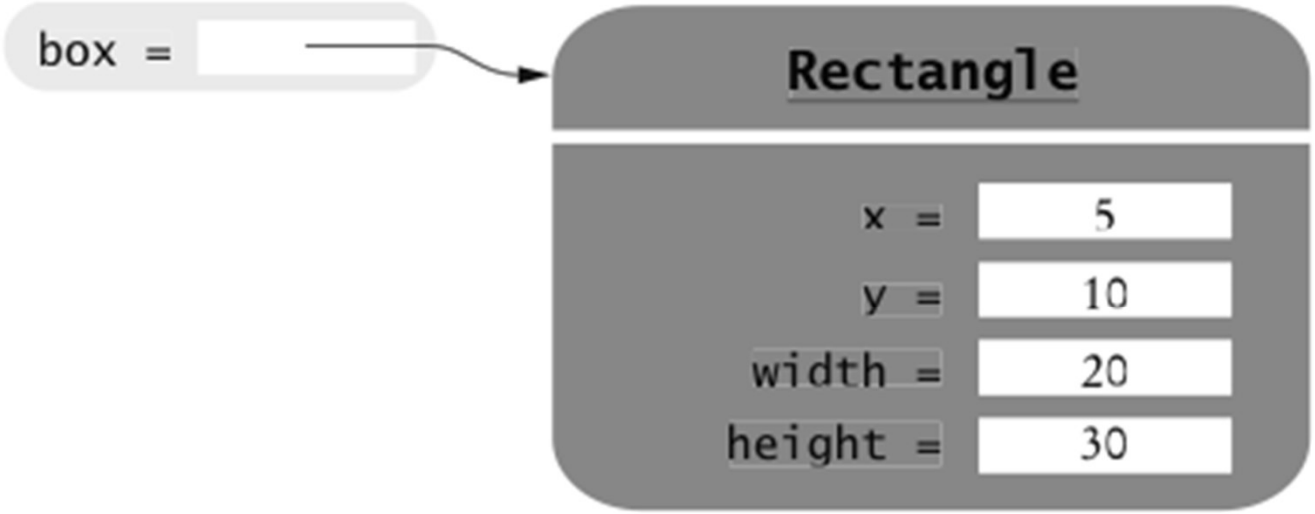


Figure 17 An Object Variable Containing an Object Reference

Object Variables and Number Variables

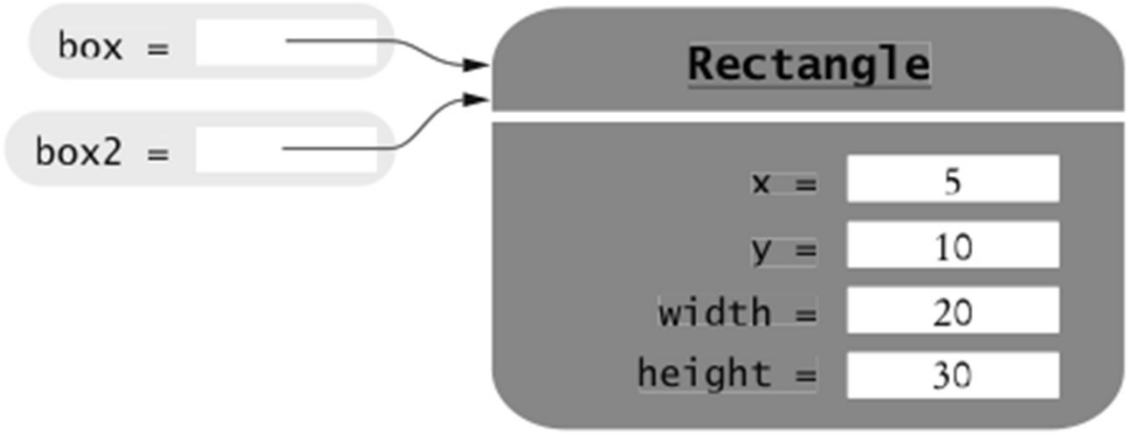


Figure 18 Two Object Variables Referring to the Same Object

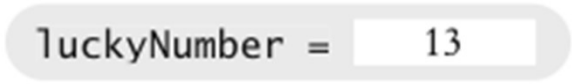
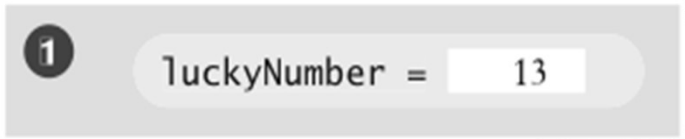


Figure 19 A Number Variable Stores a Number

Copying Numbers

int luckyNumber = 13; ❶

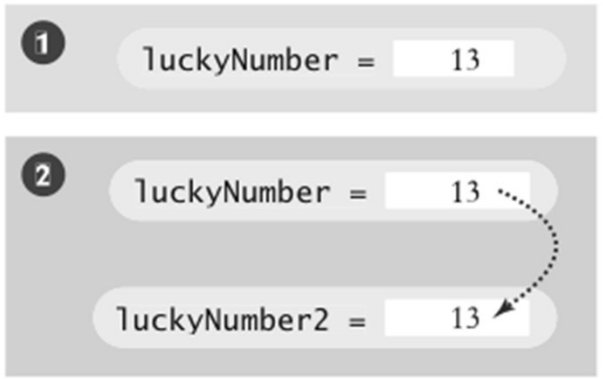
Figure 20
Copying Numbers



Copying Numbers (cont.)

```
int luckyNumber = 13;      ❶
int luckyNumber2 = luckyNumber;  ❷
```

Figure 20
Copying Numbers



Copying Numbers (cont.)

```
int luckyNumber = 13;      ❶
int luckyNumber2 = luckyNumber;   ❷
luckyNumber2 = 12;         ❸
```

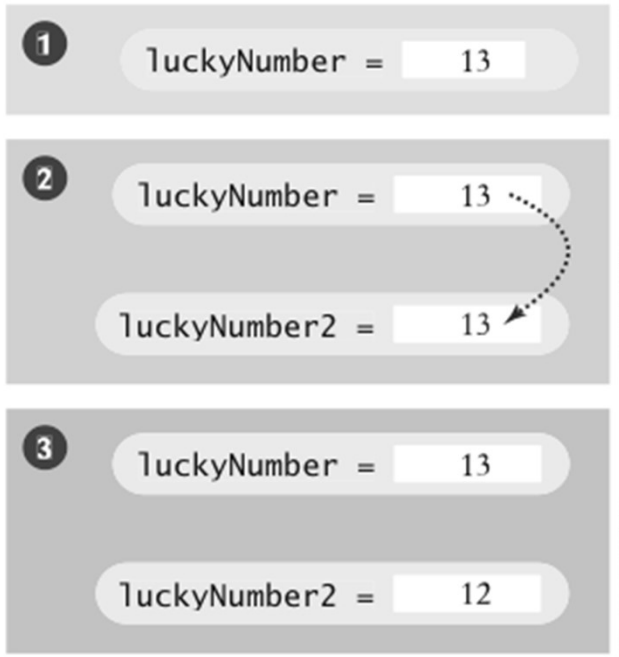
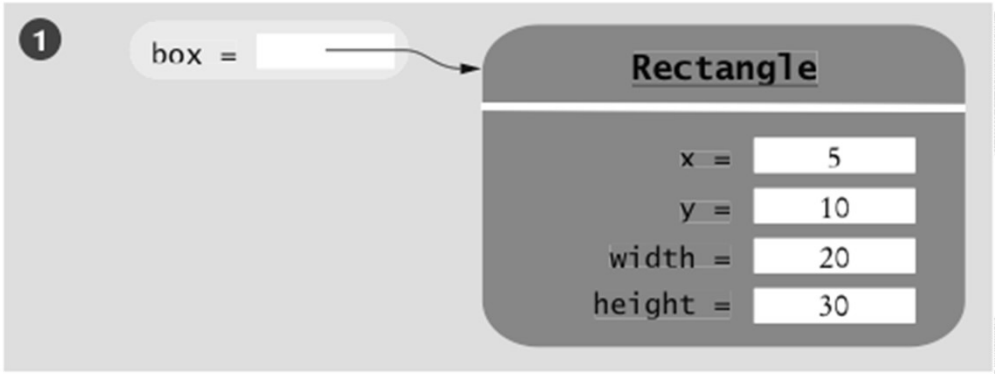


Figure 20
Copying Numbers

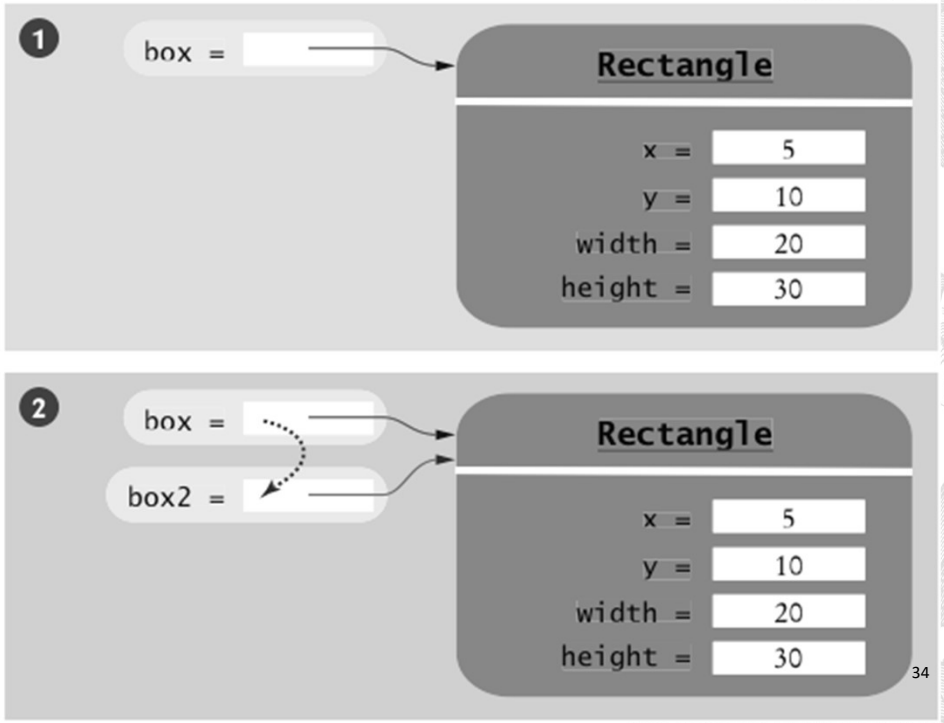
Copying Object References

Rectangle box = new Rectangle(5, 10, 20, 30); ❶



Copying Object References (cont.)

Rectangle box = new Rectangle(5, 10, 20, 30); ❶
Rectangle box2 = box; ❷



Copying Object References (cont.)

```
Rectangle box = new Rectangle(5, 10, 20, 30);  1
Rectangle box2 = box;                          2
Box2.translate(15, 25);                        3
```

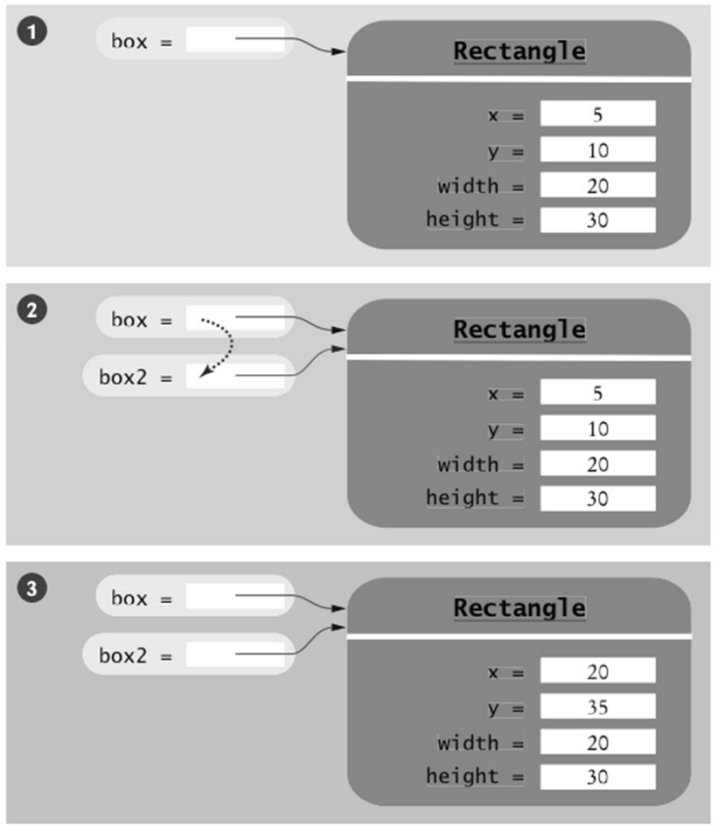
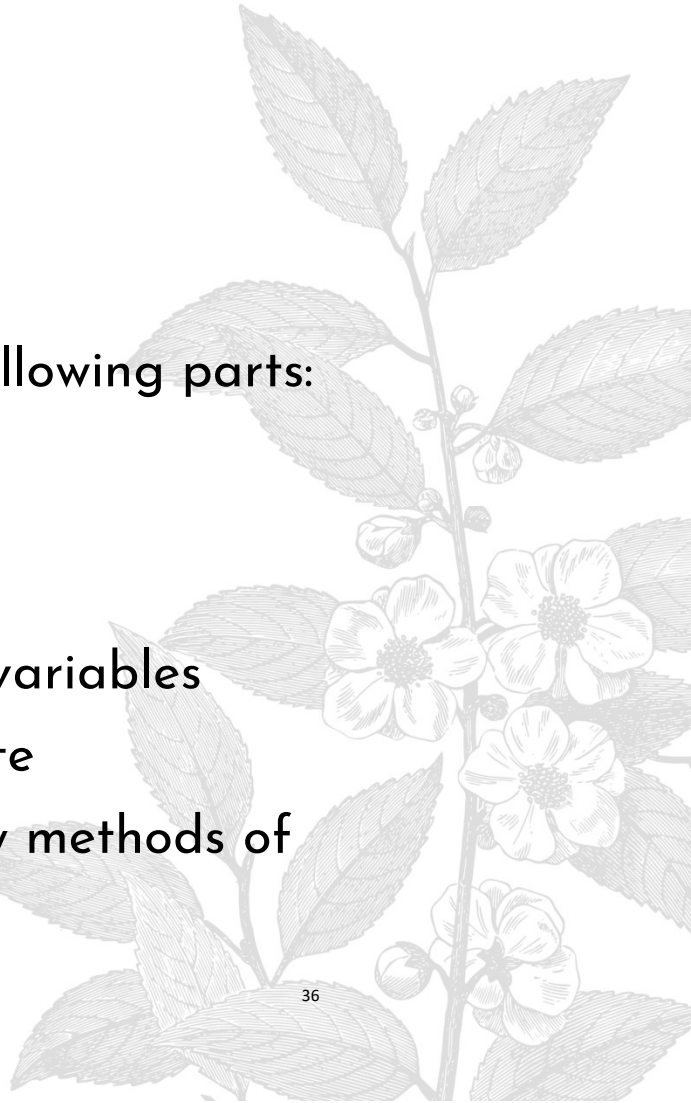


Figure 21 Copying Object References

Instance Variables

- ✓Instance variables store the data of an object
- ✓Instance of a class: an object of the class
- ✓An instance variable declaration consists of the following parts:
  - access specifier (private)
  - type of variable (such as int)
  - name of variable (such as value)
- ✓Each object of a class has its own set of instance variables
- ✓You should declare all instance variables as private
- ✓Private instance variables can only be accessed by methods of the same class



Syntax 3.1 Instance Variable Declaration

Syntax

```
accessSpecifier class ClassName
{
   accessSpecifier typeName variableName;
   . . .
}
```

Example

```
public class Counter
{
   private int value;
   . . .
}
```

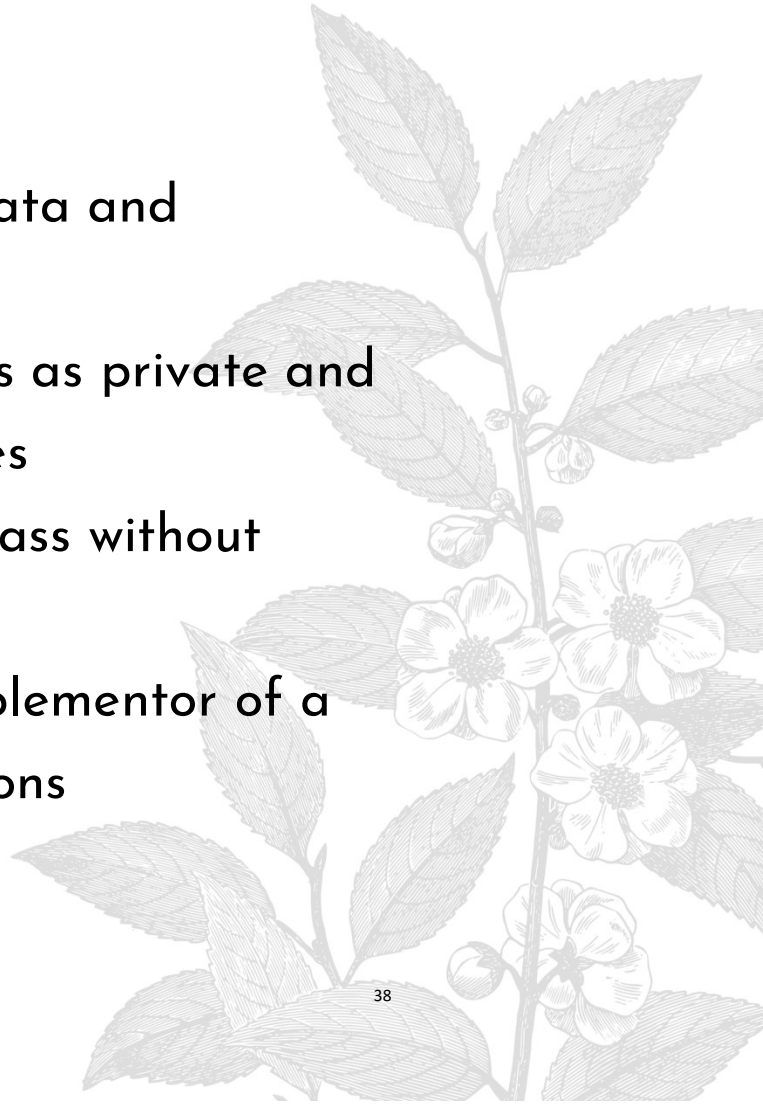
Instance variables should always be private.

Each object of this class has a separate copy of this instance variable.

Type of the variable

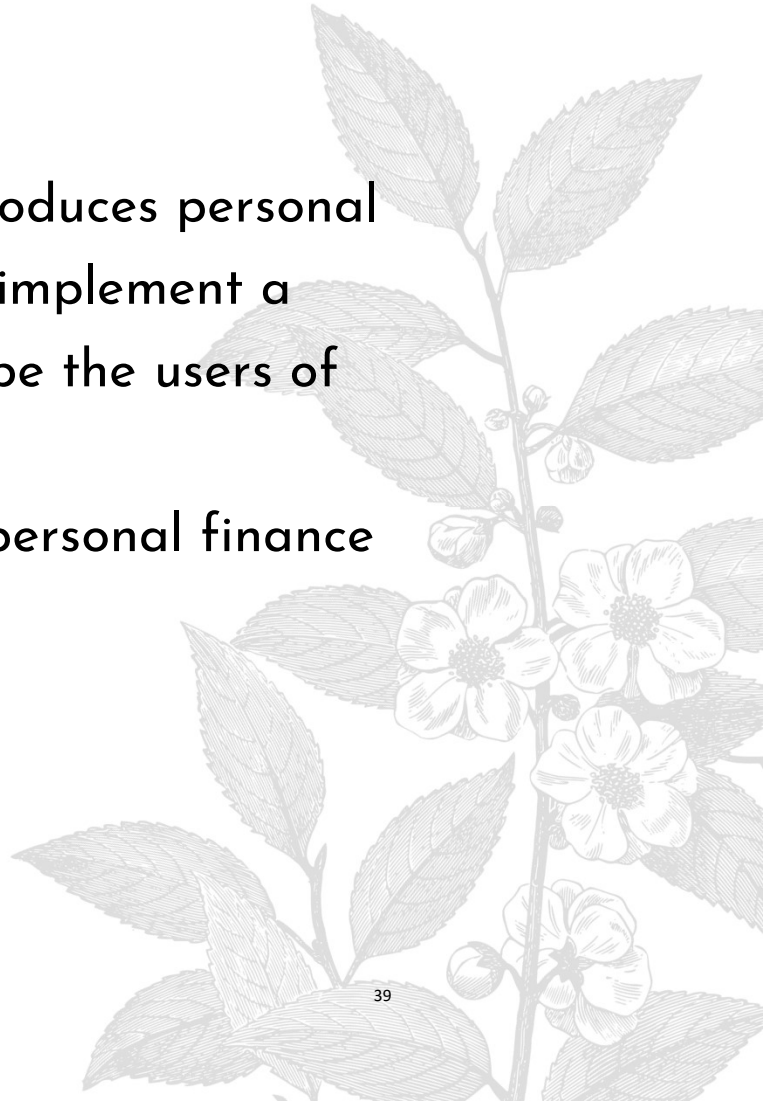
Instance Variables

- ✓Encapsulation is the process of hiding object data and providing methods for data access
- ✓To encapsulate data, declare instance variables as private and declare public methods that access the variables
- ✓Encapsulation allows a programmer to use a class without having to know its implementation
- ✓Information hiding makes it simpler for the implementor of a class to locate errors and change implementations



Self Check 3.5

- ✓Suppose you are working in a company that produces personal finance software. You are asked to design and implement a class for representing bank accounts. Who will be the users of your class?
- ✓**Answer**: Other programmers who work on the personal finance application.



Specifying the Public Interface of a Class

Behavior of bank account (abstraction):

- ✓deposit money
- ✓withdraw money
- ✓get balance

Specifying the Public Interface of a Class: Methods

- ✓Methods of BankAccount class:
  - deposit
  - withdraw
  - getBalance
- ✓We want to support method calls such as the following:

```
harrysChecking.deposit(2000);
harrysChecking.withdraw(500);
System.out.println(harrysChecking.getBalance());
```

Specifying the Public Interface of a Class

Method Declaration:

- ✓access specifier (such as public)
- ✓return type (such as String or void)
- ✓method name (such as deposit)
- ✓list of parameters (double amount for deposit)
- ✓method body in { }

Examples:

- ✓public void deposit(double amount) { . . . }
- ✓public void withdraw(double amount) { . . . }
- ✓public double getBalance() { . . . }

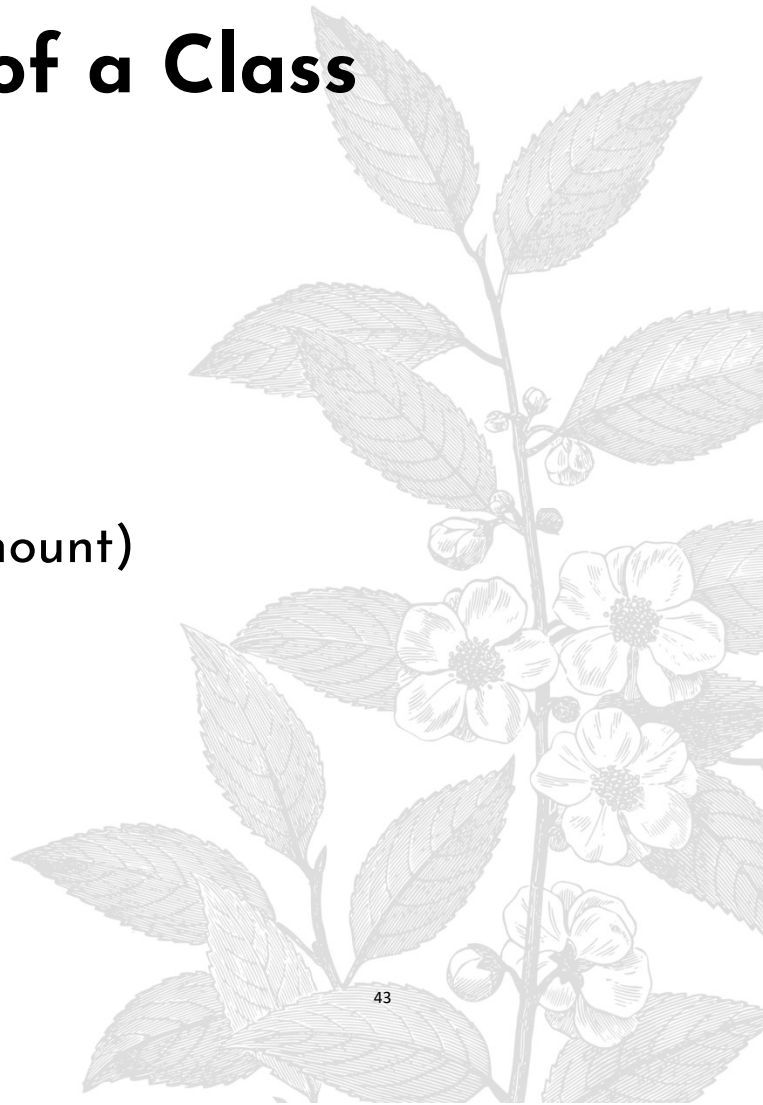
Specifying the Public Interface of a Class

Method Header

- ✓ access specifier (such as public)
- ✓ return type (such as void or double)
- ✓ method name (such as deposit)
- ✓ list of parameter variables (such as double amount)

Examples:

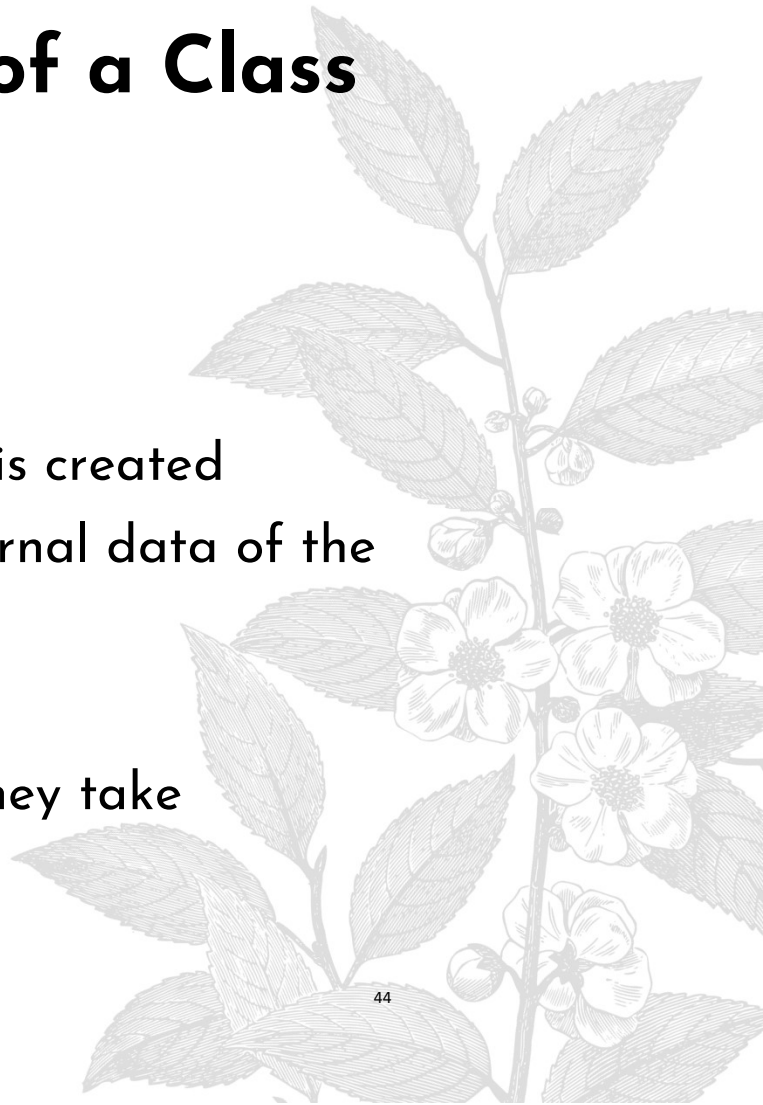
- ✓public void deposit(double amount)
- ✓public void withdraw(double amount)
- ✓public double getBalance()



Specifying the Public Interface of a Class

Constructor Declaration

- ✓A constructor initializes the instance variables
- ✓Constructor name = class name
- ✓Constructor body is executed when new object is created
- ✓Statements in constructor body will set the internal data of the object that is being constructed
- ✓All constructors of a class have the same name
- ✓Compiler can tell constructors apart because they take different parameters



Syntax 3.2 Class Declaration

Syntax

```
accessSpecifier class ClassName
{
   instance variables
   constructors
   methods
}
```

Example

```
public class Counter
{
   private int value;

   public Counter(double initialValue) { value = initialValue; }

   public void count() { value = value + 1; }
   public int getValue() { return value; }
}
```

Public interface

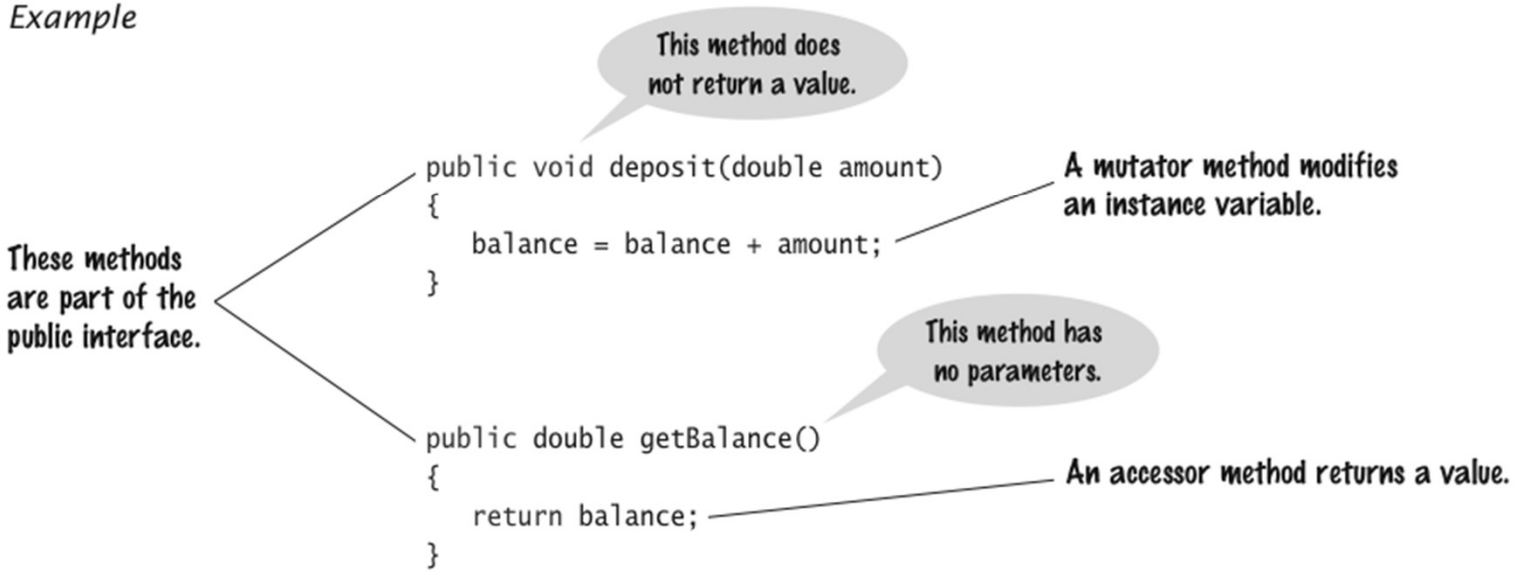
Private implementation

Syntax 3.3 Method Declaration

Syntax

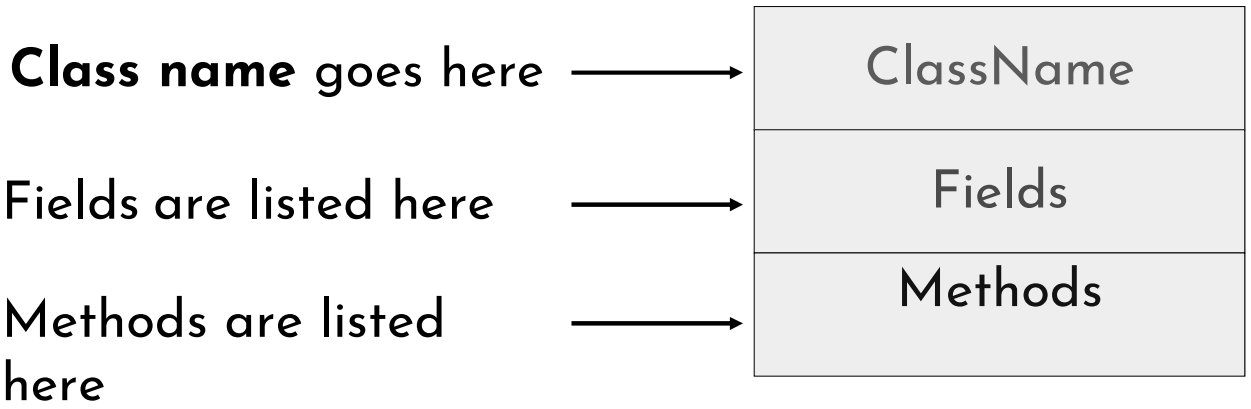
```
accessSpecifier returnType methodName(parameterType parameterName, . . . )
{
   method body
}
```

Example



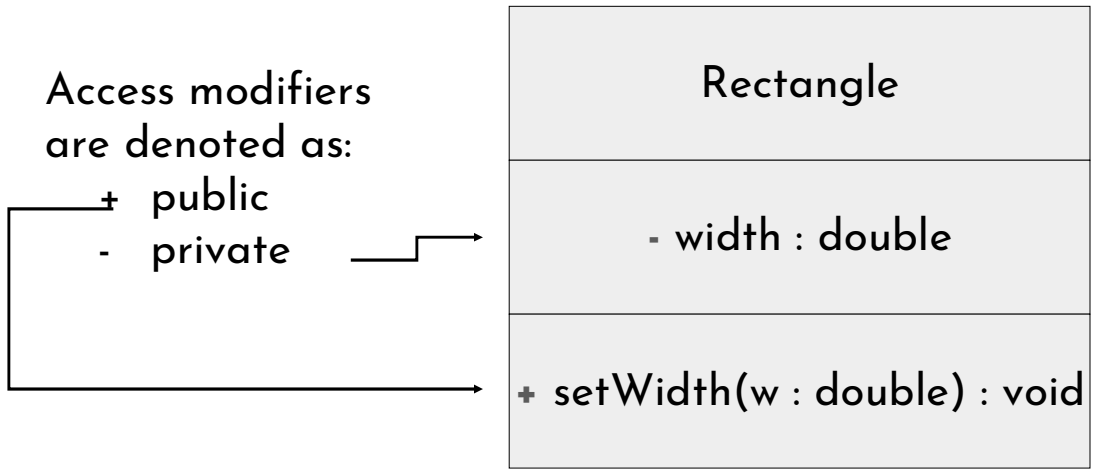
UML Diagram for a class

✓Unified Modeling Language (UML) provides a set of standard diagrams for graphically depicting object-oriented systems.



UML Data Type and Parameter Notation

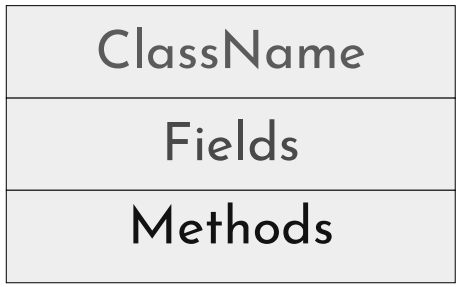
- ✓UML diagrams are language independent.
- ✓UML diagrams use an independent notation to show return types, access modifiers, etc.



Converting the UML Diagram to Code

- ✓Putting all of this information together, a Java class file can be built easily using the UML diagram.
- ✓The UML diagram parts match the Java class file structure.

```
class header
{
  Fields
  Methods
}
```



Converting the UML Diagram to Code

The structure of the class can be compiled and tested without having bodies for the methods. Just be sure to put in dummy return values for methods that have a return type other than void.

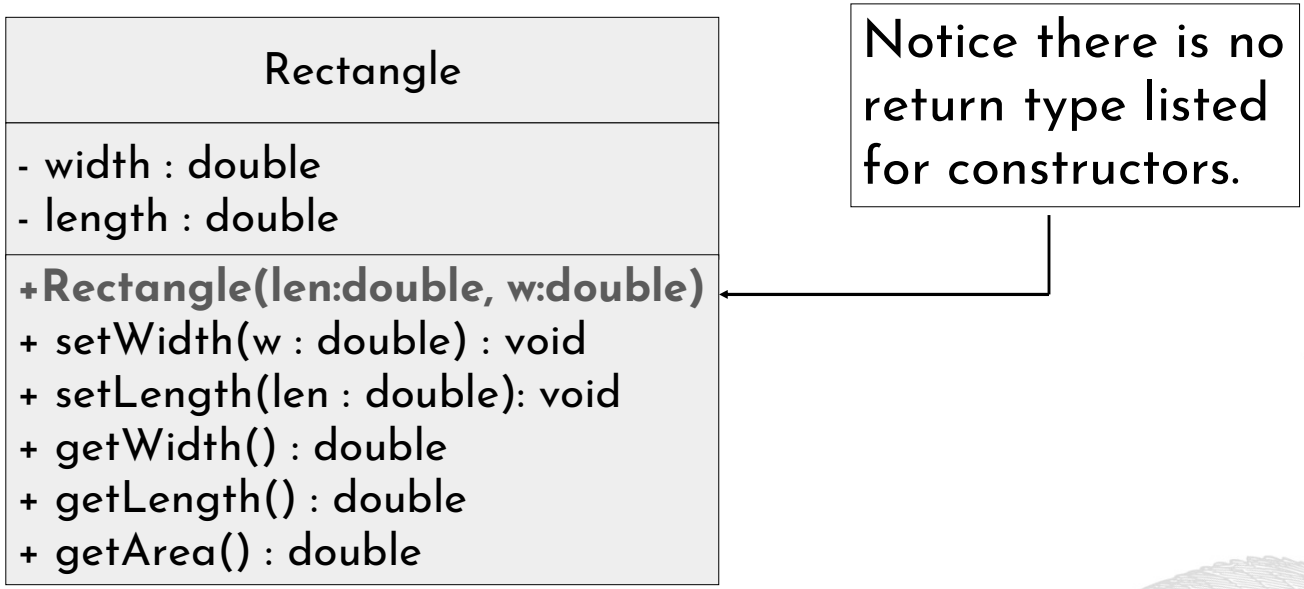
Rectangle
- width : double - length : double
+ setWidth(w : double) : void + setLength(len : double): void + getWidth() : double + getLength() : double + getArea() : double

```
public class Rectangle {
   private double width;
   private double length;

   public void setWidth(double w){
   }
   public void setLength(double len){
   }
   public double getWidth(){
        return 0.0;
   }
   public double getLength(){
        return 0.0;
   }
   public double getArea()   {
        return 0.0;
   }
}
```

Constructors in UML

✓In UML, the most common way constructors are defined is:



Unit Testing

- ✓**Unit test:** Verifies that a class works correctly in isolation, outside a complete program
- ✓To test a class, use an environment for interactive testing, or write a tester class
- ✓**Tester class:** A class with a main method that contains statements to test another class
- ✓Typically carries out the following steps:
  - Construct one or more objects of the class that is being tested
  - Invoke one or more methods
  - Print out one or more results
  - Print the expected results