Lecture 4

A Second Look at Class

References:

Tony Gaddis, Chapter 6, Starting out with Java: From Control Structures through Objects, 7 edition

Chapter Topics

- Objects and Classes
- Writing a Simple Class, Step by Step
- Instance Fields and Methods
- Constructors
- Passing Objects as Arguments
- Overloading Methods and Constructors
- Scope of Instance Fields
- Packages and `import` Statements

Objects and Classes (1 of 6)

- An object
  - Exists in memory and performs a specific task
- Objects have two general capabilities:
  - Fields (Attributes): Objects can store data
  - Methods: Objects can perform operations

3

Objects and Classes (2 of 6)

- Example of objects:
  - `Scanner` objects, for reading input
  - `Random` objects, for generating random numbers
- When a program needs the services of object,
  - Creates that object in memory,
  - Then calls that object's methods as necessary

4

Objects and Classes (3 of 6)

- Classes
  - Where objects come from
  - Code describing a particular type of object
  - Specifies the object's fields and methods
  - A class as a code "blueprint" used to create as many objects as needed

5

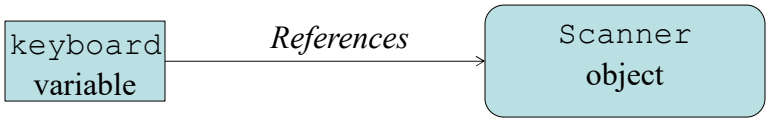
Objects and Classes (4 of 6)

Example:

This expression creates a `Scanner` object in memory.

```
Scanner keyboard = new Scanner(System.in);
```

The object's memory address is assigned to the `keyboard` variable.



6

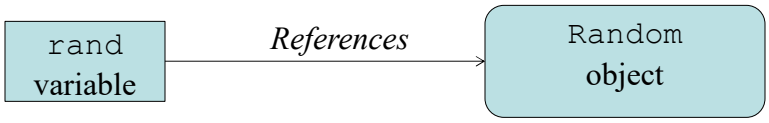
Objects and Classes (5 of 6)

Example:

This expression creates a `Random` object in memory.

```
Random rand = new Random();
```

The object's memory address is assigned to the `rand` variable.



7

Objects and Classes (6 of 6)

- The Java API provides many classes
  - The classes creating objects are provided by the Java API
  - Examples:
    - `Scanner`
    - `Random`

8

Writing a Class, Step by Step (1 of 2)

- A `Rectangle` object with two fields:
  - `length` to hold the rectangle's length.
  - `width` to hold the rectangle's width.

9

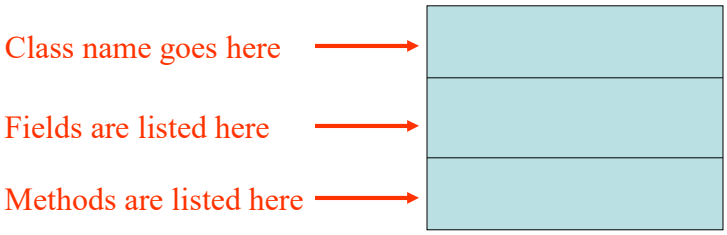
Writing a Class, Step by Step (2 of 2)

- The `Rectangle` class's methods:
  - **`setLength to`** store a value in an object's `length` field
  - **`setWidth to`** store a value in an object's `width` field
  - **`getLength to`** return the value in an object's `length` field
  - **`getWidth to`** return the value in an object's `width` field
  - **`getArea to`** return the area of the rectangle

10

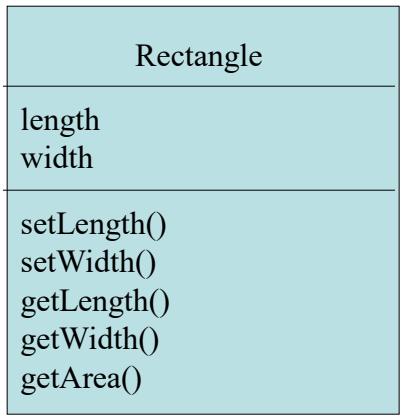
UML Diagram

- Unified Modeling Language (UML)
  - Provides a set of standard diagrams for graphically modeling object-oriented systems



11

UML Diagram for `Rectangle` class



12

Writing the Code for the Class Fields

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

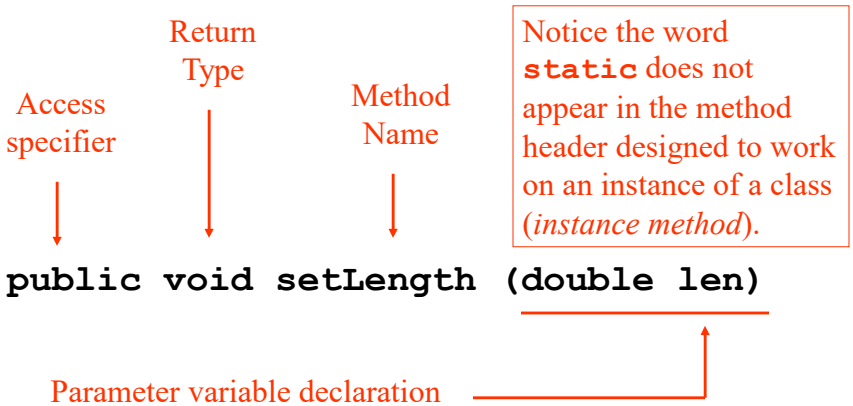
13

Access Specifiers

- An access specifier
  - A Java keyword indicating how to access a field or method
- `public`
  - The class's member accessed by code inside the class or outside
- `private`
  - The member not accessed by code outside the class
  - The member accessed only by methods in the same class

14

Header for the `setLength` Method



15

Writing and Demonstrating the `setLength` Method

```
/**
    The setLength method stores a value in the
    length field.
    @param len The value to store in length.
*/
public void setLength(double len)
{
    length = len;
}
```

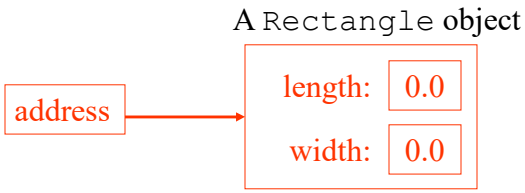
Examples: Rectangle.java, LengthDemo.java

16

Creating a `Rectangle` object

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

The `box` variable holds the address of the Rectangle object.

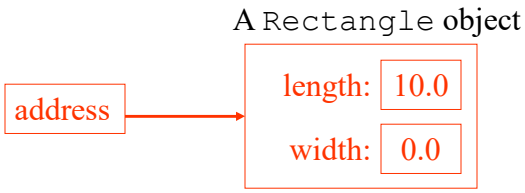


17

Calling the `setLength` Method

```
box.setLength(10.0);
```

The `box` variable holds the address of the `Rectangle` object.



*This is **the state of the box object** after the* `setLength` *method executes*

18

Writing the `getLength` Method

```
/**
    The getLength method returns a Rectangle
    object's length.
    @return The value in the length field.
*/
public double getLength()
{
   return length;
}
```

Similarly, the `setWidth` and `getWidth` methods can be created

Examples: Rectangle.java, LengthWidthDemo.java

19

Writing and Demonstrating the `getArea` Method

```
/**
    The getArea method returns a Rectangle
    object's area.
    @return The product of length times width.
 */
 public double getArea()
 {
    return length * width;
 }
```

Examples: Rectangle.java, RectangleDemo.java

20

Accessor and Mutator Methods

- Private fields
  - Because of the concept of data hiding
- Accessor (*getter)*
  - The methods that retrieve the data of fields
- Mutator (*setter)*
  - The methods that modify the data of fields

21

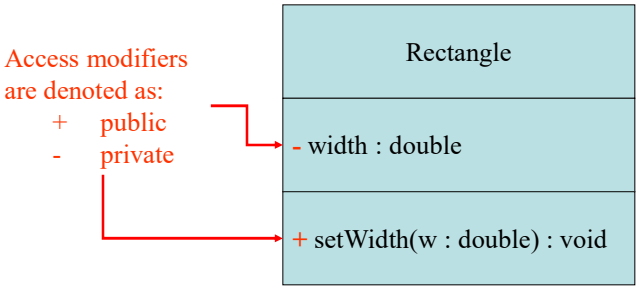
Accessors and Mutators

- For the `Rectangle` example, the accessors and mutators are:
  - **`setLength`** : Sets the value of the `length` field.
    ```
    public void setLength(double len) ...
    ```
  - **`setWidth`** : Sets the value of the `width` field.
    ```
    public void setLength(double w) ...
    ```
  - **`getLength`** : Returns the value of the `length` field.
    ```
    public double getLength() ...
    ```
  - **`getWidth`** : Returns the value of the `width` field.
    ```
    public double getWidth() ...
    ```

22

UML Data Type and Parameter Notation (1 of 4)

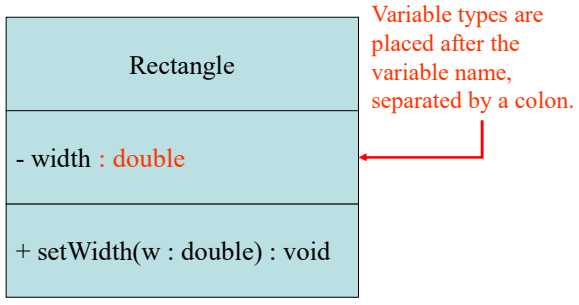
- UML diagrams are language independent
- UML diagrams showing return types, access modifiers, etc.



23

UML Data Type and Parameter Notation (2 of 4)

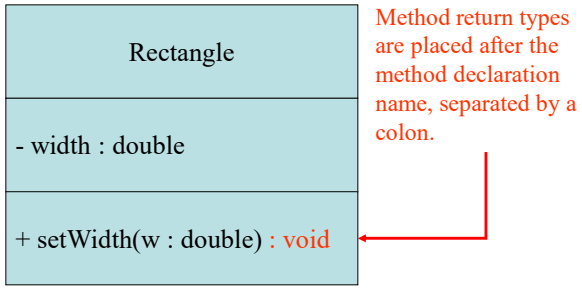
- UML diagrams are language independent.
- UML diagrams showing return types, access modifiers, etc.



24

UML Data Type and Parameter Notation (3 of 4)

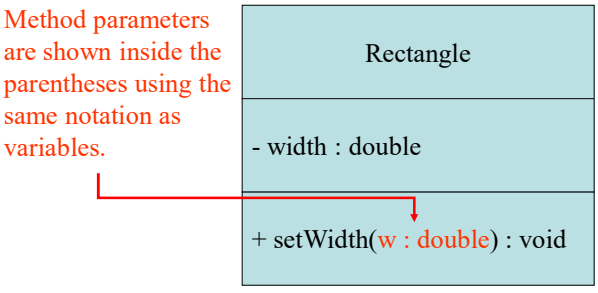
- UML diagrams are language independent.
- UML diagrams showing return types, access modifiers, etc.



25

UML Data Type and Parameter Notation (4 of 4)

- UML diagrams are language independent.
- UML diagrams showing return types, access modifiers, etc

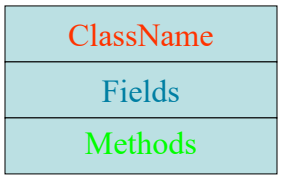


26

Converting the UML Diagram to Code (1 of 3)

- A Java class file built easily using the UML diagram
- The UML diagram matching the Java class file structure

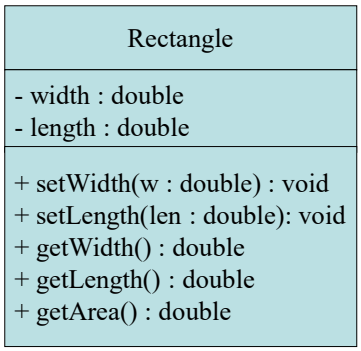
```
class header
{
   Fields
   Methods
}
```



27

Converting the UML Diagram to Code (2 of 3)

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.



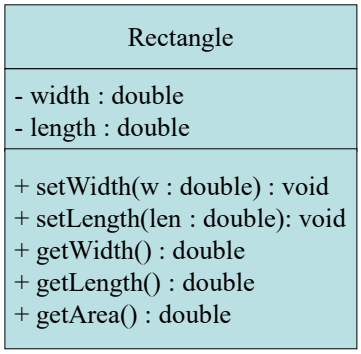

```
public class Rectangle
{
    private double width;
    private double length;

    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;
    }
}
```

28

Converting the UML Diagram to Code (3 of 3)

Once the class structure has been tested, the method bodies can be written and tested.




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

   public void setWidth(double w)
   {     width = w;
   }
   public void setLength(double len)
   {     length = len;
   }
   public double getWidth()
   {     return width;
   }
   public double getLength()
   {     return length;
   }
   public double getArea()
   {     return length * width;
   }
}
```

29

Instance Fields and Methods (1 of 2)

- Fields and methods
  - Called *instance fields* and *instance methods*
- Objects created from a class
  - Each have their own copy of instance fields
- Instance methods
  - Methods that are not declared with a special keyword, `static`

30

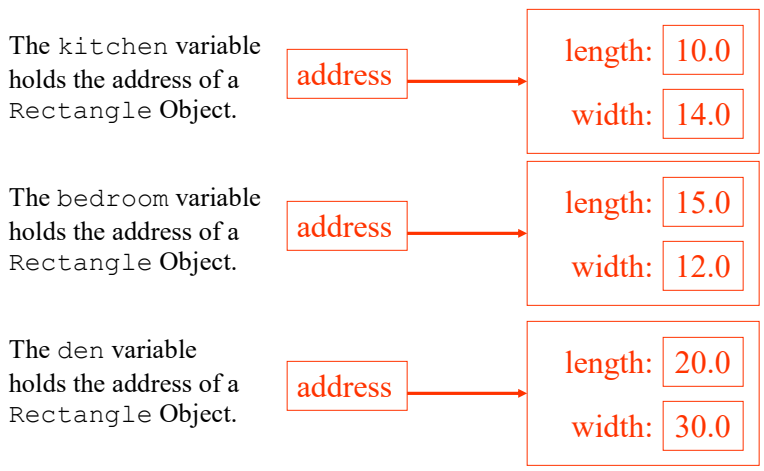
Instance Fields and Methods (2 of 2)

- An object
  - Requires its instance fields and instance methods
- See example: RoomAreas.java
- Note that each room have different dimensions

```
Rectangle kitchen = new Rectangle();
Rectangle bedroom = new Rectangle();
Rectangle den = new Rectangle();
```

31

States of Three Different Rectangle Objects



32

Constructors (1 of 2)

- A constructor
  - A method called when an object is created
  - Used to create objects
  - Typically initialize instance fields and perform other object initialization tasks

33

Constructors (2 of 2)

- Constructors have a few special properties
  - Constructors have the same name as the class.
  - Constructors have no return type (not even `void`).
  - Constructors may not return any values
  - Constructors are typically public

34

Constructor for `Rectangle` Class

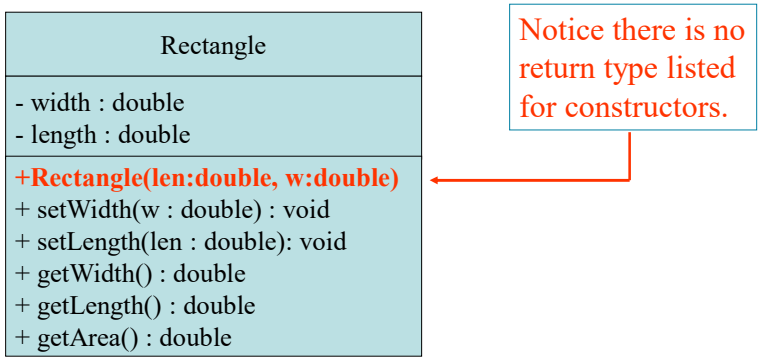
```
/**
   Constructor
   @param len The length of the rectangle.
   @param w The width of the rectangle.
*/
public Rectangle(double len, double w)
{
   length = len;
   width = w;
}
```

Examples: Rectangle.java, ConstructorDemo.java

35

Constructors in UML

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



36

Uninitialized Local Reference Variables

- Reference variables declared without being initialized

```
Rectangle box;
```

- Must reference an object before it can be used, otherwise a compiler error will occur.

```
box = new Rectangle(7.0, 14.0);
```

37

The Default Constructor (1 of 2)

- When an object is created,
  - its constructor is always called
- *default constructor*
  - Has no parameters to initialize an object
  - Sets all of the object's numeric fields to 0.
  - Sets all of the object's `boolean` fields to `false`.
  - Sets all of the object's reference variables to the special value *null*.

38

The Default Constructor (2 of 2)

- See example: First version of Rectangle.java
- A default constructor not provided by Java if a constructor is already written.
  - See example: Rectangle.java with Constructor
- We can write our own no-arg constructor (like default constructor)

```
public Rectangle()
{
   length = 1.0;
   width = 1.0;
}
```

39

The `String` Class Constructor

- `String` objects created with constructor

```
String name = new String("Welcome all");
```

- `String` objects created with a shorthand:

```
String name = "Welcome all";
```

40

Passing Objects as Arguments

- When passing an object as an argument,
  - Passed into the parameter variable is the object's memory address
  - Parameter variable references the object
- See DieArgument.java

41

Overloading Methods and Constructors

- Method overloading
  - Two or more methods having the same name
  - As long as their parameter lists are different
  - Also applies to constructors
  - Need several different ways to perform the same operation

42

Overloaded Method `add`

- Why overloading needed?
  - To perform the same operation in different ways

```
public int add(int num1, int num2)
{
 int sum = num1 + num2;
 return sum;
}

public String add (String str1, String str2)
{
 String combined = str1 + str2;
 return combined;
}
```

43

Method Signature and Binding

- A method signature
  - Consists of the method's name and the data types of the method's parameters
  - The return type is not part of the signature

`add(int, int)`
`add(String, String)`

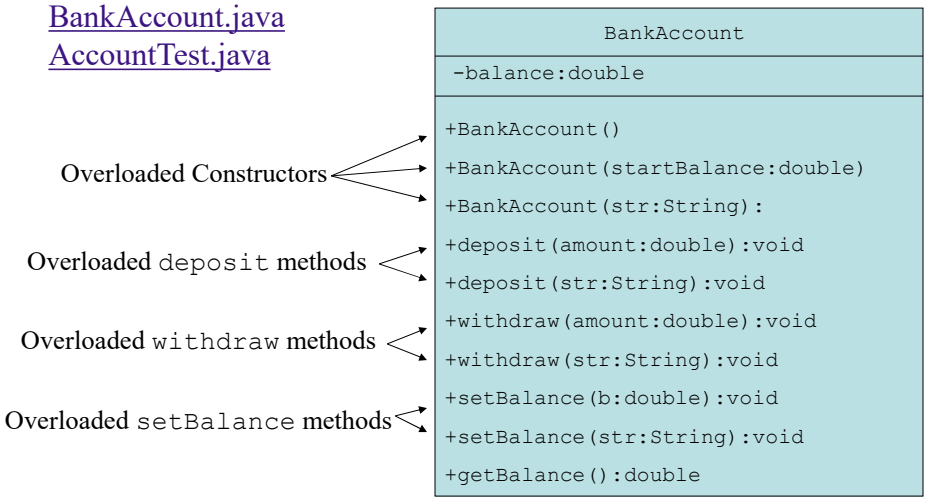
Signatures of the `add` *methods of previous slide*

- Binding
  - The process of matching a method call with the correct method
  - Determine which version of the overloaded method using the method signature

44

The `BankAccount` Example

BankAccount.java
AccountTest.java



45

Scope of Instance Fields

- Variables declared as instance fields in a class
  - Accessed by any instance method in the same class
- If an instance field is declared with the `public` access specifier
  - can also be accessed by code outside the class
  - as long as an instance of the class exists

46

Shadowing

- Instance variables can be accessed by any instance method
- Shadowing
  - A method may have a local variable with the same name as an instance field
  - The local variable will *hide* the value of the instance field
  - Shadowing is discouraged

47

Packages and `import` Statements

- Classes in the Java API organized into *packages*
- Explicit and Wildcard `import` statements
  - Explicit imports name a specific class
    - `import java.util.Scanner;`
  - Wildcard imports name a package, followed by an `*`
    - `import java.util.*;`
- The `java.lang` package automatically made available to any Java class

48

Some Java Standard Packages

Table 6-2 A few of the standard Java packages

Package	Description
java.io	Provides classes that perform various types of input and output.
java.lang	Provides general classes for the Java language. This package is automatically imported.
java.net	Provides classes for network communications.
java.security	Provides classes that implement security features.
java.sql	Provides classes for accessing databases using structured query language.
java.text	Provides various classes for formatting text.
java.util	Provides various utility classes.