ICT104

Program Design and Development

Lecture 1– Object Oriented Concepts

Adopted from: Gaddis & Gaddis (2019) Starting Out with Java: From Control Structures through Objects, 7th Edition.

King's Own Institute

Focus for this week

01 Objects and Classes

- Introduction
- Where Objects come from
- Classes in the Java API
- Primitive variables vs. Objects

02 Writing a Simple Class Step by Step

- Accessor and Mutator Methods
- The importance of data hiding
- Avoiding stale data
- Showing Access specifications in UML Diagrams
- Data type and Parameter Notation in UML Diagrams
- Layout of Class members

Subject Learning Outcomes

a) Analyse and model object-oriented programming using Java

b) Design object-oriented programs using object-oriented features such as encapsulation, inheritance and polymorphism in Java

c) Design and develop a well-designed event driven application using Java Applets which correctly implements a solution to a problem defined in a specification

d) Implement and use Java programming language features to design and create Swing Components

Assessments

Assessment Type	When assessed	Weighting	Learning Outcomes Assessed
Assessment 1: Tutorial Exercises	Weeks 2 - 11	10%	a, b, c, d
Assessment 2: Quiz A	Week 4	5%	a
Assessment 3: Quiz B	Week 8	15%	b
Assessment 4: Java Practical Exam	Week 11	20%	c, d
Assessment 5: Final examination On-campus: 2 hours + 10 mins reading time Online: 2 hours + 30 mins technology allowance	Final exam period	50%	a, b, c, d

Activity 1:
Revision Exercise

List any three concepts related to Java Programs which you already know (for instance, from ICT102)

Objects and Classes

- 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

- You have already used the following objects:
  - `Scanner` objects, for reading input
  - `Random` objects, for generating random numbers
  - `PrintWriter` objects, for writing data to files
- When a program needs the services of a particular type of object, it creates that object in memory, and then calls that object's methods as necessary

Objects and Classes

- Classes: Where Objects Come From
  - A *class* is a 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

Objects and Classes

- 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

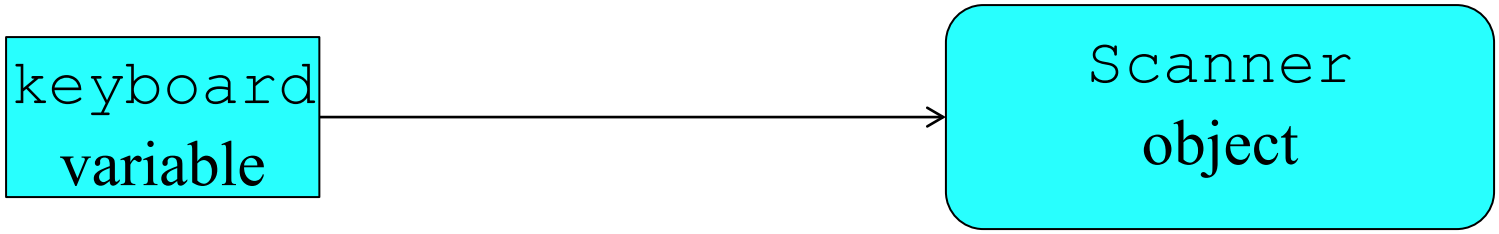
Objects and Classes

Example:

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



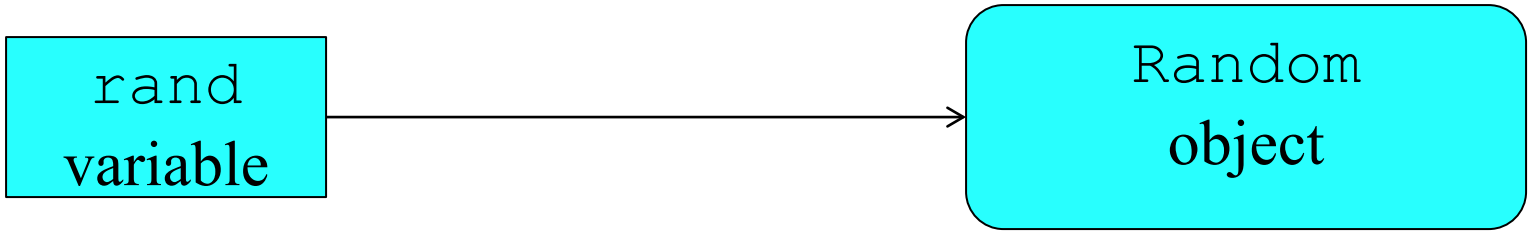
Objects and Classes

Example:

This expression creates a `Random` object in memory

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

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



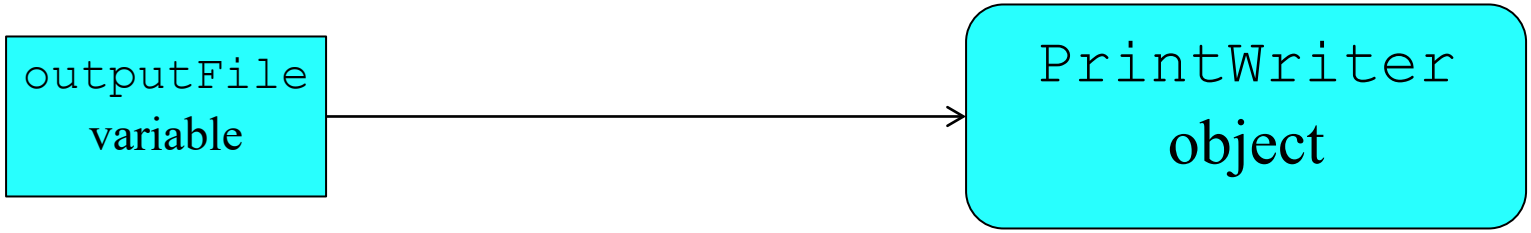
Objects and Classes

Example:

This expression creates a `PrintWriter` object in memory

```
PrintWriter outputFile = new PrintWriter("numbers.txt");
```

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



Objects and Classes

- The Java API provides many classes
  - So far, the classes that you have created objects from are provided by the Java API
  - Examples:
    - `Scanner`
    - `Random`
    - `PrintWriter`

Activity 2: Poll

Methods are commonly used to:

A) speed up the compilation of a program

B) break a problem down into small manageable pieces

C) emphasize certain parts of the logic

D) document the program

Writing a Class, Step by Step

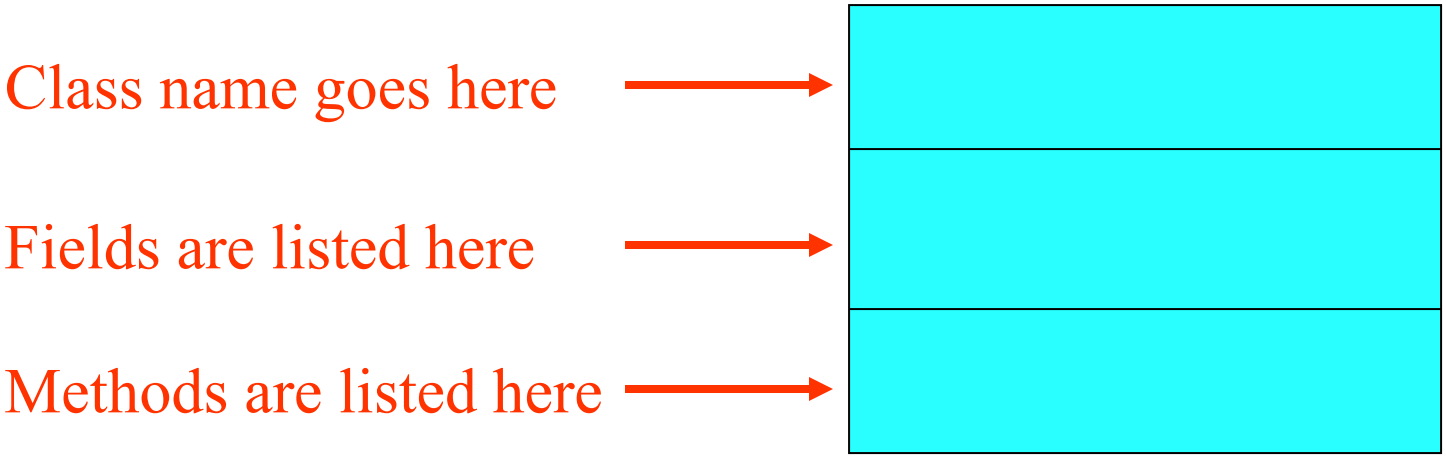
- A `Rectangle` object will have the following fields:
  - `length`. The length field will hold the rectangle's length
  - `width`. The width field will hold the rectangle's width

Writing a Class, Step by Step

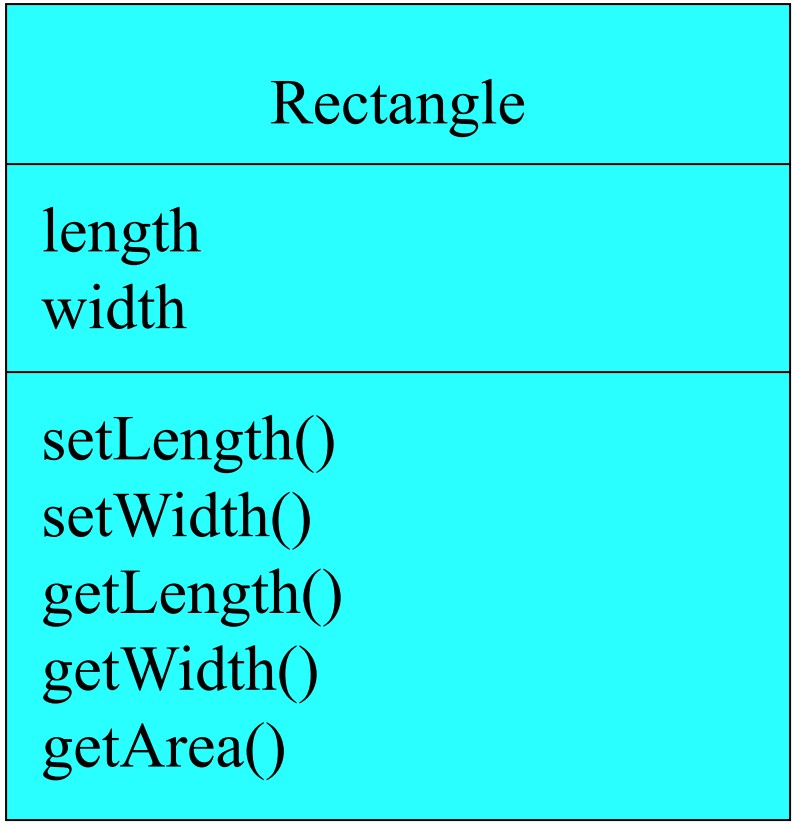
- The `Rectangle` class will also have the following methods:
  - **`setLength`.** The `setLength` method will store a value in an object's `length` field
  - **`setWidth`.** The `setWidth` method will store a value in an object's `width` field
  - **`getLength`.** The `getLength` method will return the value in an object's `length` field
  - **`getWidth`.** The `getWidth` method will return the value in an object's `width` field
  - **`getArea`.** The `getArea` method will return the area of the rectangle, which is the result of the object's `length` multiplied by its `width`

UML Diagram

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



UML Diagram for `Rectangle` class



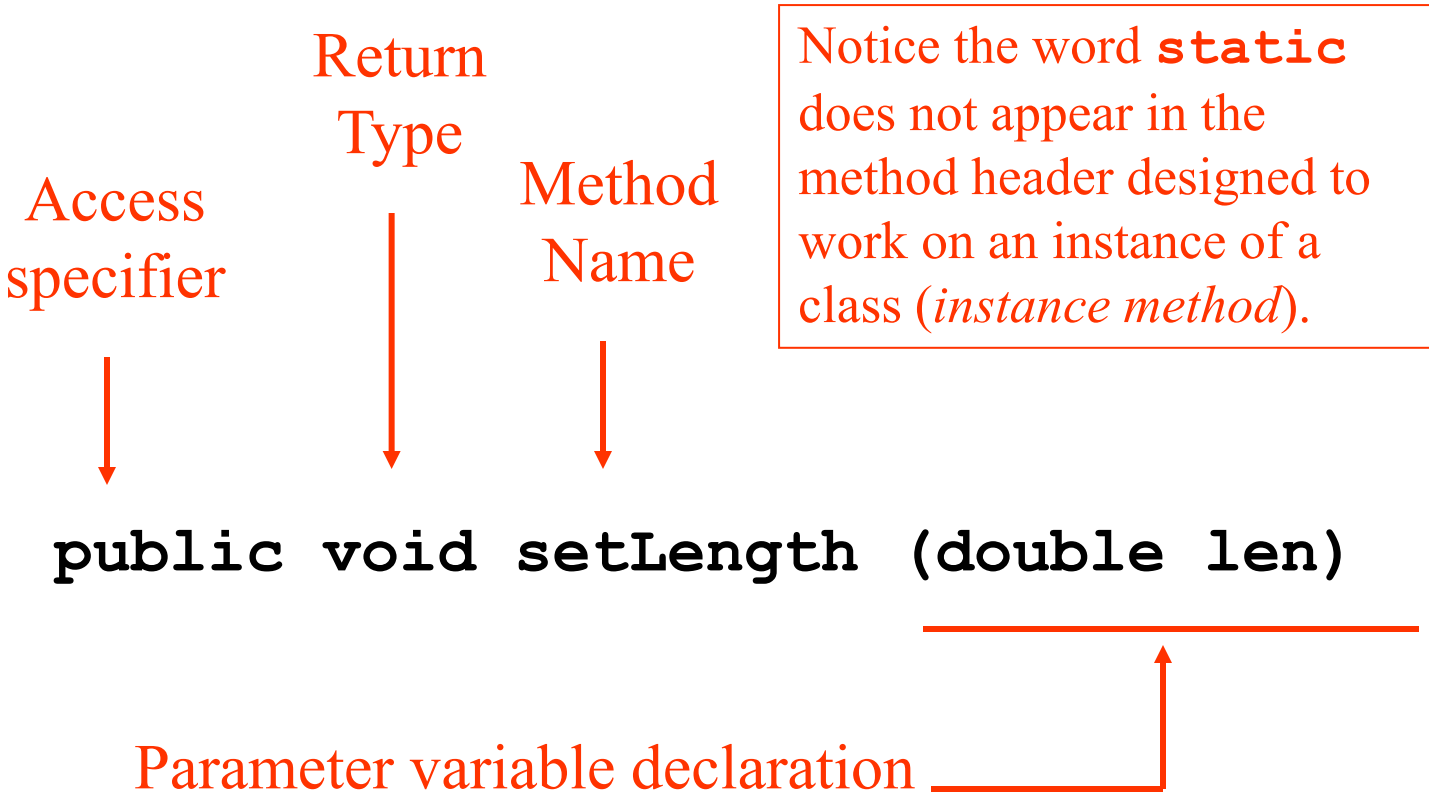
Writing the Code for the Class Fields

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

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

Header for the `setLength` Method



Writing and Demonstrating the `setLength` Method

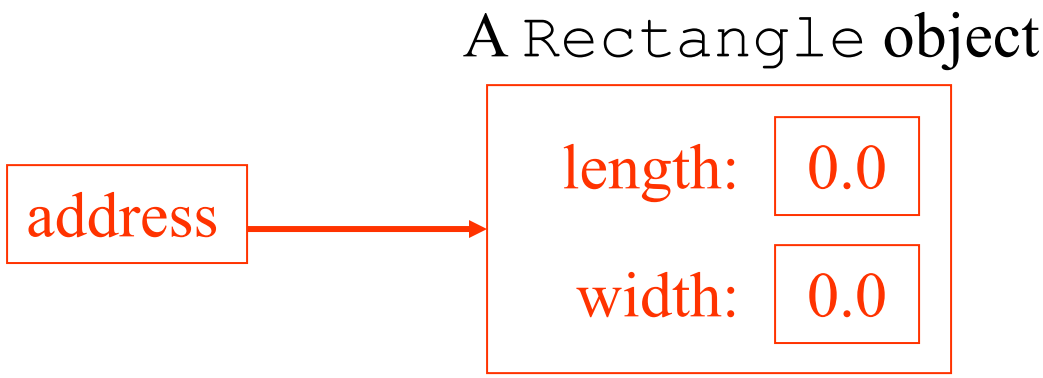
```
/**
   The setLength method stores a value in the
   length field.
   @param len The value to store in length.
*/
public void setLength(double len)
{
   length = len;
}
```

Examples (Phase 1): Rectangle.java, LengthDemo.java

Creating a `Rectangle` object

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

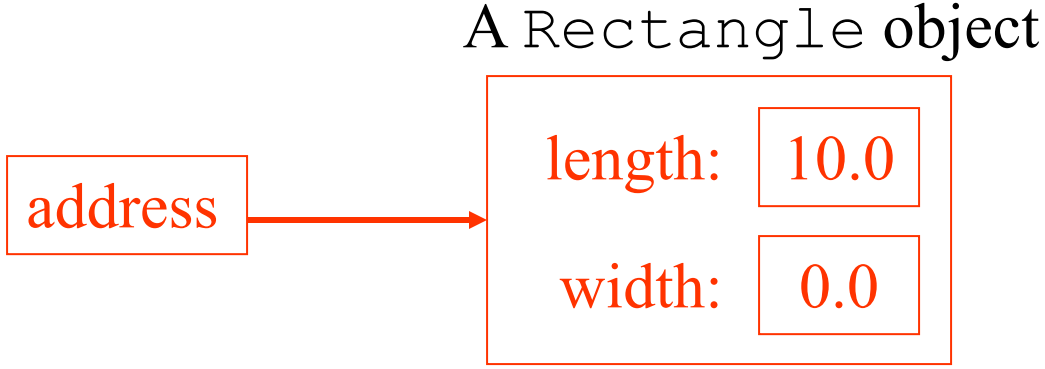
The `box` variable holds the address of the Rectangle object.



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*

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 (Phase 2): Rectangle.java, LengthWidthDemo.java

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;
 }
```

Activity 3: Poll

This type of method performs a task and sends a value back to the code that called it

A) value-returning

B) void

C) complex

D) local

Accessor and Mutator Methods

- 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

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() ...
    ```
- Other names for these methods are *getters* and *setters*

Data Hiding

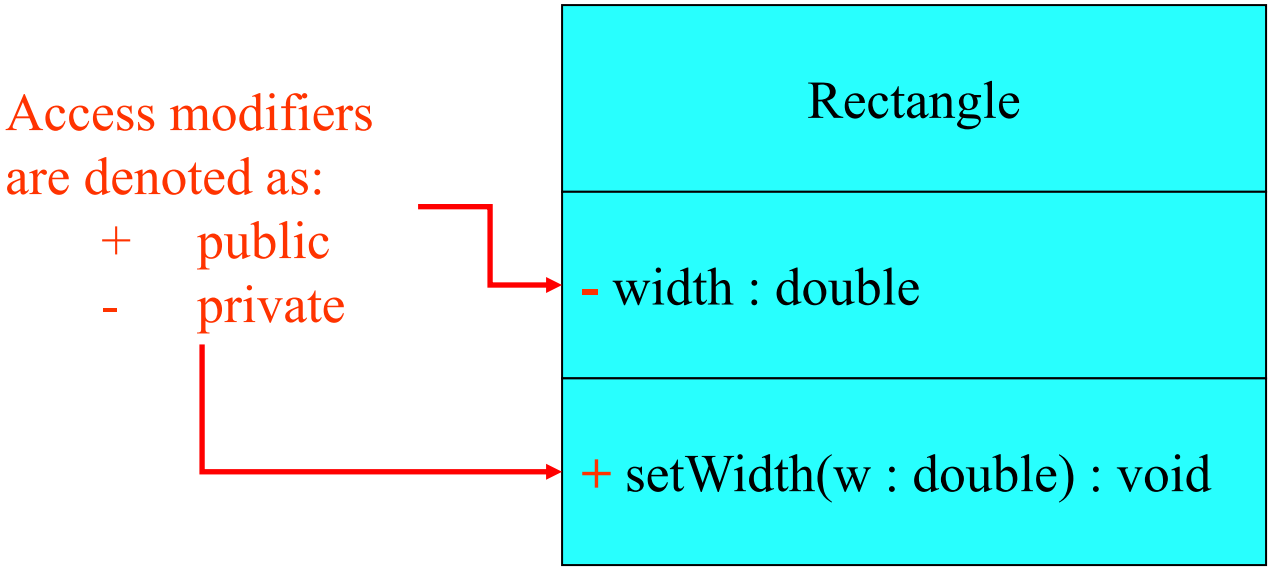
- An object hides its internal, private fields from code that is outside the class that the object is an instance of
- Only the class's methods may directly access and make changes to the object’s internal data
- Code outside the class must use the class's public methods to operate on an object's private fields

Data Hiding

- 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

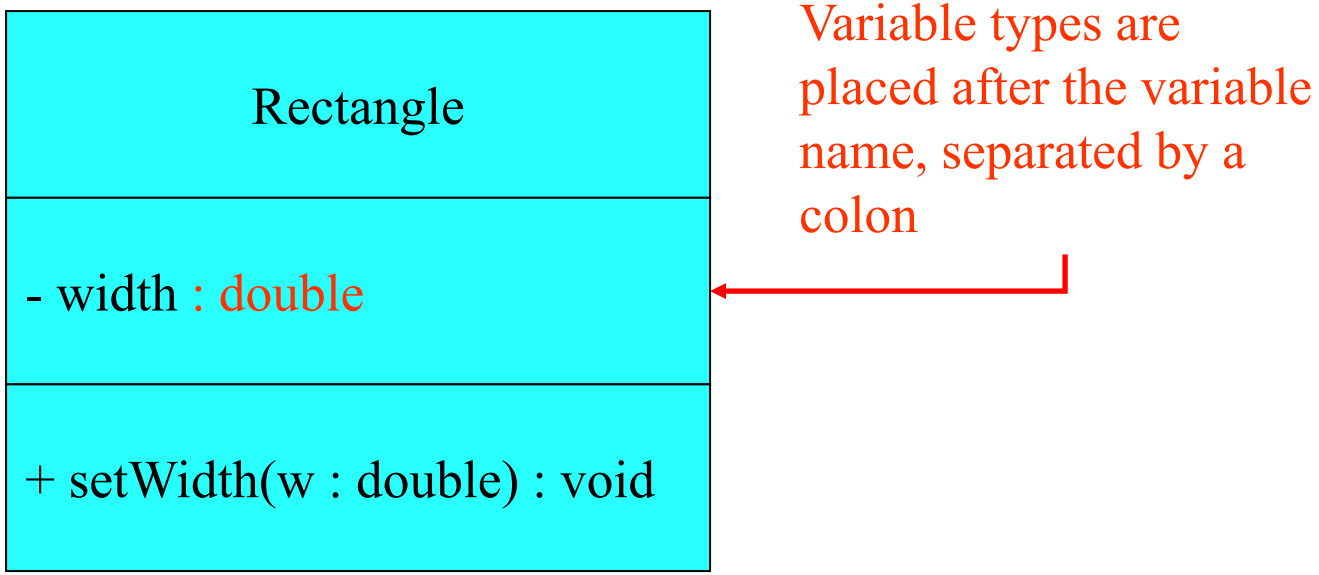
UML Data Type and Parameter Notation

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



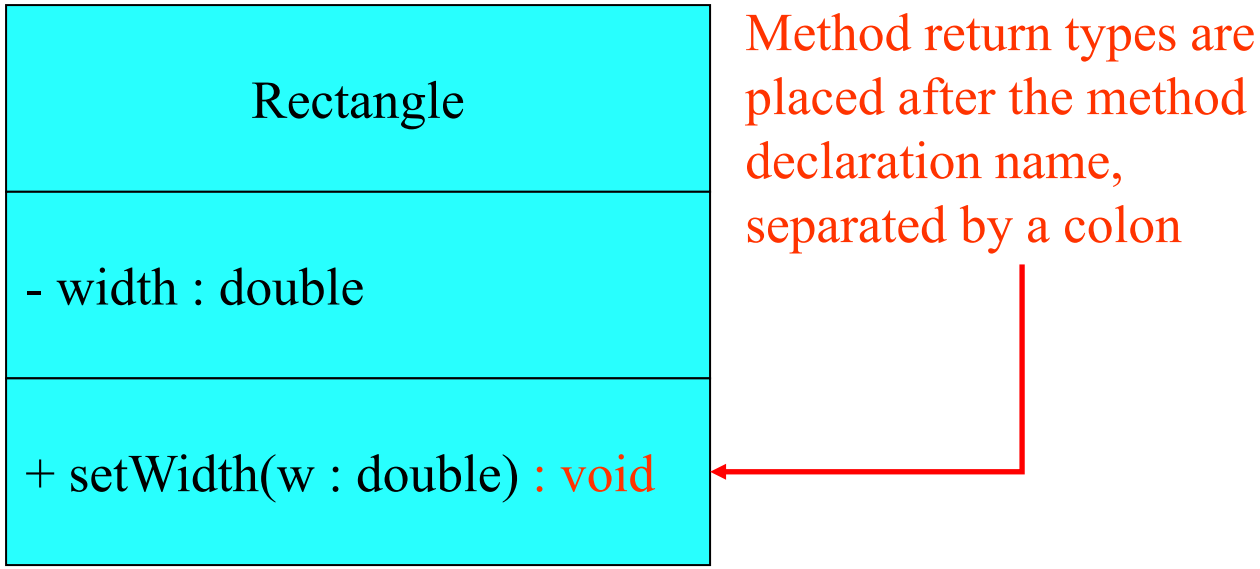
UML Data Type and Parameter Notation

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



UML Data Type and Parameter Notation

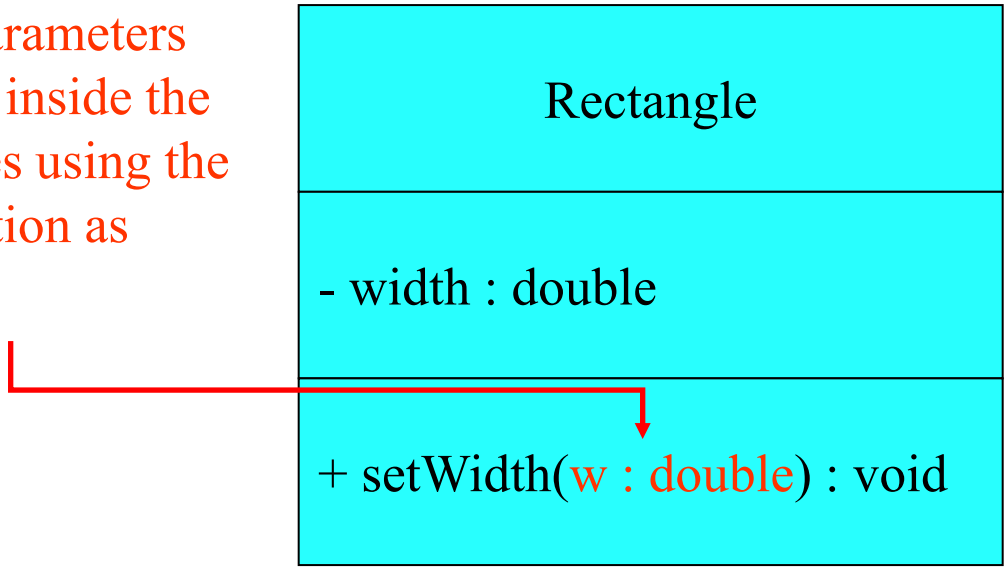
- UML diagrams are language independent
- UML diagrams use an independent notation to show return types, access modifiers, etc.



UML Data Type and Parameter Notation

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

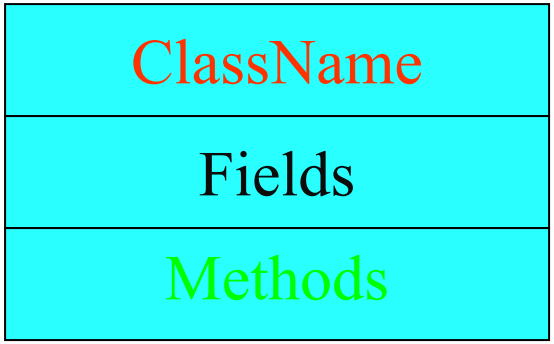
Method parameters are shown inside the parentheses using the same notation as variables



Converting the UML Diagram to Code

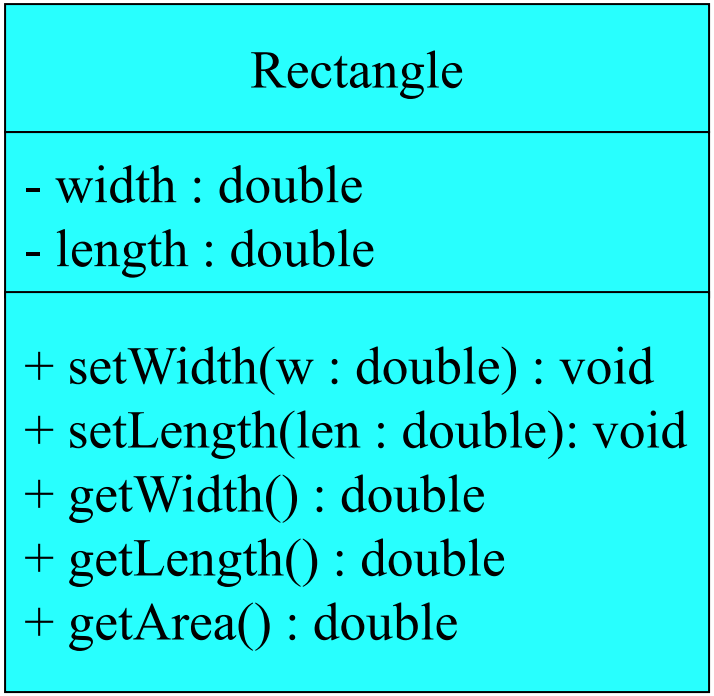
- Putting all of this information together, a Java class file can be built easily using the UML diagram
- The UML diagram parts match the Java class file structure

```
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

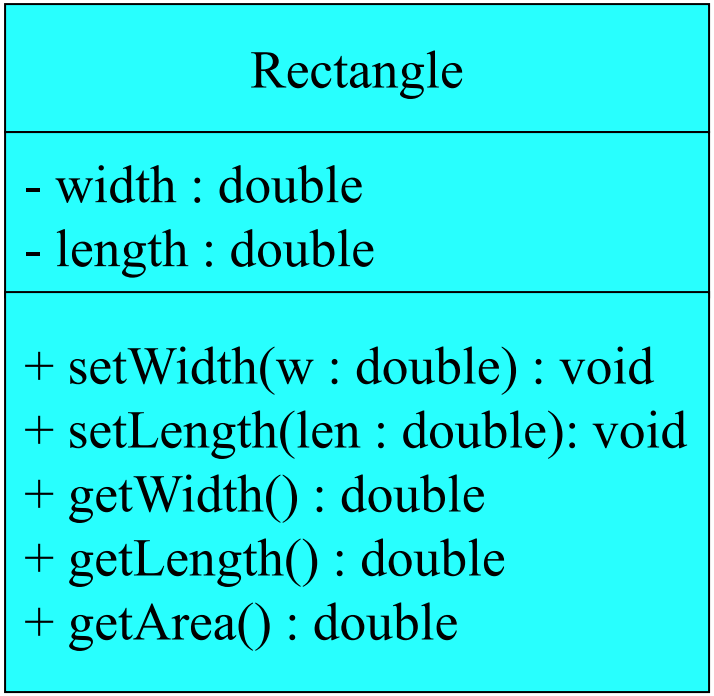



```
public class Rectangle
{
    private double width;
    private double length;

    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;
    }
}
```

Converting the UML Diagram to Code

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;
    }
}
```

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

Activity 4: Poll

In the header, the method name is always followed by this:

A) Parentheses

B) return type

C) data type

D) braces

Instance Fields and Methods

- Fields and methods that are declared as previously shown are called *instance fields* and *instance methods*
- Objects created from a class each have their own copy of instance fields
- Instance methods are methods that are not declared with a special keyword, `static`

Instance Fields and Methods

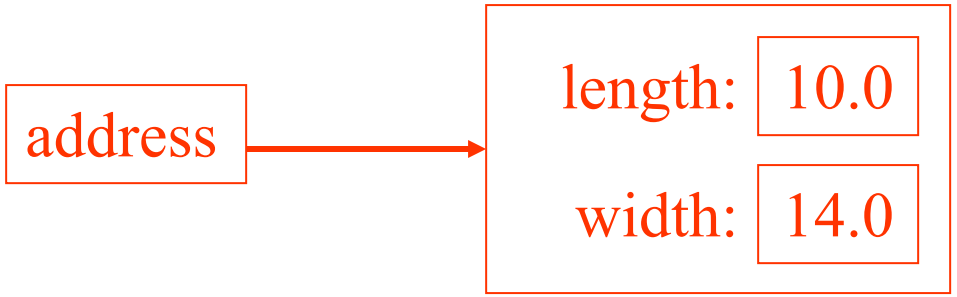
- Instance fields and instance methods require an object to be created in order to be used

- Note that each room represented in the example below can have different dimensions:

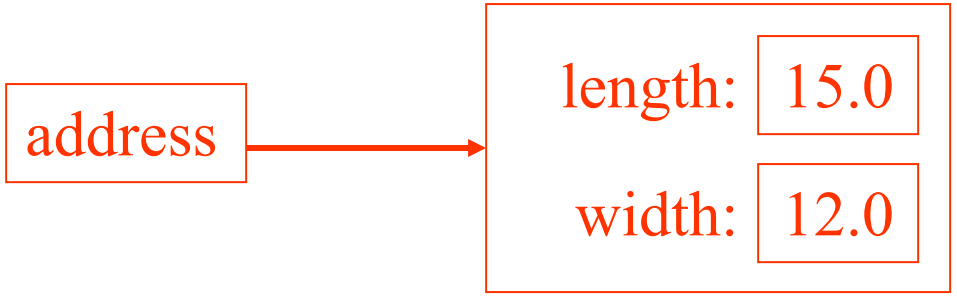
```
Rectangle kitchen = new Rectangle();
Rectangle bedroom = new Rectangle();
Rectangle den = new Rectangle();
```

States of Three Different Rectangle Objects

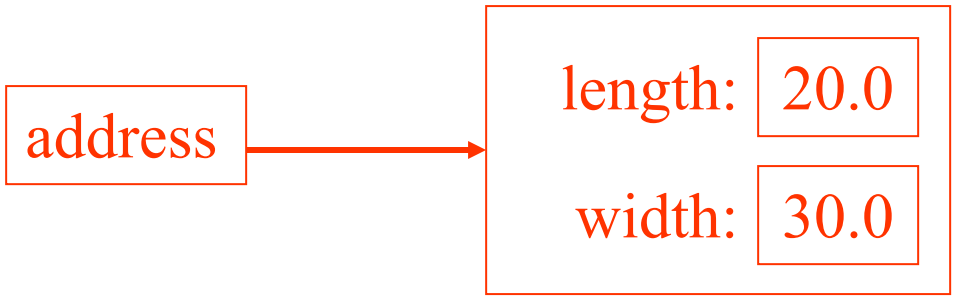
The `kitchen` variable holds the address of a `Rectangle` Object



The `bedroom` variable holds the address of a `Rectangle` Object



The `den` variable holds the address of a `Rectangle` Object



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 may not return any values
  - Constructors are typically public

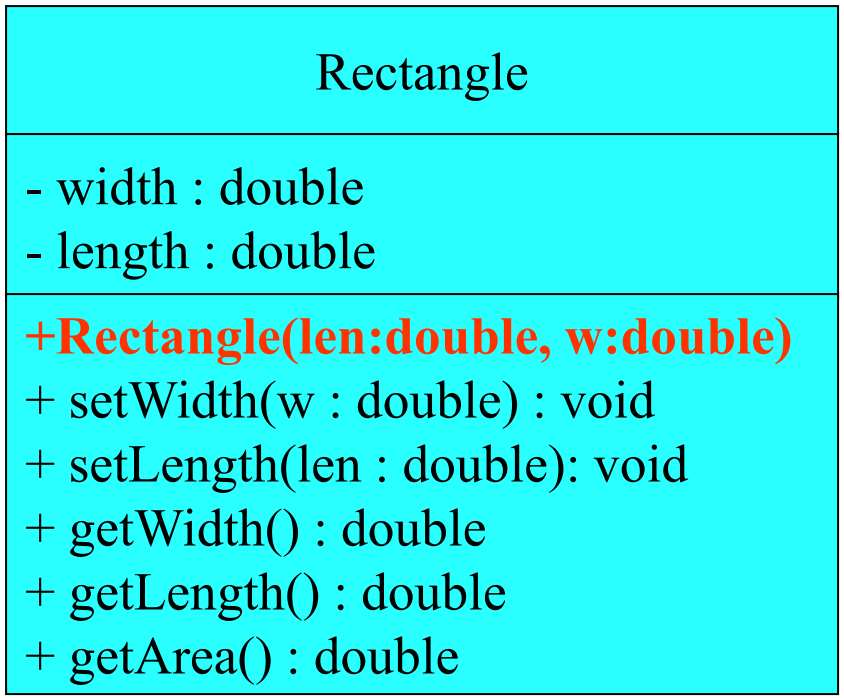
Constructor for `Rectangle` Class

```
/**
   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 (Phase 3): Rectangle.java, ConstructorDemo.java

Constructors in UML

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



Notice there is no return type listed for constructors.

Uninitialized Local Reference Variables

- Reference variables can be declared without being initialized

```
Rectangle box;
```

- This statement does not create a `Rectangle` object, so it is an uninitialized local reference variable
- A local reference variable must reference an object before it can be used, otherwise a compiler error will occur

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

- `box` will now reference a `Rectangle` object of length 7.0 and width 14.0

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
  - See example: First version of Rectangle.java
- A default constructor is not provided by Java if a constructor is already written
  - See example: Rectangle.java with Constructor

Writing Your Own No-Arg Constructor

- 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
- We can write our own no-arg constructor

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

The `String` Class Constructor

- One of the `String` class constructors accepts a string literal as an argument.
- This string literal is used to initialize a `String` object.
- For instance:

```
String name = new String("Michael
  Long");
```

The `String` Class Constructor

- This creates a new reference variable *name* that points to a `String` object that represents the name "Michael Long"
- Because they are used so often, `String` objects can be created with a shorthand:

```
String name = "Michael Long";
```

Summary of today's lecture

Objects and Classes

- Designing classes for the purpose of instantiating objects
- Class fields, methods and UML diagrams
- Identifying classes and their responsibilities within a problem domain
- The Constructor class

Activity 5:
Reflection Exercise

List any four concepts you have learnt in today's lecture

Activity 6: Homework Exercise

Write a program using Eclipse or NetBeans to implement any one concept you have learnt in today's lecture