COP 3330, Spring 2013

Intro to Object Oriented Programming - II

Instructor : Arup Ghosh
01-16-13

School of Electrical Engineering and Computer Science
University of Central Florida

Fundamentals of Object Orientation

- In non-object-oriented programming, a program is usually process-oriented or data-oriented.
- In such programs, there are typically data globally available and procedures globally available.
- The main program, or its subprograms, are in control and manipulate the data.

Fundamentals of Object Orientation

- In object-oriented programming, a program is partitioned into a set of communicating objects. Each object encapsulates all the behavior and knowledge relating to one concept.
- In this fashion, one can think of an OO program as having distributed control in that the "intelligence" (the ability to do things) and the "knowledge" (the data to be able to do those things), is distributed among the objects.
- When an object needs something from another object, it sends a message to the other object, which then performs some action and possibly returns a value to the caller.
  - The first object might even create the second object is no such object already exists. The second object, in turn, may need to communicate with other objects to help it accomplish its task.

Fundamentals of Object Orientation

- To start an OO program executing, you typically create a few objects and start them communicating with each other.
- In particular, this situation occurs when an object-oriented GUI (graphical user interface) is used as the HCI (human computer interface) in an application.
  - The windows, menus, and buttons are objects that need to be created first, and then those objects typically just sit there waiting for the user to interact with them, in which case they send messages to each other (and probably to other invisible objects) to accomplish the task.
  - GUI-based programming falls under the paradigm of event-driven programming.
  - We'll do GUI-based programming later in the semester.

Fundamentals of Object Orientation

- This view of OO programming, in which objects share the work and the responsibilities, should seem familiar in that it is the way humans typically interact with each other.
- One person, such as the owner of a business, does not do everything themselves. Instead, they assign tasks to their employees, each of whom is responsible not only for doing the assigned task, but for maintaining the data associated with that task.

- For example, a secretary might be responsible not only for typing papers, but also for storing the papers in appropriate filing cabinets. Furthermore, if the data in the files is confidential, the secretary might also be responsible for guarding the files and granting or denying access to the data. In the process of this work, the secretary may need to call on the help of other people in or out of the office.

Advantages of OO Programming

- One of the primary advantages of the OO approach compared to the non-OO approach is that, because the intelligence is distributed among objects, each of which maintains the data necessary to perform its tasks, it is easier to keep things in small manageable units and to understand how the units affect each other.
- In contrast, if every procedure is interacting with an arbitrary part of a global set of data, the effect of one procedure on all the others and on the system as a whole is harder to understand.
- Thus, the distributed nature of OO programming enhances the readability of the code.

Advantages of OO Programming

- An even bigger advantage of the OO approach is that a small change in the structure of the global data in a non-OO program may force a change to all the procedures that access that data.
- In contrast, a well-designed OO program has little global data and instead stores the data in objects mostly for their local use.
- Thus, making a change to the way data is stored in one class of objects often means that the only part of the program that needs to be changes is the code in that class.
- Similarly, if a programmer decides that a particular object is working too inefficiently, the programmer can redesign that object's behavior to be more efficient without affecting the rest of the system, thus supporting the maintainability of the software.

Advantages of OO Programming

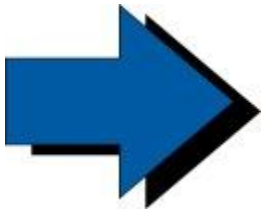
- Similarly, since each object has typically one small well-defined role and carries the data it needs with it, it is usually easier to reuse these objects in other situations.
- Thus, the use of OO programming techniques, if done well, increases the modifiability, readability, reusability, and maintainability of the software.

Object-Oriented Languages

- Programming languages support the OO paradigm if they have certain features that make it easier for the programmer to create objects and have them send messages to each other.
- A programming language is said to be object-oriented if it supports classes, objects, messages, inheritance, and (subtype) polymorphism.
  - Java is such a language, so too is C++.
  - C is not an OO language.

Classes and Objects

- Objects and classes are two fundamental concepts in the object-oriented software development.
- An object has a unique identity, a state, and behaviors. In real life, an object is anything that can be distinctly identified.
- A class characterizes the structure of states and behaviors that is shared by all of its instances.
- The terms object and instance are often used interchangeably.



REQUEST
FOR
A NEW **CAR**

CAR CLASS
"FACTORY"

INSTANCE
OF **CAR**

Classes and Objects

- The features of an object are the combination of the state and behavior of that object.
  - The state of an object is composed of a set of attributes (fields) and their current values.
  - The behavior of an object is defined by a set of methods (operations, functions, procedures).
- A class is a template for its instances. Instead of defining the features of objects, we define features of the classes to which these objects belong.

Recap - Objects in Java

- A class in Java can be defined as follows:

```
class Rectangle {
    private int length, width;
    public int area() {……}
    public void changeSizes(int x, int y)
     { …… }
```

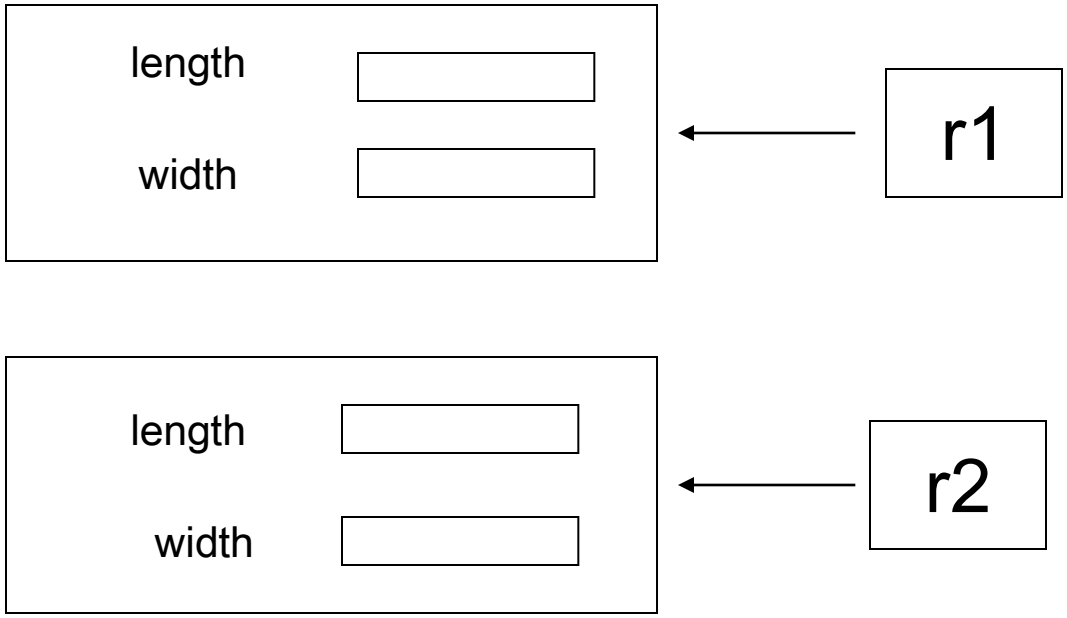
- The name of the class is: Rectangle. Its attributes are: length, width. Its methods are: area, changeSizes

- This Rectangle class is a template for all rectangle objects. All instances of this class will have same structure.

Objects in Java

- An object in Java is created from a class using the `new` operator.

```
Rectangle r1 = new Rectangle();
Rectangle r2 = new Rectangle();
```



A Good Design

… features Encapsulation

- This is a key aspect of object-oriented design

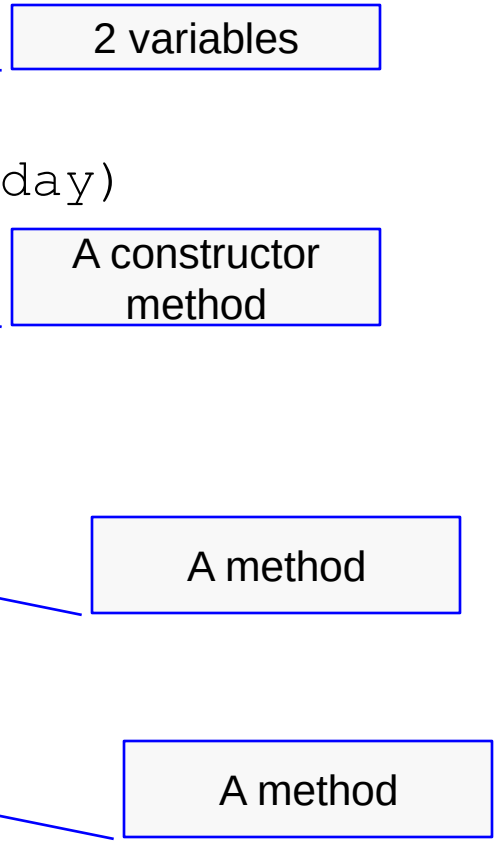
- Encapsulation means grouping together related data and operations on that data into a unit

Recap - Class Basics

- A class defines a new data type
- A class is a blueprint for objects
- Classes are mostly made up of three basic components (members)
  - Instance variables (fields)
  - Constructors
  - Methods

Recap - An Example Class in Java

```
public class Person
{   private String name;
    private Date birthdate;
    public Person (String who, Date bday)
    { this.name = who;
      this.birthdate = bday;
    }
    public String getName()
    {   return name;
    }
    public Date getBirthdate()
    {   return birthdate;
    }
 }
```



A Good Design

… features Information Hiding

- This is a key aspect of object-oriented design

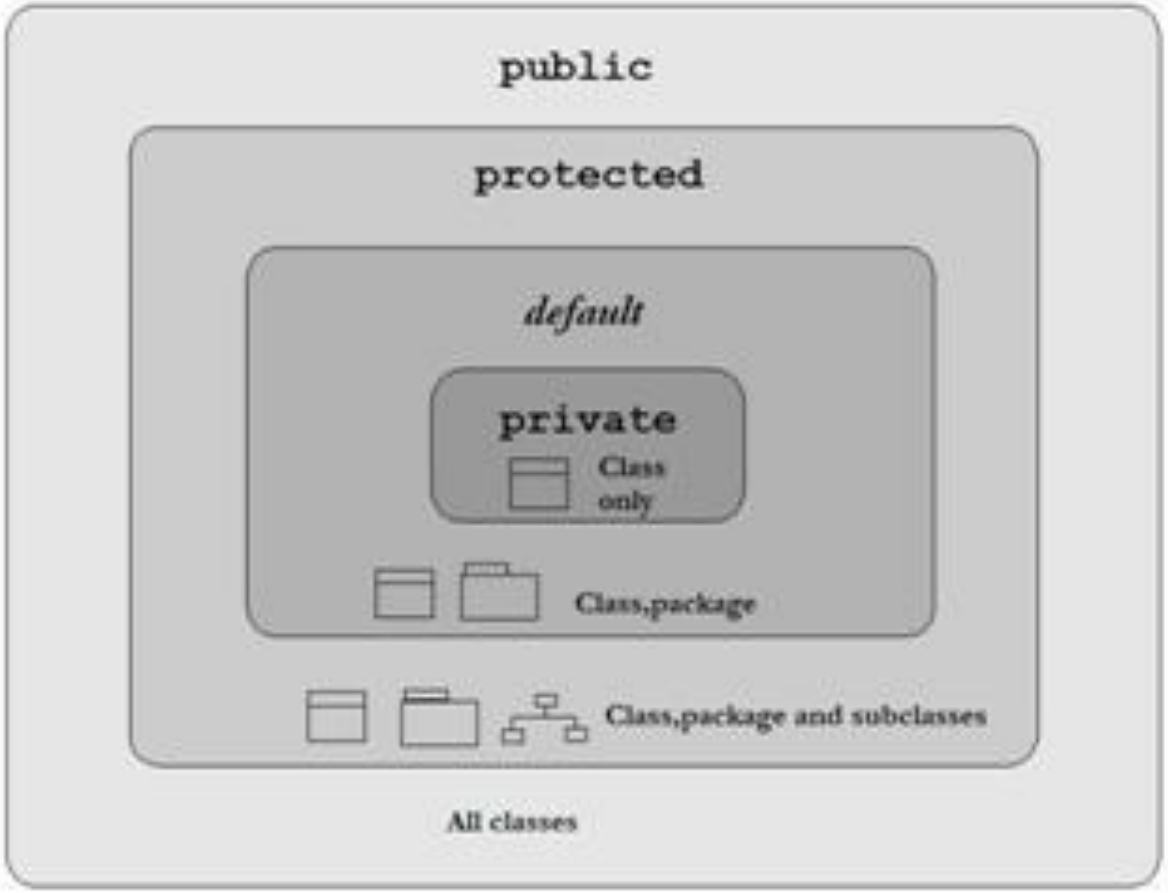
- Information hiding means concealing unnecessary details
- This allows those using a system to more easily understand the big picture
- This makes it much more difficult to use the system in incorrect ways

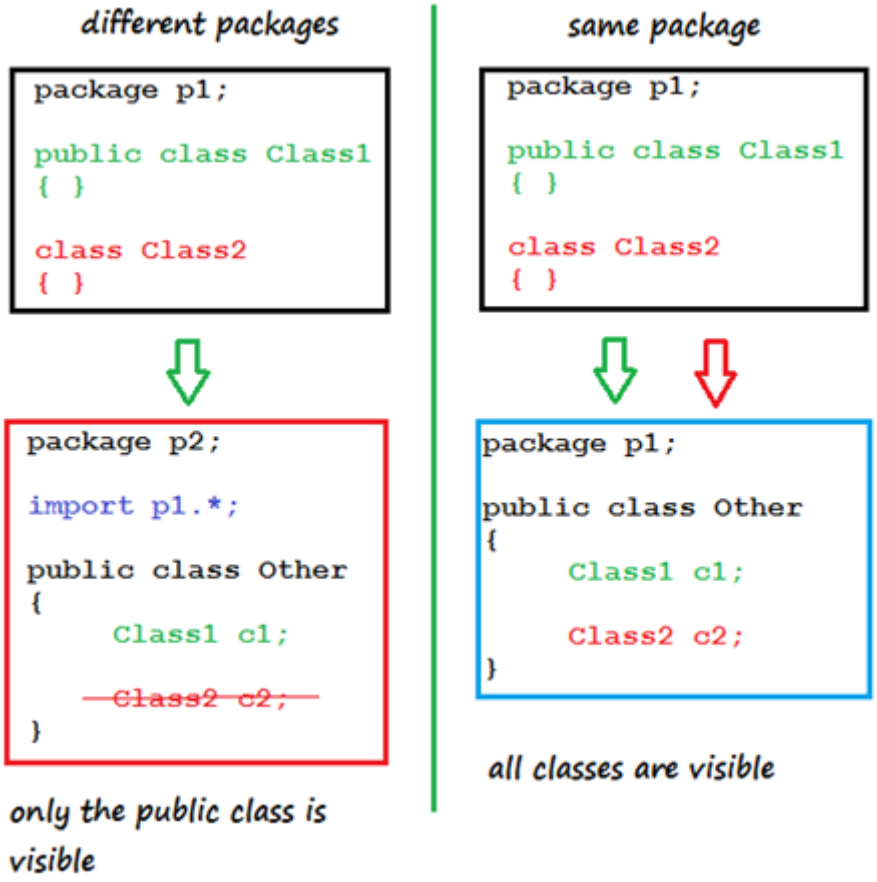
Recap – Class: Access Modifiers

- All members have access modifiers
- The most common are
  - `public`
  - `private`
- The other two will be discussed at a later point
  - `protected`
  - Default access (package)

Instance Variables

- Instance variables persist throughout the life of the object
- They can be of the same types as any other variables
- Example
  - `private double realPart;`
  - `private double imaginaryPart;`


public
protected
default
private
Class
only
Class,package
Class,package and subclasses
All classes


different packages
package p1;
public class Class1
{ }
class Class2
{ }
package p2;
import p1.*;
public class Other
{
Class1 c1;
Class2 c2;
}
only the public class is visible
same package
package p1;
public class Class1
{ }
class Class2
{ }
package p1;
public class Other
{
Class1 c1;
Class2 c2;
}
all classes are visible

Constructor

- A constructor is a method, but it is so important that it needs separate consideration
- A constructor creates, initializes, and returns a reference to a new object
- Creation and return are handled automatically
- The constructor always bears the name of its class

Constructor

- Example

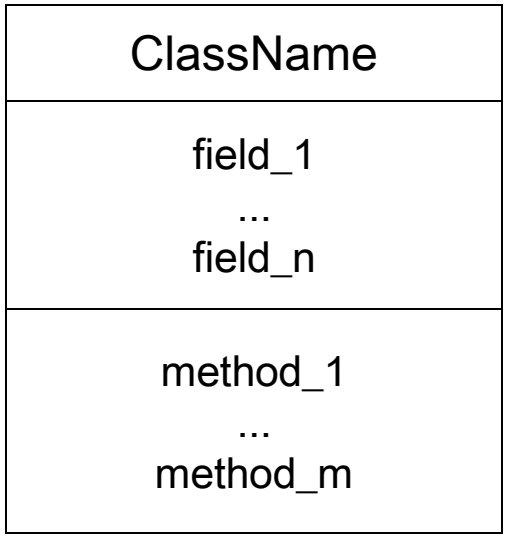
```
public class Complex
{
    private double realPart;
    private double imagPart;

    public Complex()
    {
          realPart = imagPart = 0;
    }

    public Complex(double r, double i) {
          realPart = r;
          imagPart = I;
    }
}
```

Constructor

- Constructors are called by using the `new` keyword
  - Example:

```
Complex foo = new Complex();
Complex foo2 =
 new Complex(1.0,-1.0);
```

Graphical Representation of Classes: Intro to UML



ClassName is the name of the class

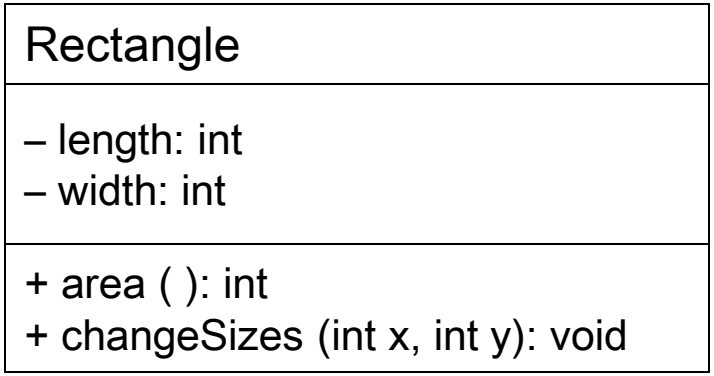
Each field is

[Visibility] identifier [Type] [=initial value]

Each method is

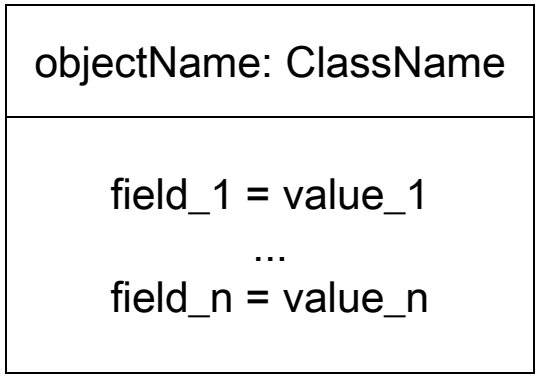
[Visibility] identifier (parameter-list) [Type]

Example:

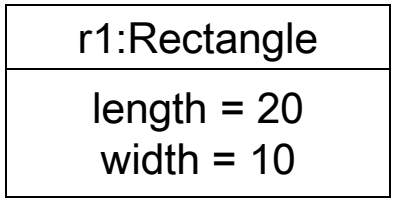


- We may not give field, methods parts

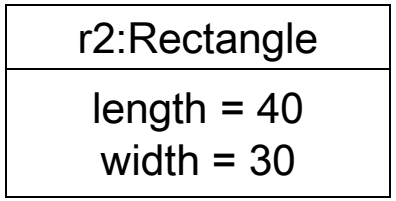
Graphical Representations of Objects



- We may omit ClassName, and just use objectName. In this case the class of the object is no interest for us.
- We may omit objectName, and just use :ClassName. In this case, the object is an anonymous object.




```
Rectangle r1 = new Rectangle();
r1.length = 20;
r1.width = 10;
```




```
Rectangle r2 = new Rectangle();
r2.length = 40;
r2.width = 30;
```