COP 3330: Object-Oriented Programming
Summer 2011

Introduction to Object-Oriented Programming – Part 2

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3330/sum2011

Department of Electrical Engineering and Computer Science
Computer Science Division
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.
  - That is, each part of the program goes to the global data, gets part of it, manipulates it, and then, if necessary, saves any changes to the data.
  - One can think of the main program, through its subprograms as having all the “intelligence” or behavior in the program and the data has none of the intelligence. In this case, the main program and its subprograms are responsible for everything.

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, doesn't 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

- A class can be understood from a modeling perspective as well as a programming language perspective.
- When designing a software application, classes model abstract concepts that play an important role in the system with well-defined responsibilities and relationships with other classes.
- In an OO language, classes can be viewed as templates (i.e., a blueprint) for objects that describe a certain type of behavior or a certain set of responsibilities and any associated data (characteristics or attributes).

Classes

- As an example, consider a job description for a secretary or a police officer. The job description indicates the responsibilities of any person filling one of those roles.
- Individual secretaries have their own data (e.g. files, desks, bosses) to maintain, but they all have similar responsibilities for handing that data. In the same way, all police officers in a given precinct will have similar responsibilities but will have individual differences in the data involved in their work, for example, their name, which section of the precinct they are to patrol, who their partners are, and which patrol cars they will use.

Objects

- In an OO language, an object is an instance of a class. An object is similar to an individual secretary or police officer.
- An object's associated class defines the type of data the object maintains and its behaviors or responsibilities toward that data.
- As with individual secretaries, individual objects have their own set of data (their own state) to maintain.

Objects

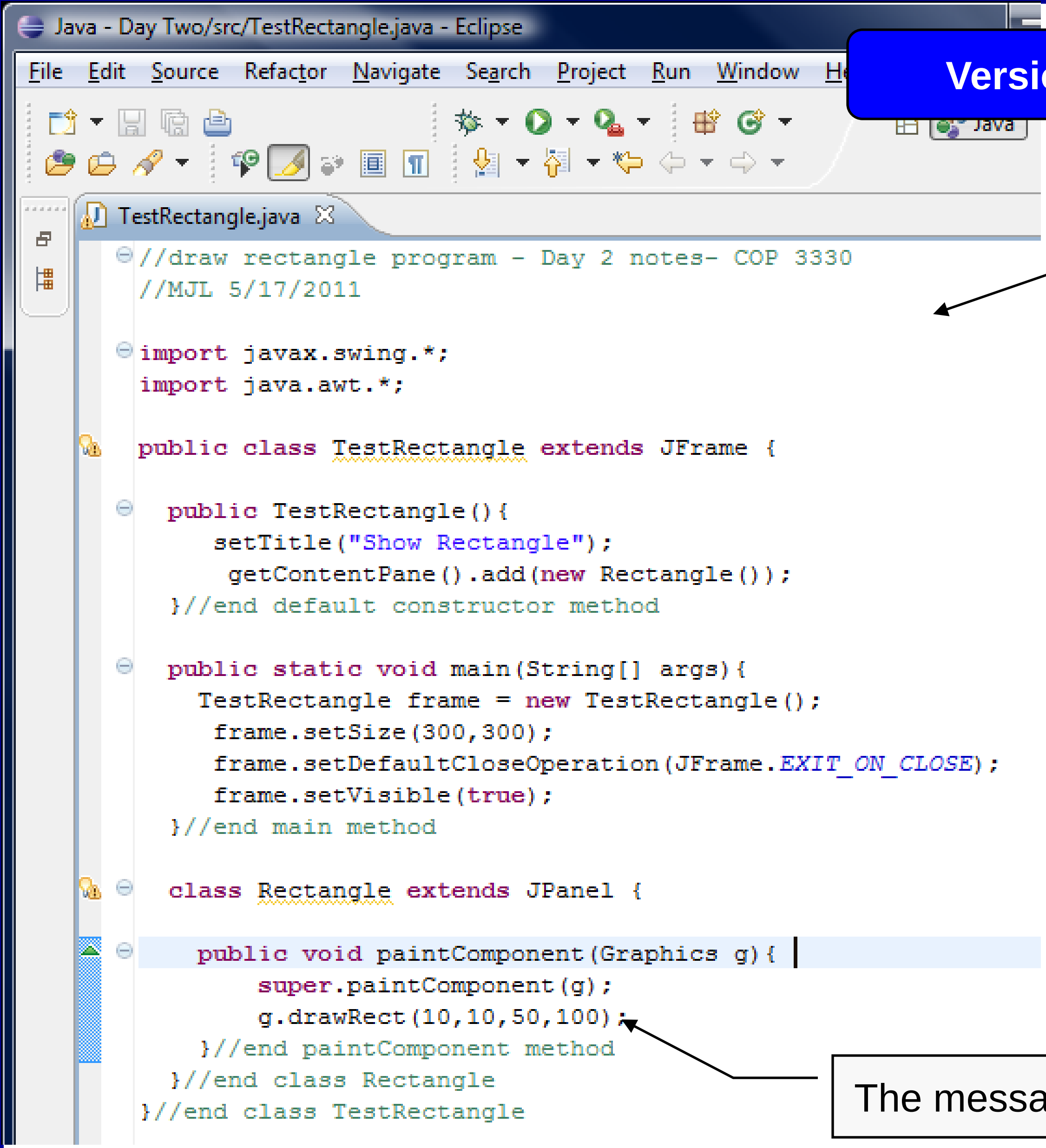
- The way that objects communicate and get each other to perform some action is by sending messages to each other.
- By sending a message to another object, the first object causes the second object to execute some code. That code is actually a procedure – which in object-oriented languages is called a method – associated with the second object.
  - Message sending is actually a request (or command) from one object to another object to execute one of its methods.
- Through this mechanism, objects can be thought of as servers that provide a service to any client who asks (by sending them a message).

Objects

- For example, a Graphics object `g` (in the `java.awt` package) is an object designed to do you the service of drawing shapes, among other things, in visual components such as windows. For example, it can draw a rectangle for you in its associated component at whatever coordinates and of whatever size you want. To get it to do so, you just send it a message such as:

```
    g.drawRect(10, 10, 50, 100);
```

  and `g`'s `drawRect` method will get executed, causing a rectangle with upper left level corner at coordinates (10, 10) and with width 50 and height 100 to be drawn in the component.

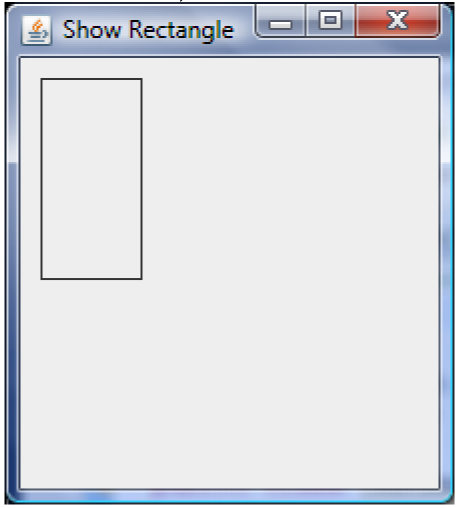

Java - Day Two/src/TestRectangle.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window He
TestRectangle.java

```
//draw rectangle program - Day 2 notes- COP 3330
//MJL 5/17/2011

import javax.swing.*;
import java.awt.*;

public class TestRectangle extends JFrame {

  public TestRectangle(){
      setTitle("Show Rectangle");
       getContentPane().add(new Rectangle());
  }//end default constructor method

  public static void main(String[] args){
    TestRectangle frame = new TestRectangle();
     frame.setSize(300,300);
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setVisible(true);
  }//end main method

  class Rectangle extends JPanel {

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawRect(10,10,50,100);
    }//end paintComponent method
  }//end class Rectangle
}//end class TestRectangle
```

Version 1

The code

The results



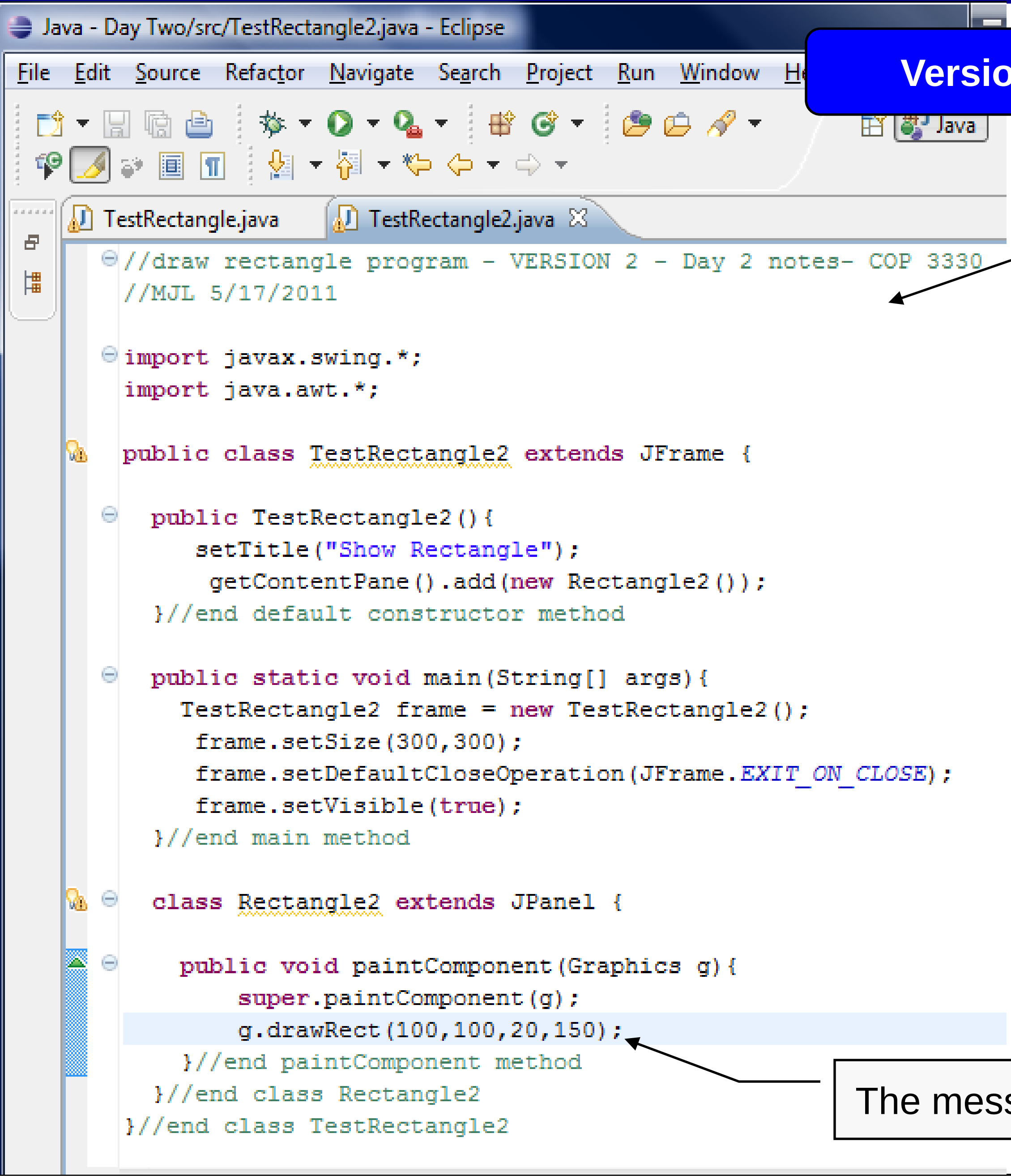
The message

Version 2

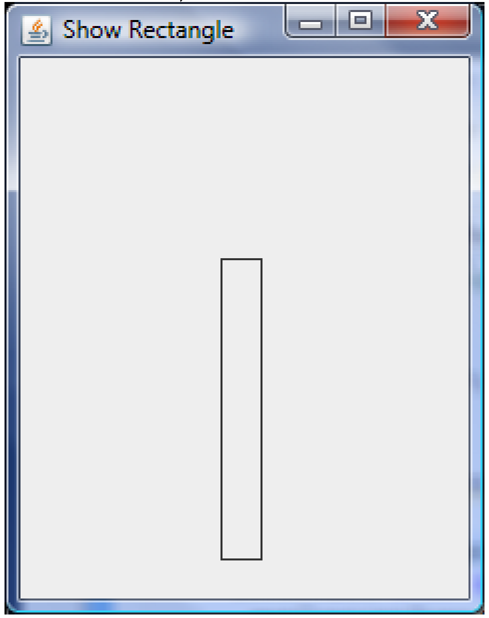
The code

The results

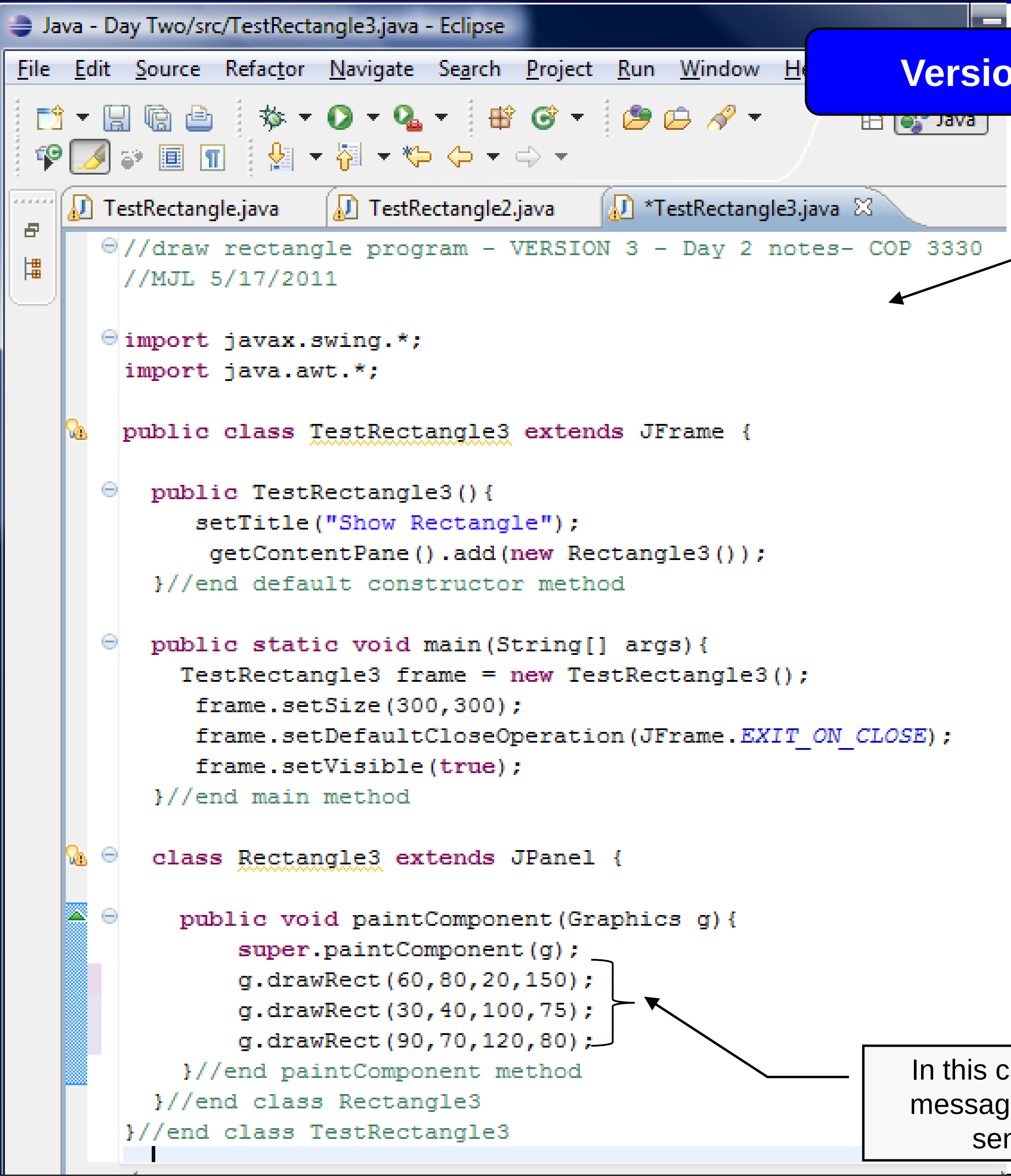
The message


Java - Day Two/src/TestRectangle2.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window He
Java
TestRectangle.java
TestRectangle2.java

```
//draw rectangle program - VERSION 2 - Day 2 notes- COP 3330
//MJL 5/17/2011

import javax.swing.*;
import java.awt.*;

public class TestRectangle2 extends JFrame {

  public TestRectangle2(){
      setTitle("Show Rectangle");
       getContentPane().add(new Rectangle2());
  }//end default constructor method

  public static void main(String[] args){
    TestRectangle2 frame = new TestRectangle2();
     frame.setSize(300,300);
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setVisible(true);
  }//end main method

  class Rectangle2 extends JPanel {

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawRect(100,100,20,150);
    }//end paintComponent method
  }//end class Rectangle2
}//end class TestRectangle2
```



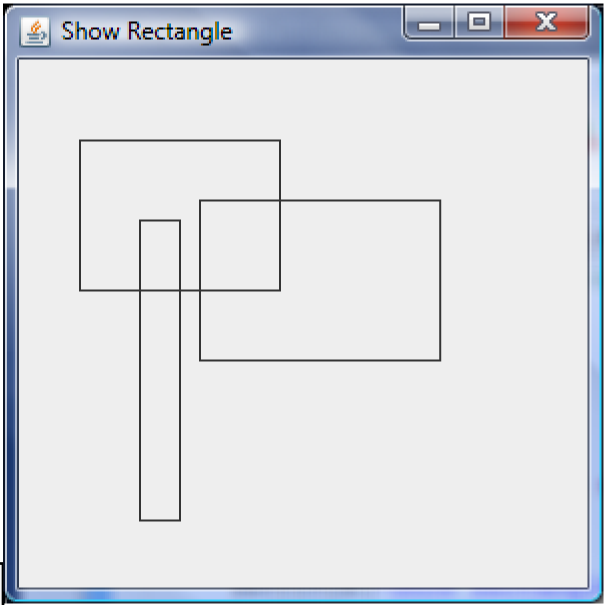
Version 3


Java - Day Two/src/TestRectangle3.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window He
TestRectangle.java
TestRectangle2.java
*TestRectangle3.java

```
//draw rectangle program - VERSION 3 - Day 2 notes- COP 3330
//MJL 5/17/2011

import javax.swing.*;
import java.awt.*;

public class TestRectangle3 extends JFrame {

  public TestRectangle3(){
     setTitle("Show Rectangle");
      getContentPane().add(new Rectangle3());
  }//end default constructor method

  public static void main(String[] args){
    TestRectangle3 frame = new TestRectangle3();
     frame.setSize(300,300);
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setVisible(true);
  }//end main method

  class Rectangle3 extends JPanel {

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawRect(60,80,20,150);
        g.drawRect(30,40,100,75);
        g.drawRect(90,70,120,80);
    }//end paintComponent method
  }//end class Rectangle3
}//end class TestRectangle3
```

The code

The results



In this case, 3 messages are sent

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. An object is an instance of its class.

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.

Classes in Java

- For the moment we'll ignore the optional parts of a Java class definition.
- A class is defined in using the keyword `class` followed by a class name and braces surrounding the declaration and implementation of the methods of the class and the declaration of the variables storing the data of objects of the class.
- For example, the Java code shown on the next page defines a class Person that stores a name and birth date. It has two variables, a constructor and two methods.

Java Convention

Class names should be nouns, in mixed case with the first letter capitalized and the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words – avoid acronyms and abbreviations (unless the abbreviations is much more widely used than the long form, such as URL or HTML. Examples: House, HousePlans, ObjectViewOrientation

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;
   }
 }
```

2 variables

A constructor method

A method

A method

Classes in Java

- The non-optional parts of a method declaration include: a return type, the method name, the list of parameters in parentheses, and then the body of the method encloses in braces.
- A variable declaration includes the type of the variable, the name of the variable, and an optional initial value.
- Notice how this class actually uses two other classes, namely `String` (in the `java.lang` package) and `Date` (in the `java.util` package) for storing the data for this class.

Java Convention

Method names should be verbs, in mixed case with the first letter lowercase and the first letter of each internal word capitalized.

Examples: getHouseName(), setObjectColor();

Classes in Java

- A user of the Person class would typically construct a Person object (an instance) using the constructor method:

```
Person firstPerson = new Person("Suzi", new Date(1000000000));
```

- Then the user would send the object a message such as:

```
    String firstPersonName = firstPerson.getName();
```

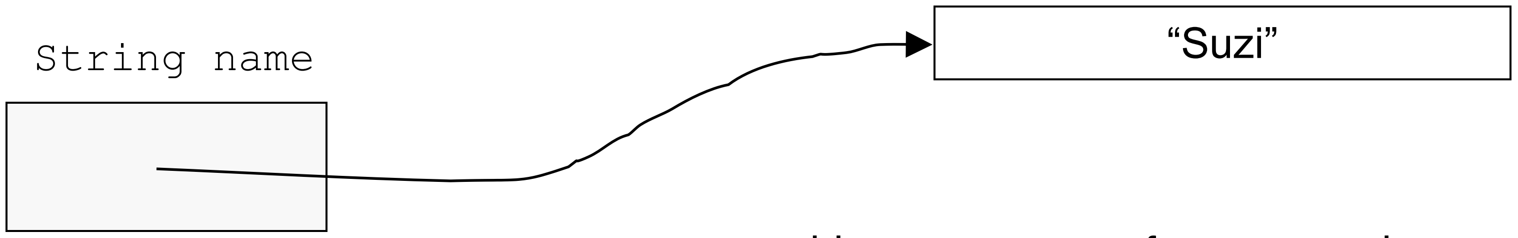
Classes in Java

- In order to execute their methods properly, most objects need to store data. This data is stored in instance variables (also called fields), which, in Java, are declared in the body of the class declaration, but outside the body of any method or constructor.
- Instance variables are different than local variables, which are variables declared in method or constructor bodies, in that local variables exist and store data only during execution of the body in which they were declared whereas instance variables exist and store data during the lifetime of the object.
- Therefore, instance variables provide state information for the objects.

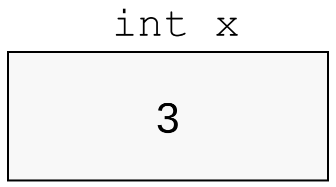
Classes in Java

- In our previous example, a Person object has an instance variable called `name` that stores a reference to the String object containing the person’s name. This String forms part of the state of the Person object.
- In Java, only variables of a primitive type actually store their data in the variable. For all variables of an object type, the variables store a reference to the data.
  - Its common to visualize the reference as a pointer from the variable to the data (see next page), although the reference can actually be implemented through means other than direct pointers.

Classes in Java



object types use references to data



primitive types store data directly

Classes in Java

- The `public` and `private` accessibility keywords (modifiers) in front of the methods, variables and class declarations restrict the objects that can access the objects of the Person class and send them messages.
- We'll look at this in much greater detail later, but in short:
  - public classes are accessible by any other class, public methods can be invoked on an object by any other object, and public variables of an object can be accessed (read and written) by any other object.
  - private methods and instance variables can be invoked or accessed only within the class body itself.

Classes in Java

- Because two objects of the same class have the same set of methods and therefore, can be sent the same set of messages, you might initially think that all of the objects of that class behave identically, in which case there would be little reason to ever create more than one object of a class.
- What distinguishes the behavior of two objects of the same class is their state. The two objects may behave differently because the two objects' instance variables may have different values.
- Consider the following example:

Classes in Java

- Suppose we create two Person objects:

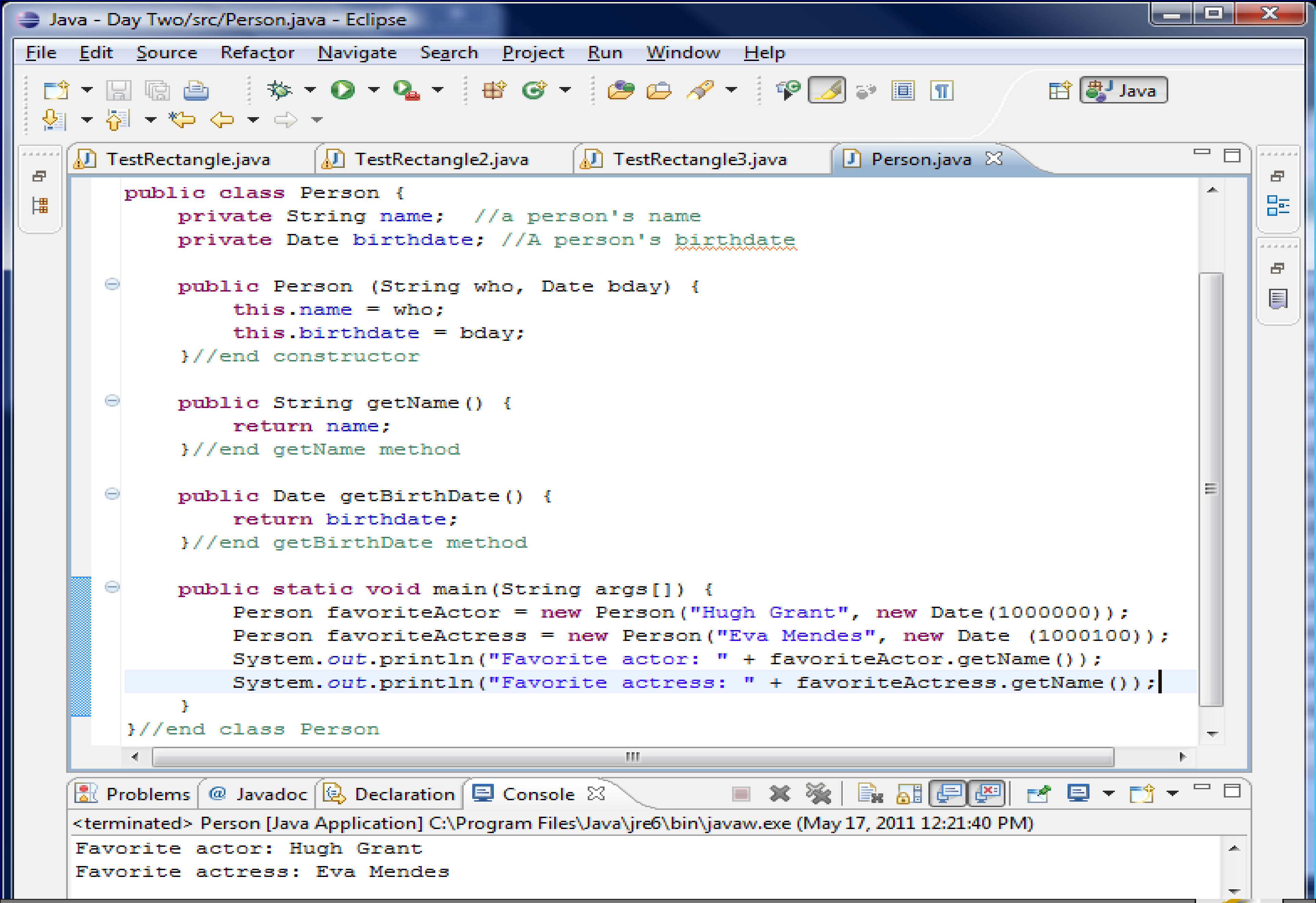
```
Person favoriteActor = new Person(“Hugh Grant”, new Date(1000000));
Person favoriteActress = new Person(“Eva Mendes”, new Date(100000));
```

- Now both object can respond to the getName message as in:

```
String hisName = favoriteActor.getName();
String herName = favoriteActress.getName();
```

They will however, return different values (see next page).

- Thus, the result of a method call may vary depending on the state of the object, these methods are also called instance methods.


Java - Day Two/src/Person.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
TestRectangle.java
TestRectangle2.java
TestRectangle3.java
Person.java
Problems
@ Javadoc
Declaration
Console

```
public class Person {
    private String name;  //a person's name
    private Date birthdate; //A person's birthdate

    public Person (String who, Date bday) {
        this.name = who;
        this.birthdate = bday;
    }//end constructor

    public String getName() {
        return name;
    }//end getName method

    public Date getBirthDate() {
        return birthdate;
    }//end getBirthDate method

    public static void main(String args[]) {
        Person favoriteActor = new Person("Hugh Grant", new Date(1000000));
        Person favoriteActress = new Person("Eva Mendes", new Date (1000100));
        System.out.println("Favorite actor: " + favoriteActor.getName());
        System.out.println("Favorite actress: " + favoriteActress.getName());
    }
}//end class Person
```

```
<terminated> Person [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 17, 2011 12:21:40 PM)
Favorite actor: Hugh Grant
Favorite actress: Eva Mendes
```

Class Methods and Class Variables in Java

- Java also includes objectless variables and objectless methods called class variables and class methods.
  - In fact, by using the objectless features of Java, it is possible to write a program that is almost completely non-OO. We're not going to do that, but it is important to know when and when not to use these features of Java.
  - Another way of thinking about class methods, is that they are messages that can be sent to the class itself instead of to an object belonging to that class, and class variables can be thought of as the state of the class itself.
- Both class variable and class methods are declared in Java using the keyword `static`.

Class Variables in Java and Their Uses

- Class variables can be thought of as variables that are shared among all the objects of a given class (and among objects outside of the class is the variable is made public) and so such variables cannot have unique values for each object of the class.
- The typical use of class variables is for defining constants. Such constants are not only shared among all objects of the class, but are often made publicly available to be shared among all object and classes in a program.
- In Java, a constant is indicated by the keyword `final`.

Class Variables in Java and Their Uses

- As an example of a class variable, a programmer might define the physical constant `c` representing the speed of light in a vacuum in a class PhysicalConstants by including in the body of that class a line like:

```
public final static int c = 299792458;
```

- Since it is declared public and static, this constant can be accessed by any object at any time independently of any objects of the class PhysicalConstants.

- A Java program could refer to this constant by using the notation: PhysicalConstants.c as in:

```
double distance = PhysicalConstants.c *40;
```

which would compute the distance in meters that light travels in 40 seconds.

Class Variables in Java and Their Uses

- In addition to use in defining constants, class variables can be used to allow all instances (all objects of the class) to share a piece of data.

- For example, suppose all objects of a class need to know how many objects of that class have been created. You could give each object a copy of that data or give each object a reference to that data, but in either case, not only is space wasted because of the duplication, but if the data is changed in one object, it becomes necessary to update the value in all the objects of the class. This would obviously be a waste of time and may easily lead to errors if one object is accidently missed. A much better solution is to make the count a static variable in the class to be shared by all the objects.

Class Methods in Java and Their Uses

- Class methods can be thought of as methods that are not a form of message passing to objects of the class, but rather can be invoked independently of any objects of the class.
- In general, class methods are useful when objects of that class are stateless (i.e., have no instance variables) or when some of the methods of the class do not use the state of the objects but instead merely manipulate the data passed in parameters.
- For example, all the methods in the Math class are public class methods (they have public and static modifiers in their declarations) and so can be accessed and executed by any body of Java code without reference to Math objects. For example, the `sin` method in the Math class is executed as:

```
double y = Math.sin(x);
```

Class Methods in Java and Their Uses

- It is appropriate that these methods are class methods because they perform mathematical operations on the arguments passed to those methods, and as such, a Math object would play no significant role.
- When designing a class and figuring out what methods it should have, it is not always immediately obvious whether a method should be a class method or an instance method.
- For example, suppose you are defining a Set class (different from the `java.util.Set` interface), objects of which behave like finite unordered mathematical sets of integers. A natural operation to be performed on such a set is the intersection of it with another set.
- There are (at least) two ways such an operation can be declared in the Set class:

Class Methods in Java and Their Uses

```
Public Set intersect (Set otherSet)

Public static Set intersect (Set firstSet, Set secondSet)
```

- In the first case, the user would get the intersection by sending a Set object `s1` a message asking it to return the intersection of itself and a second Set object `s2`, though a method call such as:

```
    Set intersection = s1.intersect(s2);
```

- In the second case, the user could find the intersection by calling the class method, passing both sets as parameters:

```
    Set intersection = Set.intersect(s1,s2);
```

- Which version is better?

Class Methods in Java and Their Uses

- One advantage of the second version is that it displays the natural symmetry in the intersection operation, in which neither set plays a special role. Also, the second version will not necessarily fail with a `NullPointerException` if `s1` or `s2` happened to be null. In other words, the `intersect(Set, Set)` class method could test the nullity of `s1` and `s2` and treat a null `s1` or `s2` as an empty set and simply return an empty set. In contrast, the instance version (verison 1) will throw an exception before the `intersect(Set)` instance method even begins execution if `s1` is null.

Class Methods in Java and Their Uses

- However, an advantage of the first version (the instance method) is that it is a natural way to proceed from an OO perspective.
- That is, it is natural to think of asking a Set object to tell you what it has in common with another set.
- Also, unless the intersect(Set) instance method is declared final, it can be overridden by sublcasses of Set, which, although not obviously useful here, is a feature that future users might find very valuable. We'll discuss overriding later.

Objects in Java – An Example

- 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)
    { ……… }
```

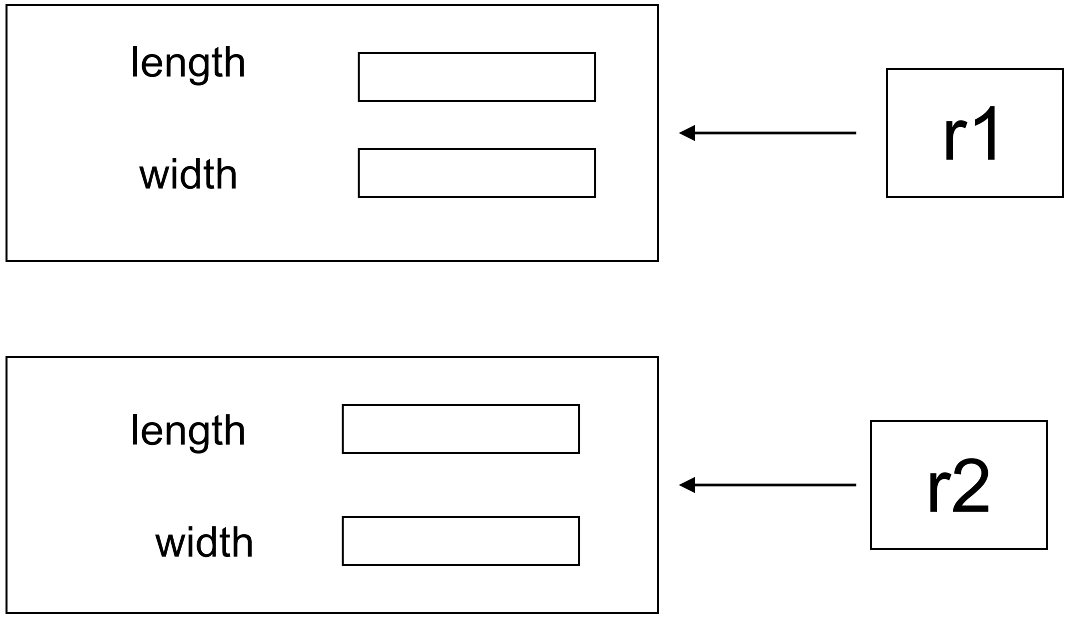
- The name of the class is: Rectangle. Its attributes are: length, width. Its methods are: area, changeSizes

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

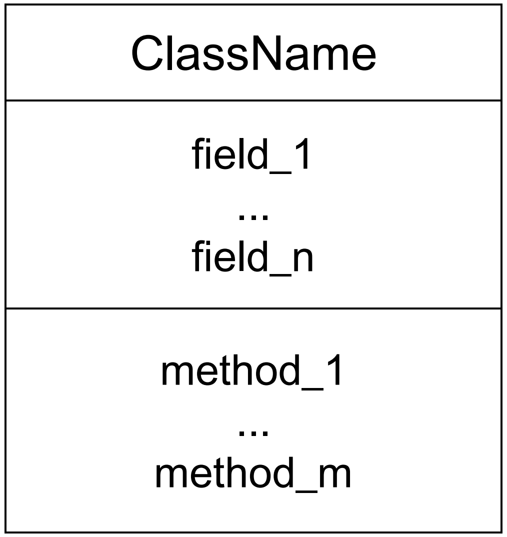
Objects in Java – An Example

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

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



Graphical Representation of Classes Using UML



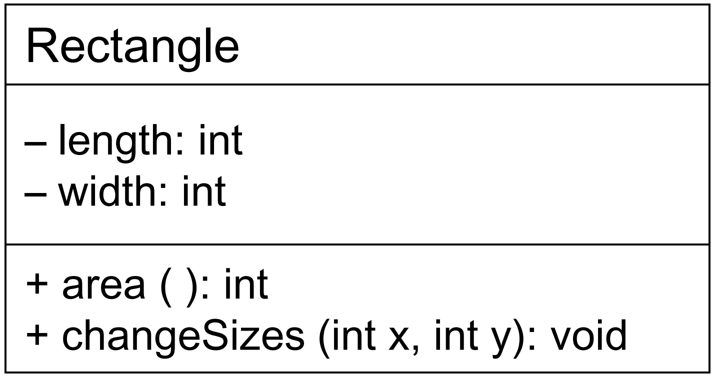
ClassName is the name of the class

Each field is
    [Visibility] identifier [Type] [=initial value]

Each method is
    [Visibility] identifier (parameter-list) [Type]

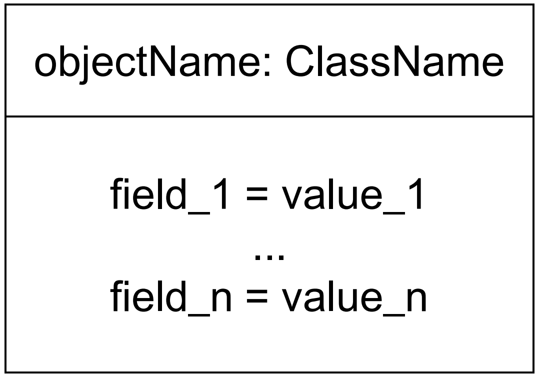
Visibility/Accessibility modifiers:
    − indicates private,   + indicates public,
    # indicates protected, ~ indicates package

Example:

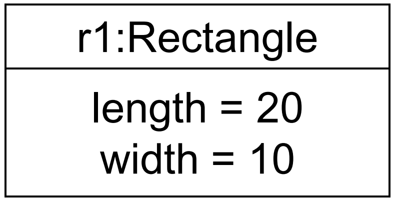


- We may not specify field, methods parts in complex abstract diagrams

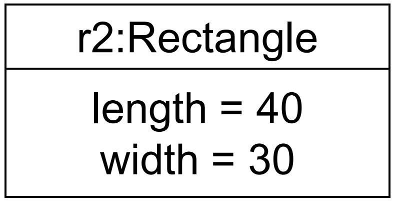
Graphical Representation of Classes Using UML



- 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.



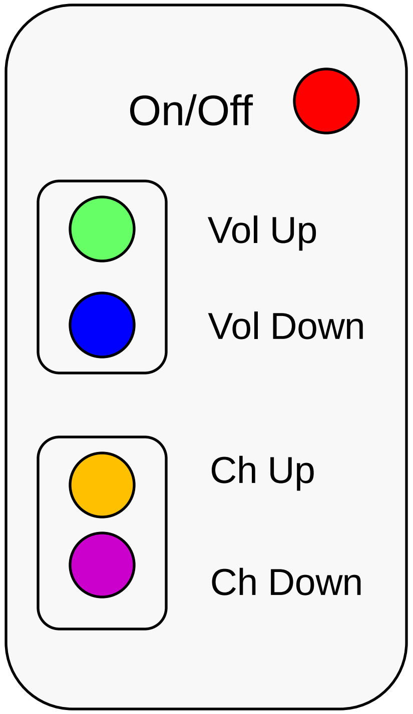
```
Rectangle r1 = new Rectangle();
r1.length = 20;
r1.width = 10;
```



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

DAY 2 – Practice Problem 1

- Using the basic UML notation introduced on the previous few pages, construct a UML class diagram for a basic TV remote control. Name the class `RemoteControl`. The class has three attributes (fields), current channel (integer), current volume (integer), and current state of the TV (on or off). The class will have five methods (not counting the constructor which is often omitted from UML diagrams). These five methods allow for increasing and decreasing the volume by 1 unit, increasing and decreasing the channel by 1 unit, and switching the TV on or off.
- We'll look at the solution next class.


On/Off
Vol Up
Vol Down
Ch Up
Ch Down