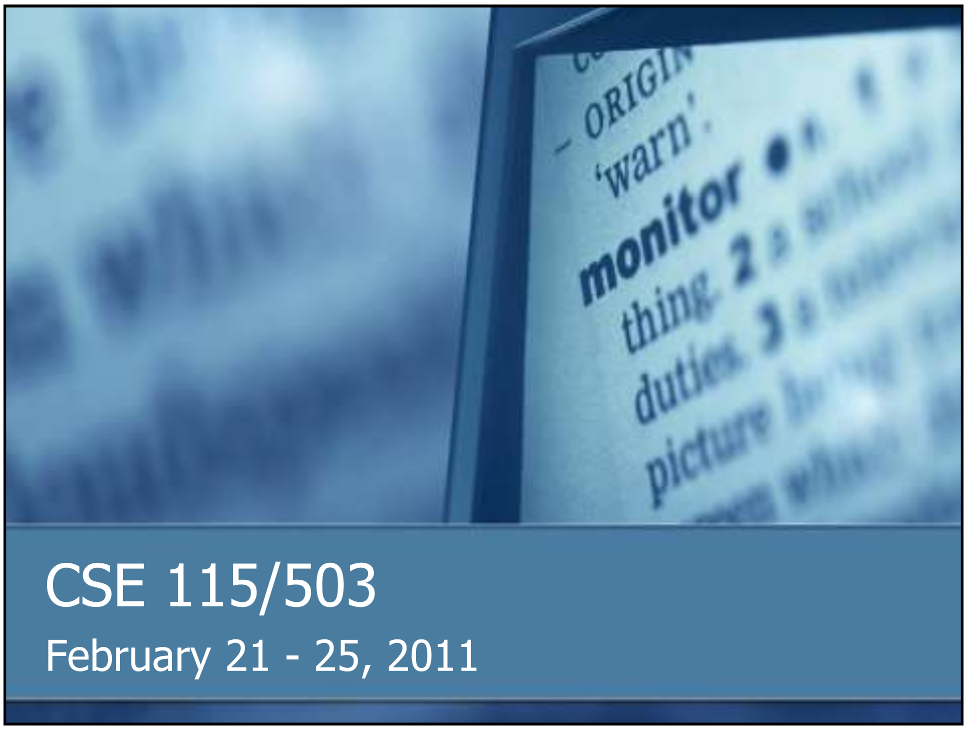
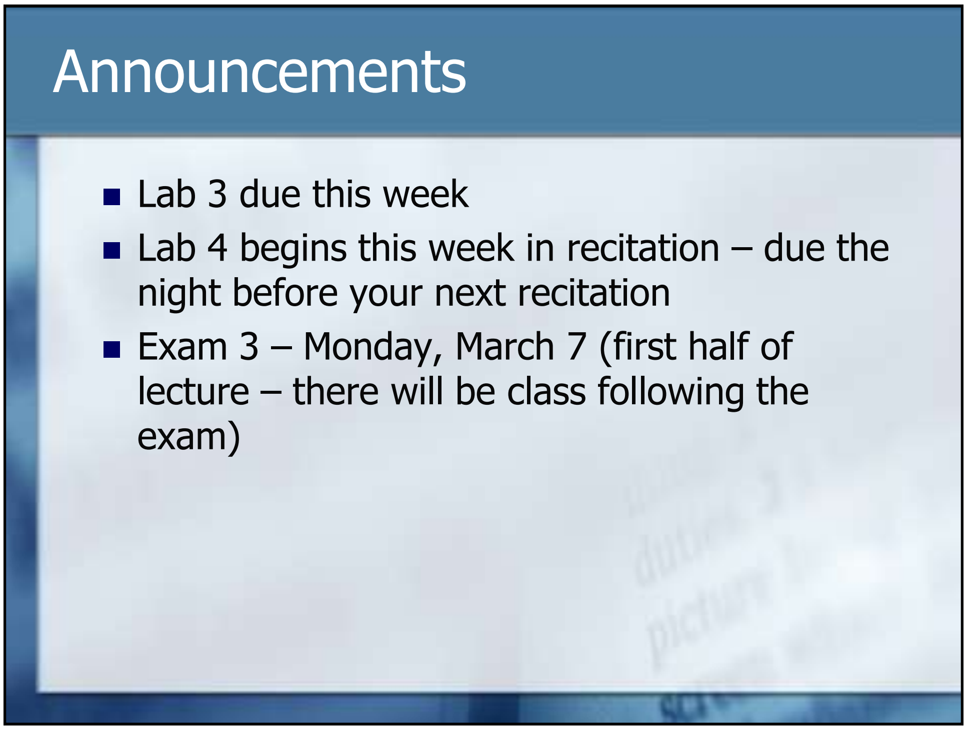


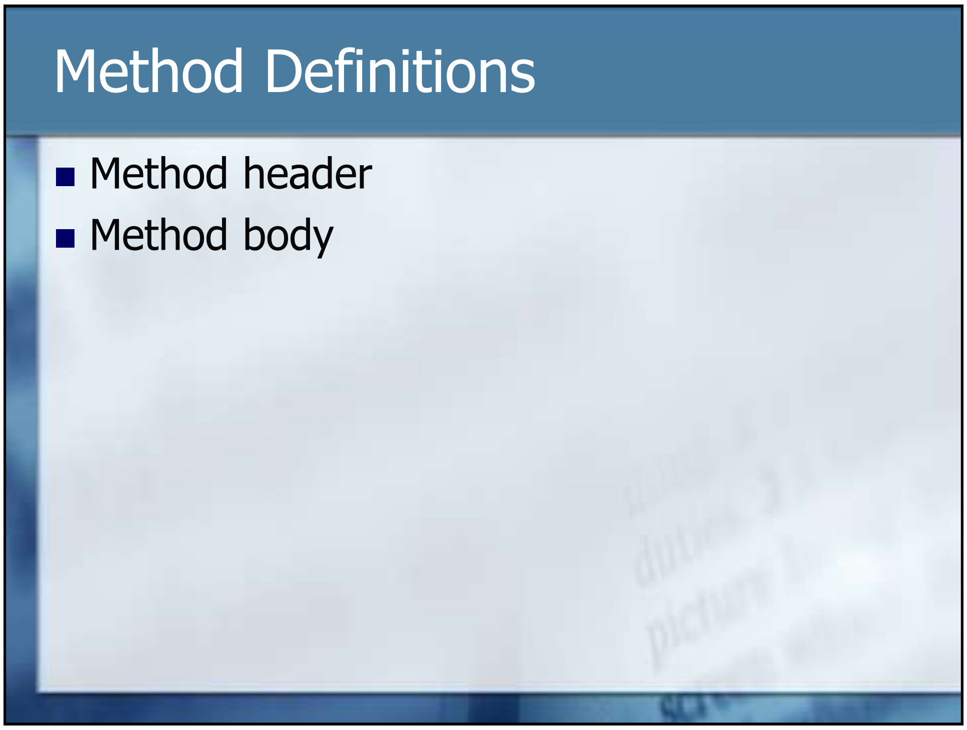
CSE 115/503

February 21 - 25, 2011



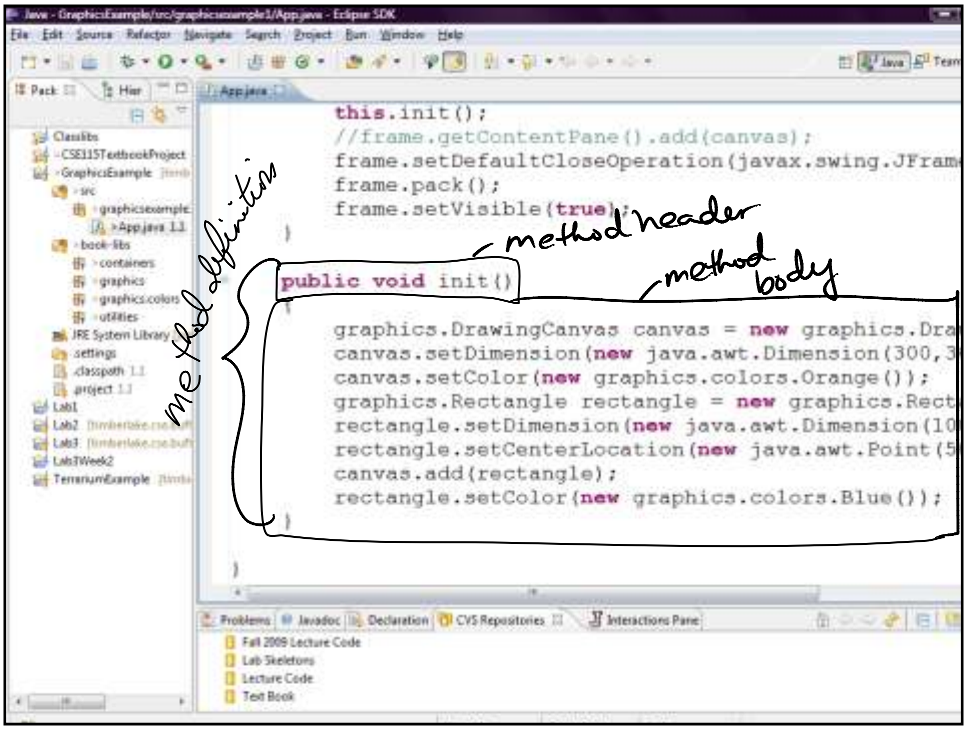
Announcements

- Lab 3 due this week
- Lab 4 begins this week in recitation – due the night before your next recitation
- Exam 3 – Monday, March 7 (first half of lecture – there will be class following the exam)

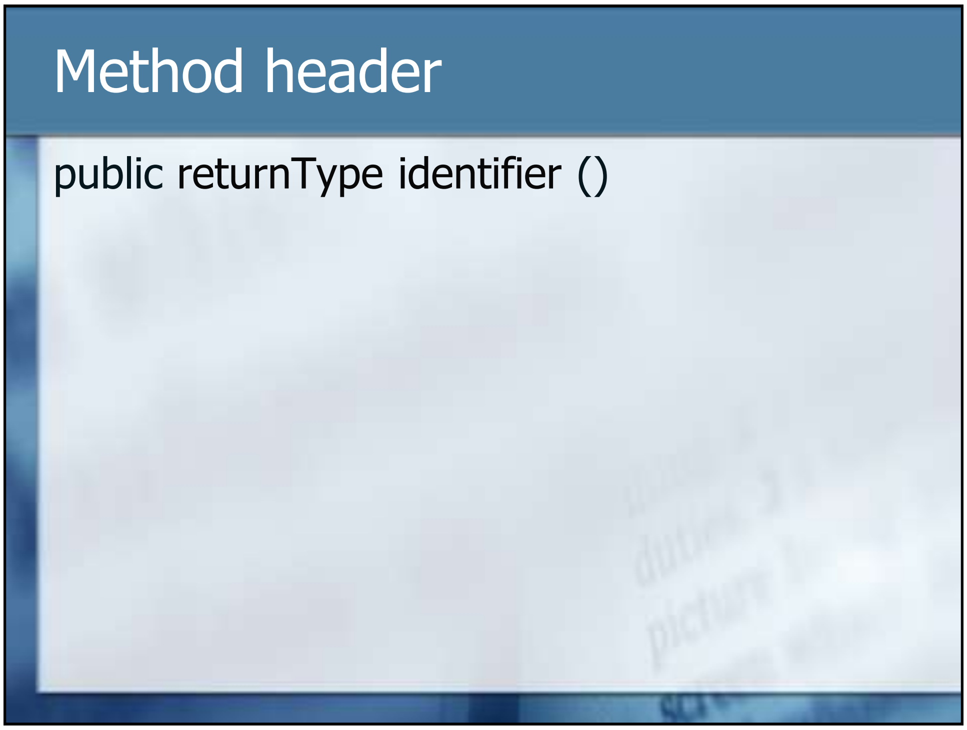


Method Definitions

- Method header
- Method body

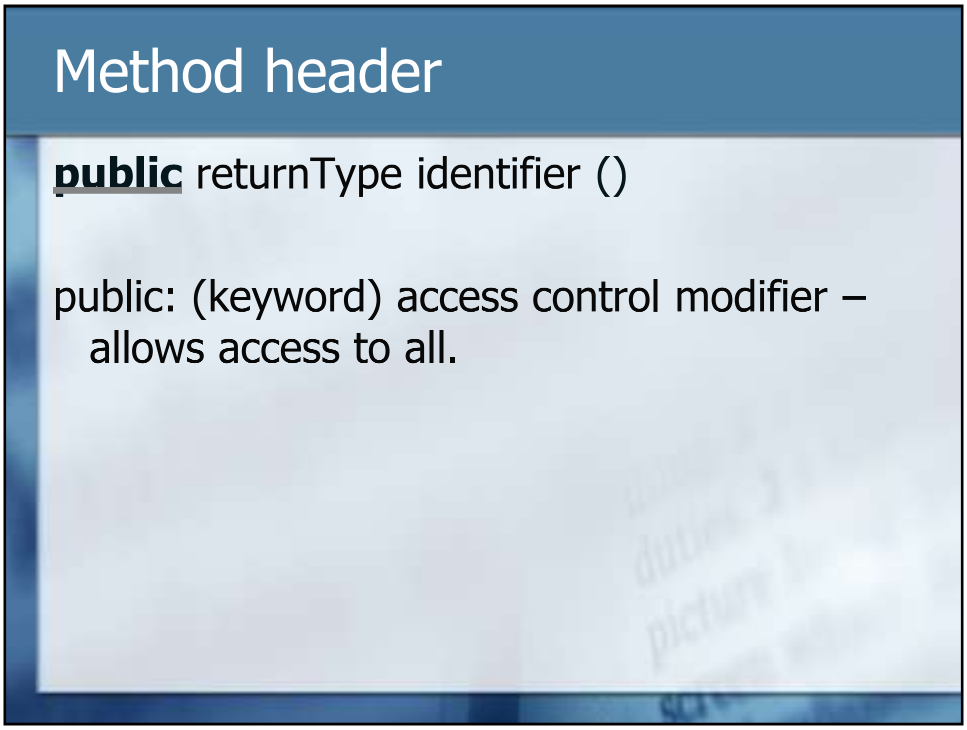


```
        this.init();
        //frame.getContentPane().add(canvas);
        frame.setDefaultCloseOperation(javax.swing.JFram
        frame.pack();
        frame.setVisible(true);
    }
    public void init()
    {
        graphics.DrawingCanvas canvas = new graphics.Dra
        canvas.setDimension(new java.awt.Dimension(300,3
        canvas.setColor(new graphics.colors.Orange());
        graphics.Rectangle rectangle = new graphics.Rect
        rectangle.setDimension(new java.awt.Dimension(10
        rectangle.setCenterLocation(new java.awt.Point(5
        canvas.add(rectangle);
        rectangle.setColor(new graphics.colors.Blue());
    }
}
```



Method header

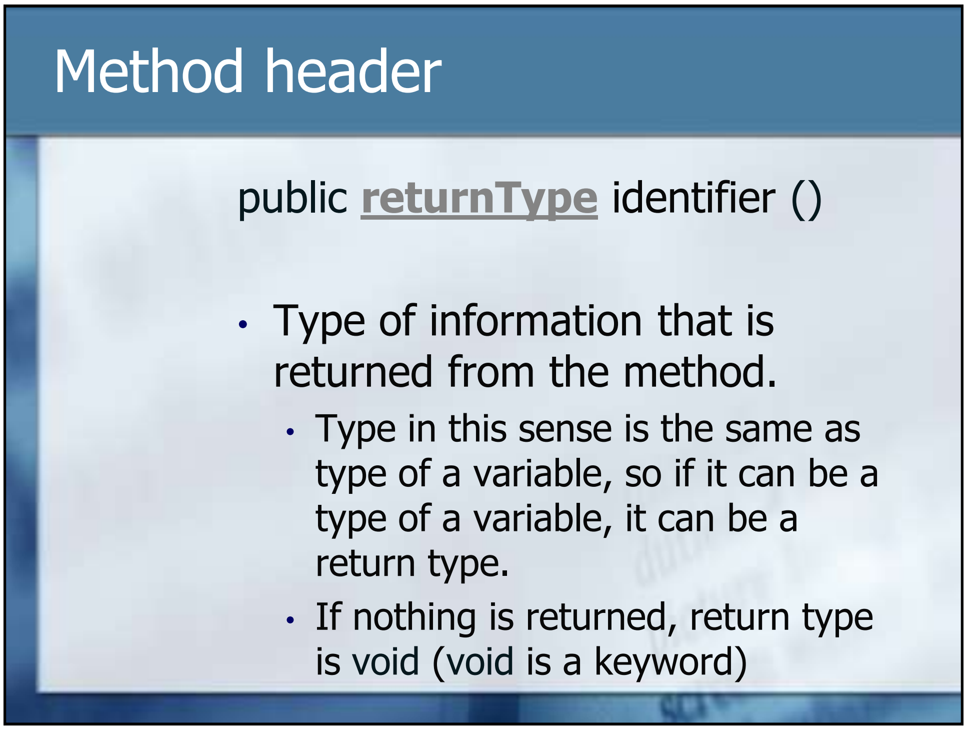
public returnType identifier ()



Method header

public returnType identifier ()

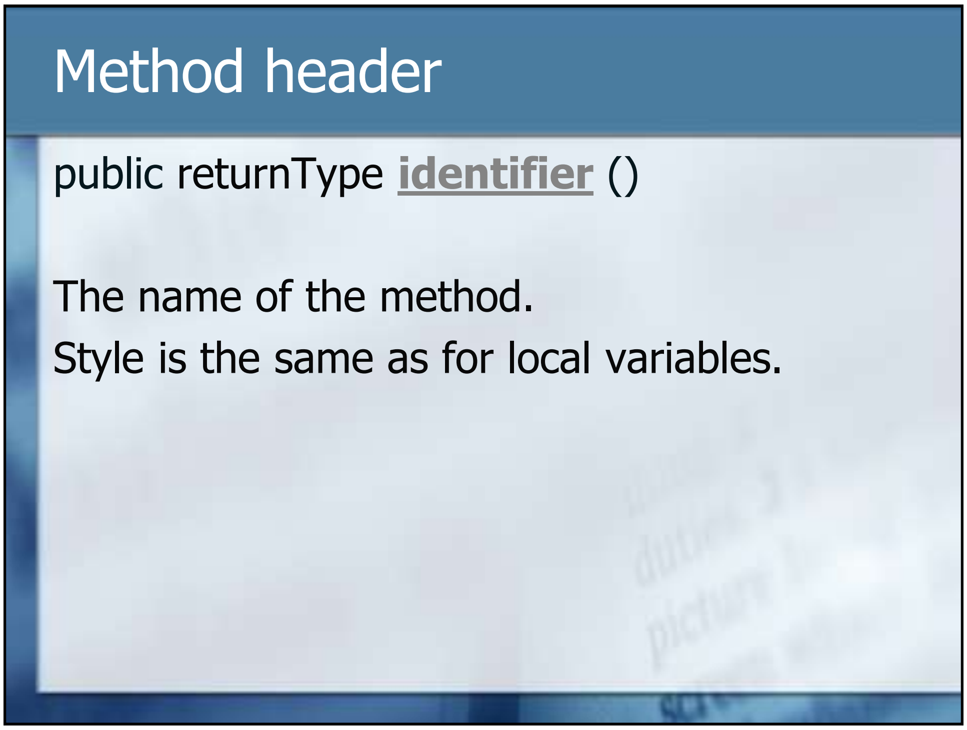
public: (keyword) access control modifier – allows access to all.



Method header

public **returnType** identifier ()

- Type of information that is returned from the method.
  - Type in this sense is the same as type of a variable, so if it can be a type of a variable, it can be a return type.
  - If nothing is returned, return type is void (void is a keyword)



Method header

public returnType **identifier** ()

The name of the method.

Style is the same as for local variables.

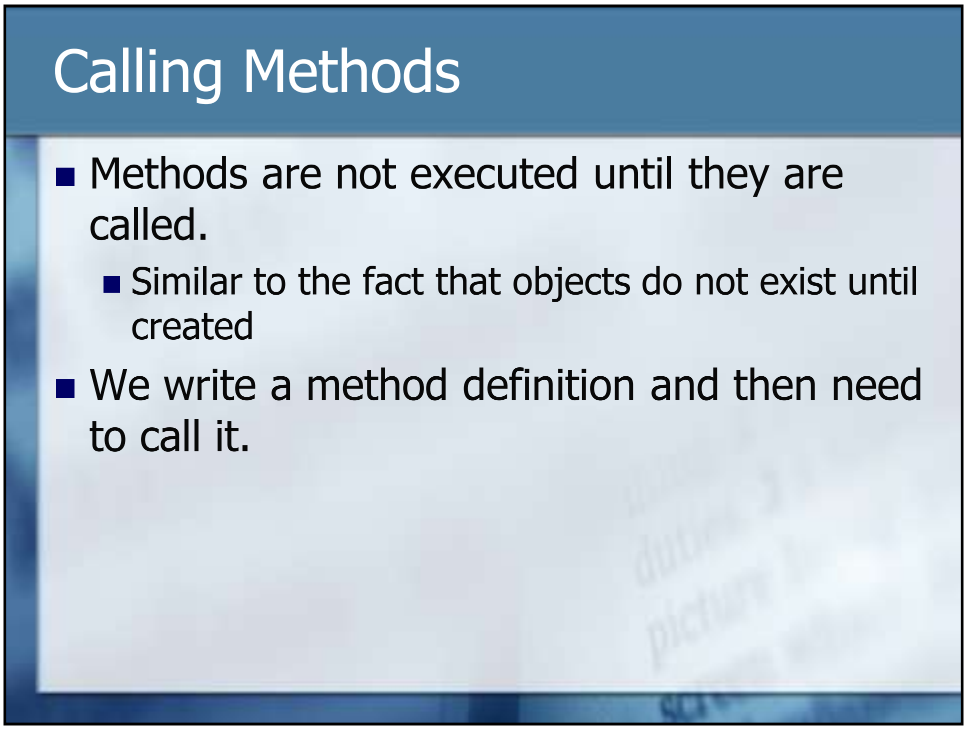
Method header

public returnType identifier ()

Parameter list: additional information that is needed so that the method can perform its task.

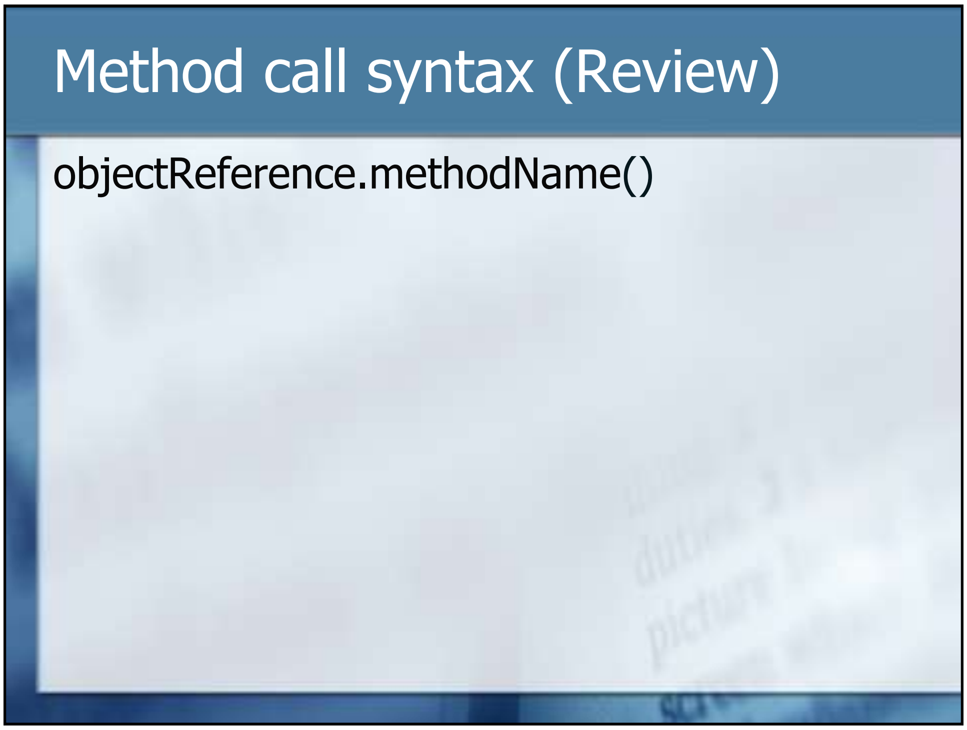
Parameter lists

- Can be empty
  - No additional information needed for method
- Can contain one parameter
  - Parameter declaration syntax:
    - type identifier
    - Looks like a local variable declaration; same style rules apply
- Can contain more than one parameter
  - Comma-separated list of "one parameters"
    - i.e. each parameter needs a type and an identifier



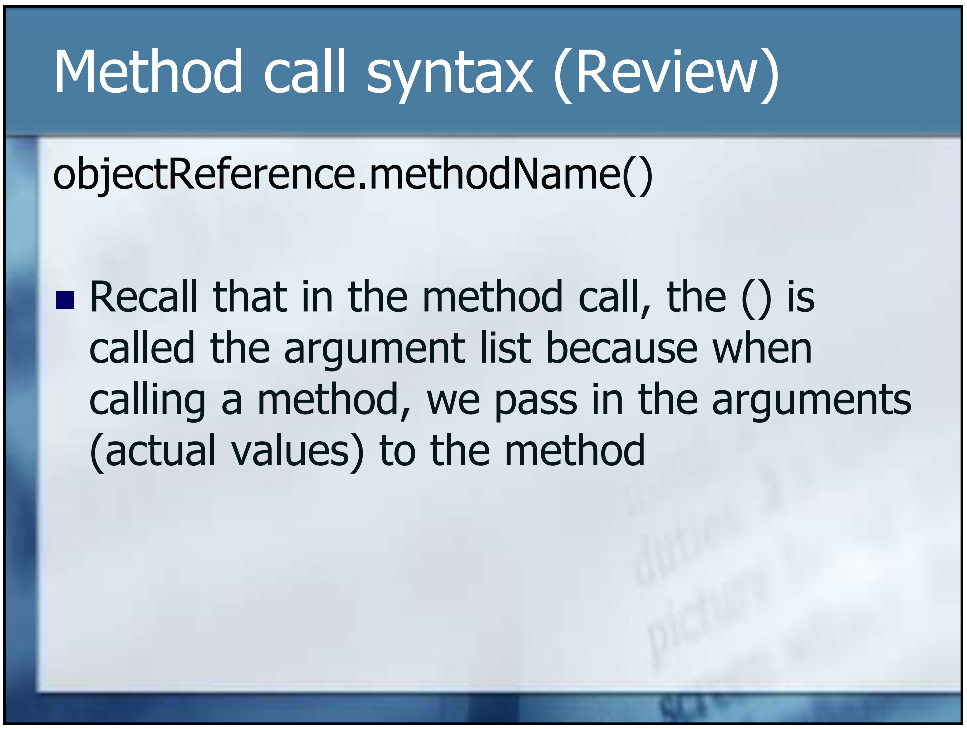
Calling Methods

- Methods are not executed until they are called.
  - Similar to the fact that objects do not exist until created
- We write a method definition and then need to call it.



Method call syntax (Review)

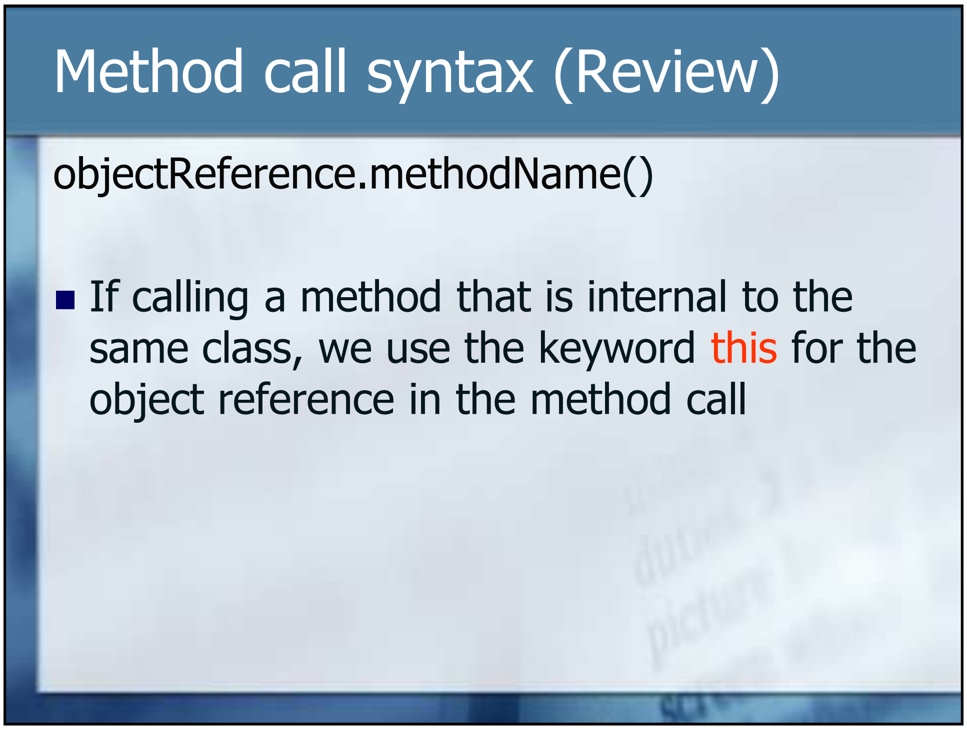
objectReference.methodName()



Method call syntax (Review)

objectReference.methodName()

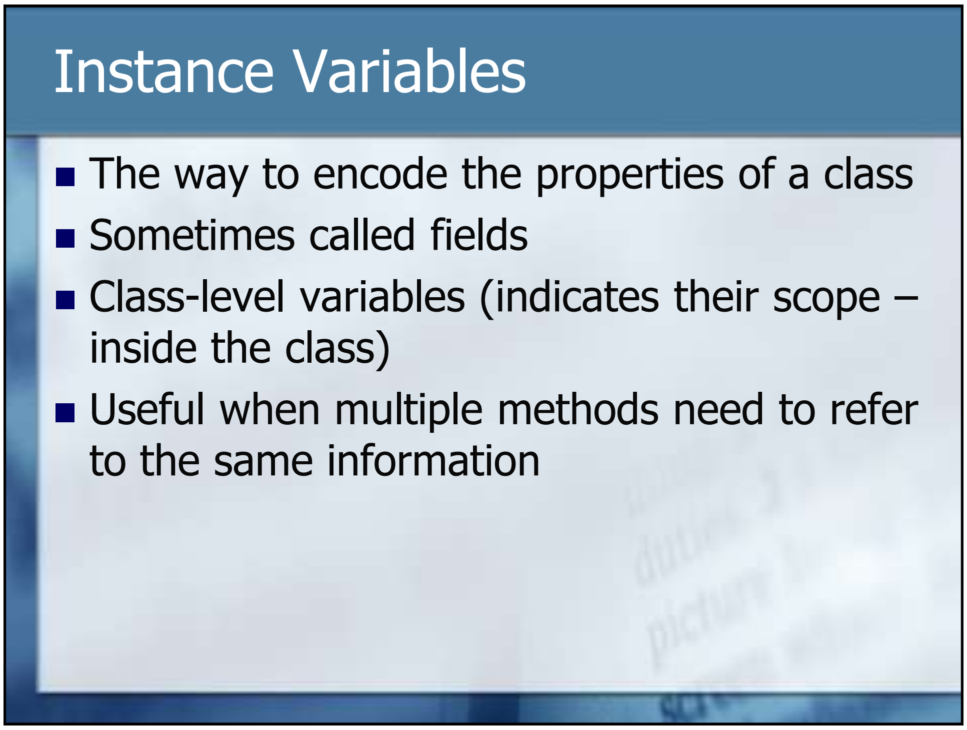
- Recall that in the method call, the () is called the argument list because when calling a method, we pass in the arguments (actual values) to the method



Method call syntax (Review)

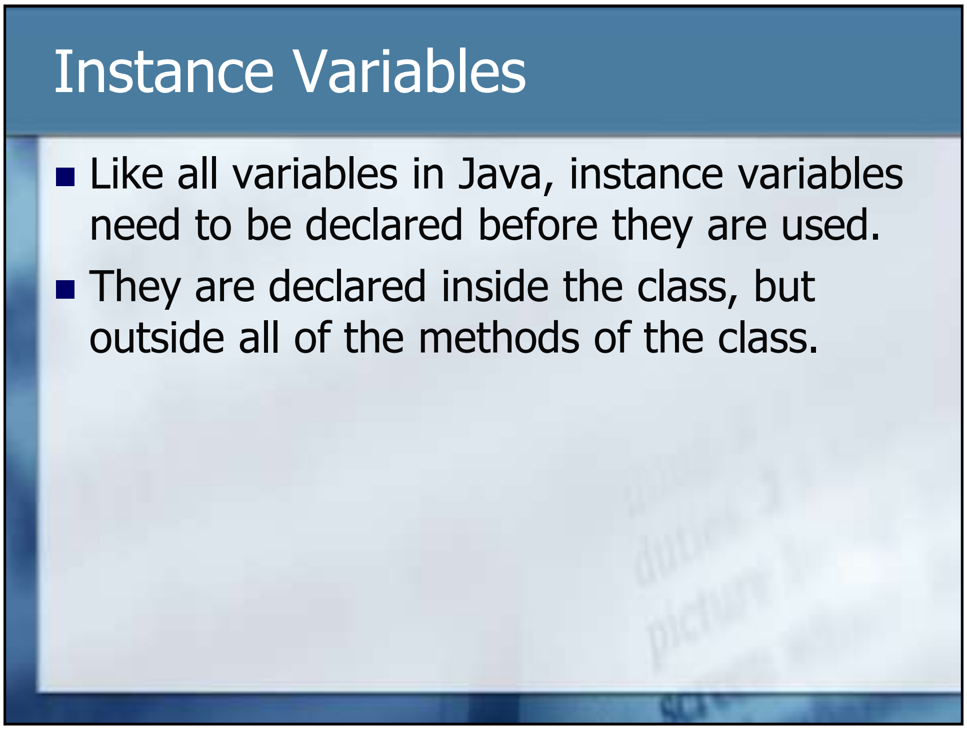
objectReference.methodName()

- If calling a method that is internal to the same class, we use the keyword this for the object reference in the method call



Instance Variables

- The way to encode the properties of a class
- Sometimes called fields
- Class-level variables (indicates their scope – inside the class)
- Useful when multiple methods need to refer to the same information



Instance Variables

- Like all variables in Java, instance variables need to be declared before they are used.
- They are declared inside the class, but outside all of the methods of the class.

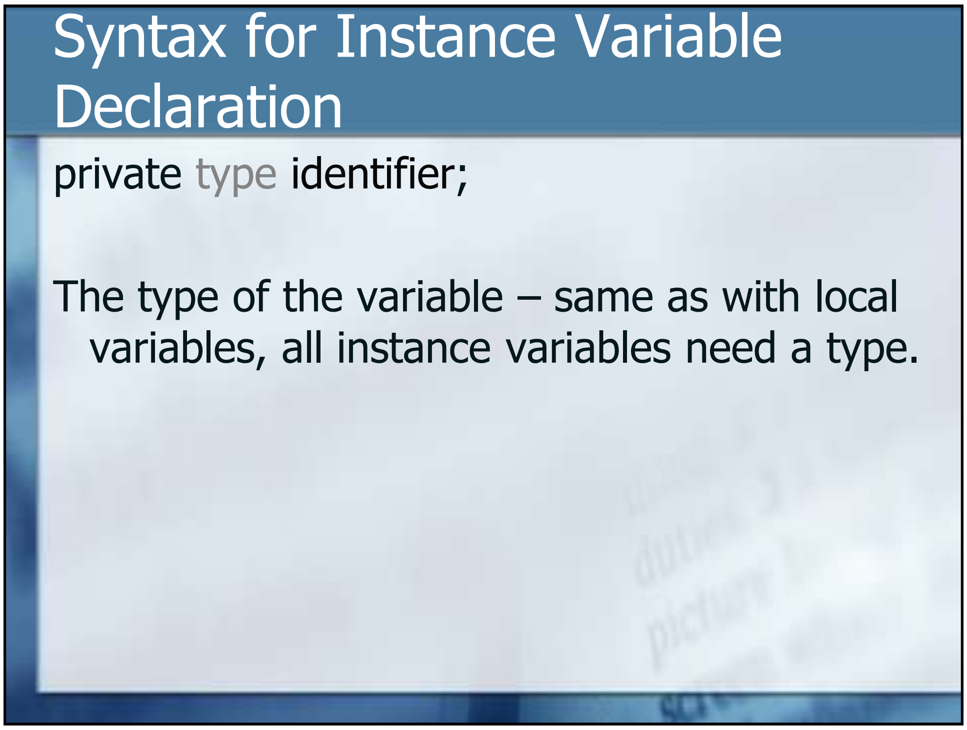
Syntax for Instance Variable Declaration

private type identifier;

Syntax for Instance Variable Declaration

private type identifier;

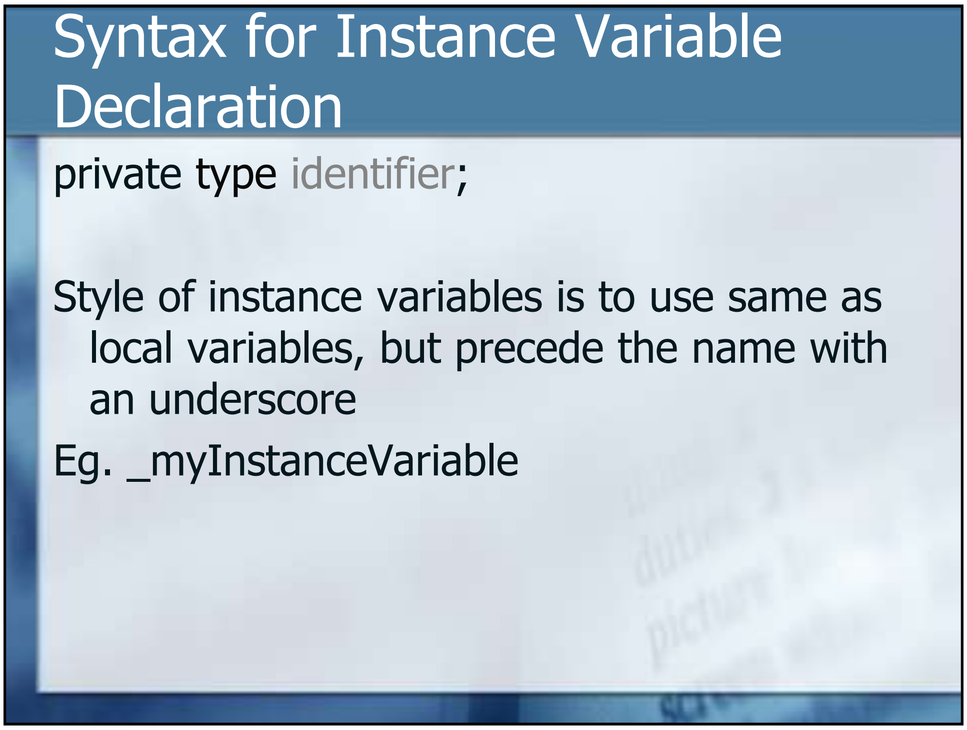
private (keyword) access control modifier indicating access only available inside the current class.



Syntax for Instance Variable Declaration

private type identifier;

The type of the variable – same as with local variables, all instance variables need a type.

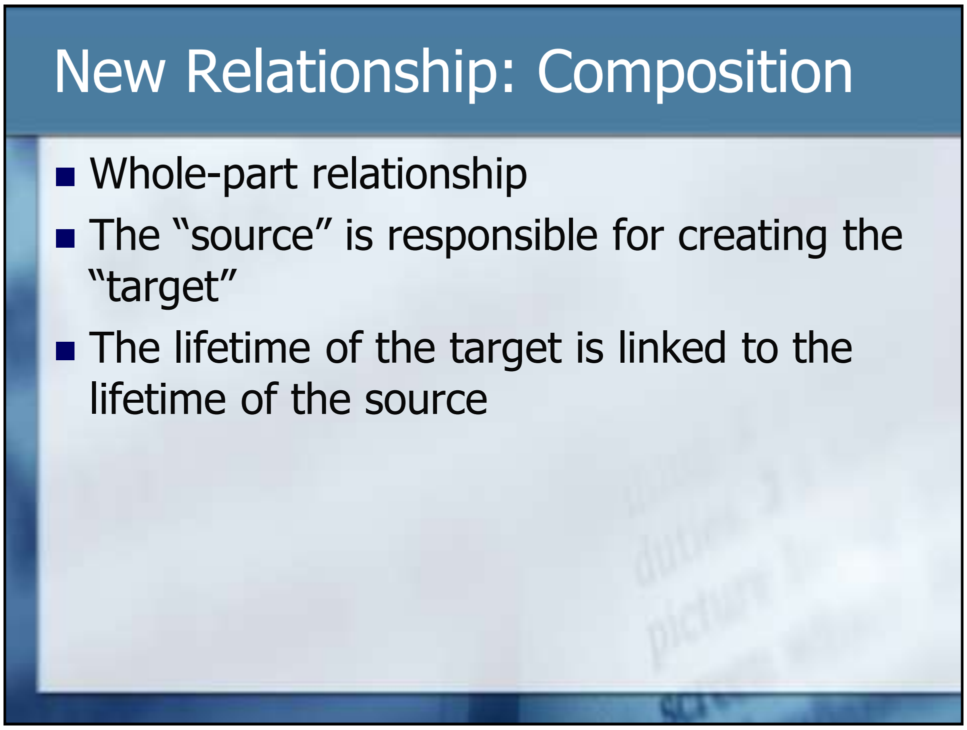


Syntax for Instance Variable Declaration

private type identifier;

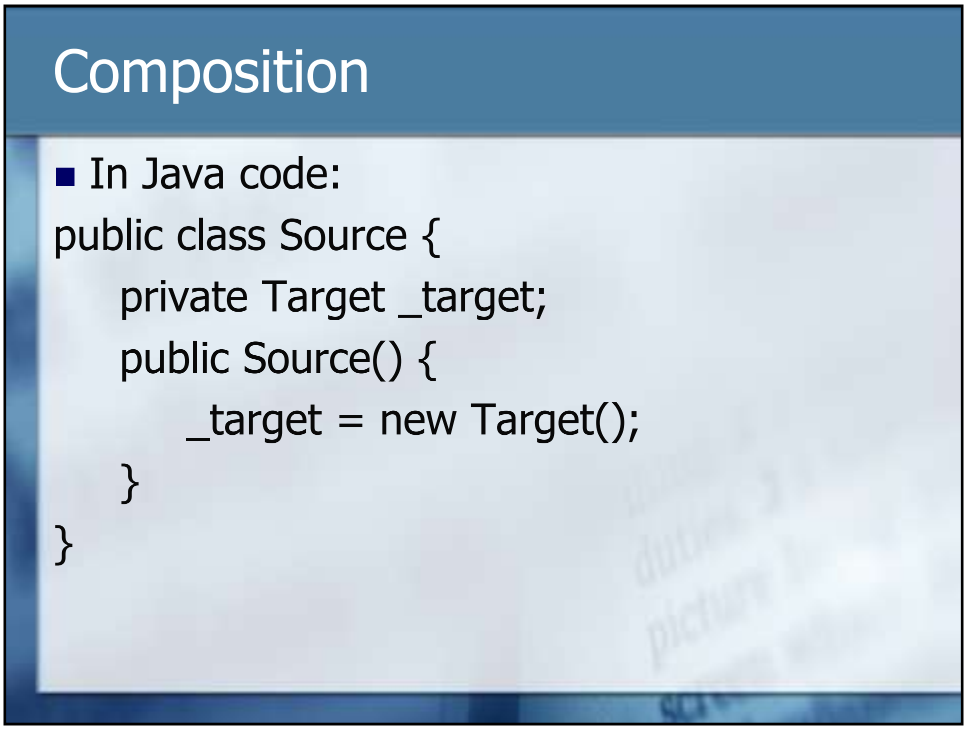
Style of instance variables is to use same as local variables, but precede the name with an underscore

Eg. _myInstanceVariable



New Relationship: Composition

- Whole-part relationship
- The “source” is responsible for creating the “target”
- The lifetime of the target is linked to the lifetime of the source



Composition

- In Java code:

```
public class Source {
    private Target _target;
    public Source() {
        _target = new Target();
    }
}
```