# Class Relationships

**November 3, 2006**

*ComS 207: Programming I (in Java)*
*Iowa State University, FALL 2006*
*Instructor: Alexander Stoytchev*



## Example: Sudoku_Solver.java



## Solving Sudoku Puzzles With Recursion (http://www.websudoku.com/)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 7 | 6 | 4 | 5 |
| 7 | 2 | | 4 | 9 | | | | 1 |
| | | 4 | | | | 7 | 9 | |
| | | | 5 | | | | | |
| 4 | | 9 | | 6 | | 2 | | 3 |
| | | | | | 2 | | | |
| | 3 | 8 | | | | 1 | | |
| 5 | | | | 2 | 8 | | 6 | 9 |
| 2 | 6 | 7 | 9 | | | | | |



## Rule #1: 1..9 must be in each row

| | | | | | 7 | 6 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|
| 7 | 2 | | 4 | 9 | | | | 1 |
| | | 4 | | | | 7 | 9 | |
| | | | 5 | | | | | |
| 4 | | 9 | | 6 | | 2 | | 3 |
| | | | | | 2 | | | |
| | 3 | 8 | | | | 1 | | |
| 5 | | | | 2 | 8 | | 6 | 9 |
| 2 | 6 | 7 | 9 | | | | | |



## Sample that satisfies rule #1

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 3 | 2 | 8 | 7 | 6 | 4 | 5 |
| 7 | 2 | | 4 | 9 | | | | 1 |
| | | 4 | | | | 7 | 9 | |
| | | | 5 | | | | | |
| 4 | | 9 | | 6 | | 2 | | 3 |
| | | | | | 2 | | | |
| | 3 | 8 | | | | 1 | | |
| 5 | | | | 2 | 8 | | 6 | 9 |
| 2 | 6 | 7 | 9 | | | | | |



## Rule #2: 1..9 must be in each column

| | | | | | 7 | 6 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|
| 7 | 2 | | 4 | 9 | | | | 1 |
| | | 4 | | | | 7 | 9 | |
| | | | 5 | | | | | |
| 4 | | 9 | | 6 | | 2 | | 3 |
| | | | | | 2 | | | |
| | 3 | 8 | | | | 1 | | |
| 5 | | | | 2 | 8 | | 6 | 9 |
| 2 | 6 | 7 | 9 | | | | | |

## Sample that satisfies rules #1 and #2

| 1 | 9 | 3 | 2 | 8 | 7 | 6 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|
| 7 | 2 |  | 4 | 9 |  |  |  | 1 |
| 8 |  | 4 |  |  |  | 7 | 9 |  |
| 3 |  |  | 5 |  |  |  |  |  |
| 4 |  | 9 |  | 6 |  | 2 |  | 3 |
| 6 |  |  |  |  | 2 |  |  |  |
| 9 | 3 | 8 |  |  |  | 1 |  |  |
| 5 |  |  |  | 2 | 8 |  | 6 | 9 |
| 2 | 6 | 7 | 9 |  |  |  |  |  |



## Rule #3: 1..9 must be in each 3x3 window

| 1 | 9 | 3 | 2 | 8 | 7 | 6 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|
| 7 | 2 |  | 4 | 9 |  |  |  | 1 |
| 8 |  | 4 |  |  |  | 7 | 9 |  |
| 3 |  |  | 5 |  |  |  |  |  |
| 4 |  | 9 |  | 6 |  | 2 |  | 3 |
| 6 |  |  |  |  | 2 |  |  |  |
| 9 | 3 | 8 |  |  |  | 1 |  |  |
| 5 |  |  |  | 2 | 8 |  | 6 | 9 |
| 2 | 6 | 7 | 9 |  |  |  |  |  |



## Sample that satisfies rules #1, #2, and #3

| 1 | 9 | 3 | 2 | 8 | 7 | 6 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 6 | 4 | 9 |  |  |  | 1 |
| 8 | 5 | 4 |  |  |  | 7 | 9 |  |
| 3 |  |  | 5 |  |  |  |  |  |
| 4 |  | 9 |  | 6 |  | 2 |  | 3 |
| 6 |  |  |  |  | 2 |  |  |  |
| 9 | 3 | 8 |  |  |  | 1 |  |  |
| 5 |  |  |  | 2 | 8 |  | 6 | 9 |
| 2 | 6 | 7 | 9 |  |  |  |  |  |



## Quick Review of Last Lecture



## Visibility Modifiers

| | `public` | `private` |
|---|---|---|
| Variables | Violate encapsulation | Enforce encapsulation |
| Methods | Provide services to clients | Support other methods in the class |



## The static Modifier

- We declare static methods and variables using the `static` modifier
- It associates the method or variable with the class rather than with an object of that class
- Static methods are sometimes called *class methods* and static variables are sometimes called *class variables*

## Static Variables

- Normally, each object has its own data space, but if a variable is declared as static, only one copy of the variable exists

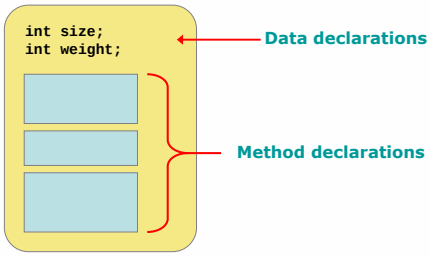
```
private static float price;
```

- Memory space for a static variable is created when the class is first referenced
- All objects instantiated from the class share its static variables
- Changing the value of a static variable in one object changes it for all others



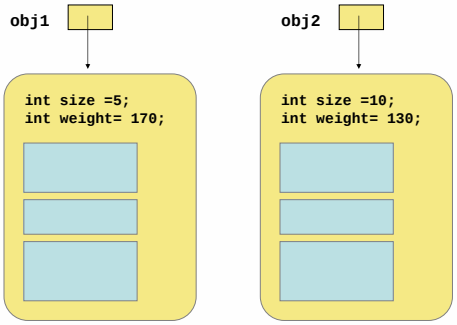
## Classes

- A class can contain data declarations and method declarations



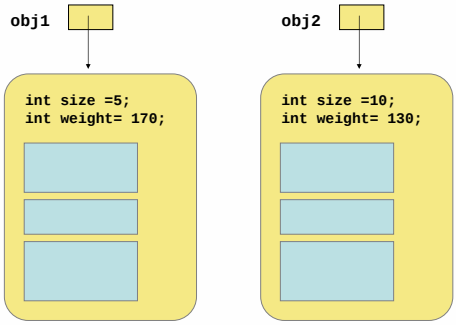



## Objects – instances of classes





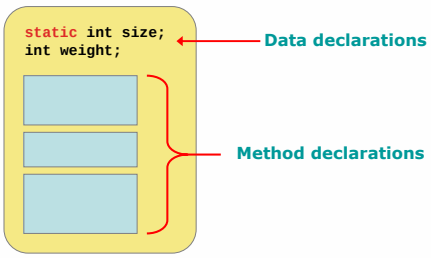

## Note that the variables can have different values in the two objects





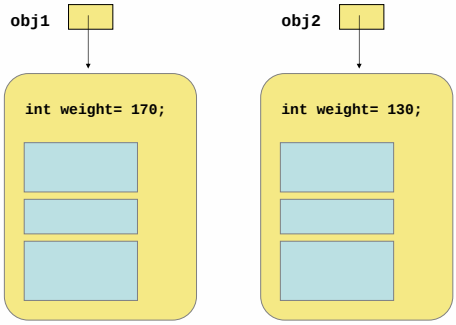

## Classes

- Things change if we declare a static variable





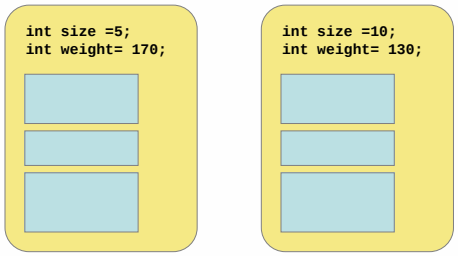

## Objects – instances of a class with a static variable 'size'




static int size;

## Objects – instances of classes

- Note that the variables can have different values in the two objects





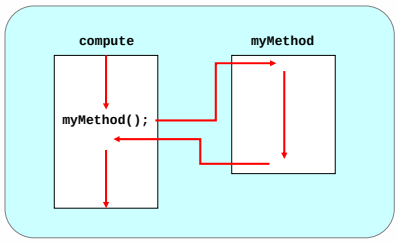

## Static Class Members

- The order of the modifiers can be interchanged, but by convention visibility modifiers come first
- Recall that the `main` method is static – it is invoked by the Java interpreter without creating an object
- Static methods cannot reference instance variables because instance variables don't exist until an object exists
- However, a static method can reference static variables or local variables



## Method Control Flow

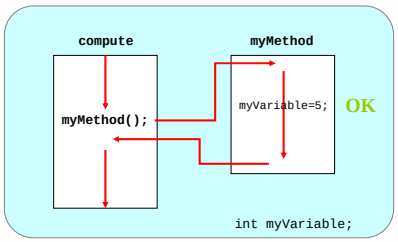
- If the called method is in the same class, only the method name is needed






## Accessing Variables

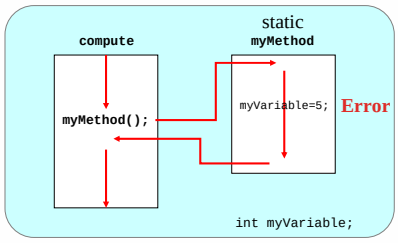
- If the called method is in the same class, only the method name is needed






## Accessing Variables

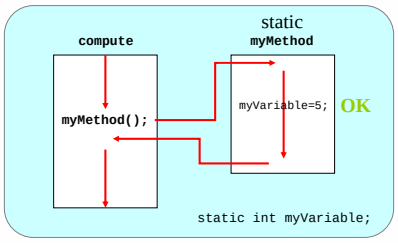
- Static methods cannot use non static class variables.






## Accessing Variables

- Static methods can use static class varables



## Static Class Members

- Recall that a static method is one that can be invoked through its class name
- For example, the methods of the `Math` class are static:

```
result = Math.sqrt(25);
```

- Variables can be static as well
- Determining if a method or variable should be static is an important design decision



## Static Methods

```
class Helper
{
   public static int cube (int num)
   {
      return num * num * num;
   }
}
```

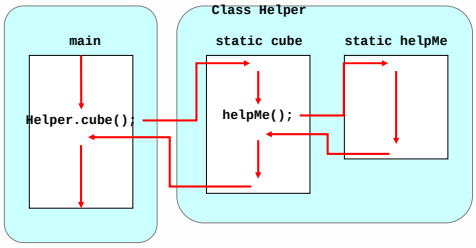
Because it is declared as static, the method can be invoked as

```
value = Helper.cube(5);
```



## Method Control Flow

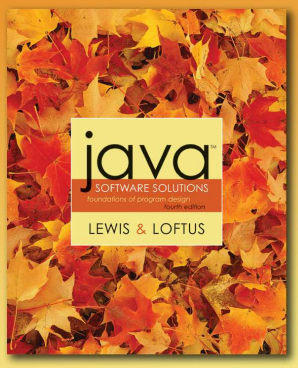
- Static methods can only call other static methods within the same classs






Chapter 6

Section 6.4






## Class Relationships

- Classes in a software system can have various types of relationships to each other
- Three of the most common relationships:
  - Dependency: A *uses* B
  - Aggregation: A *has-a* B
  - Inheritance: A *is-a* B



## Dependency

- A *dependency* exists when one class relies on another in some way, usually by invoking the methods of the other
- We've seen dependencies in many previous examples
- We don't want numerous or complex dependencies among classes
- Nor do we want complex classes that don't depend on others
- A good design strikes the right balance

## Dependency Example: Client-Server



## Dependency

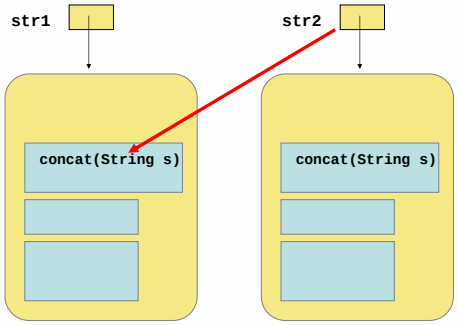
- Some dependencies occur between objects of the same class
- A method of the class may accept an object of the same class as a parameter
- For example, the `concat` method of the `String` class takes as a parameter another `String` object

```
str3 = str1.concat(str2);
```

- This drives home the idea that the service is being requested from a particular object



## Concatenation Example






## Dependency

- The following example defines a class called `Rational` to represent a rational number
- A rational number is a value that can be represented as the ratio of two integers
- Some methods of the `Rational` class accept another `Rational` object as a parameter
- See RationalTester.java (page 297)
- See Rational.java (page 299)



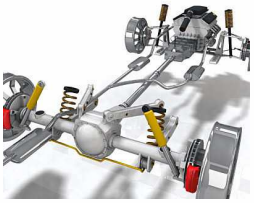
## Aggregation

- An *aggregate* is an object that is made up of other objects
- Therefore aggregation is a *has-a* relationship
  - A car *has a* chassis
  - A student has an address



## Aggregation Example: Components of a Car

## Chasis





## Tyres



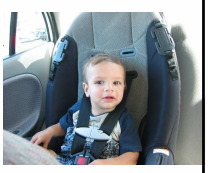
## Steering Wheel



## What type of Steering Wheel?



## Car Seat





## Aggregation

- **In software, an aggregate object contains references to other objects as instance data**
- **The aggregate object is defined in part by the objects that make it up**
- **This is a special kind of dependency – the aggregate usually relies on the objects that compose it**

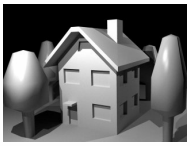
## Aggregation Example: Copmonents of a Student



## Student

First Name

Last Name



Home Address

School Address



## john

John

Smith

21 Jump Street

800 Lancaster Ave.



## marsha

Marsha

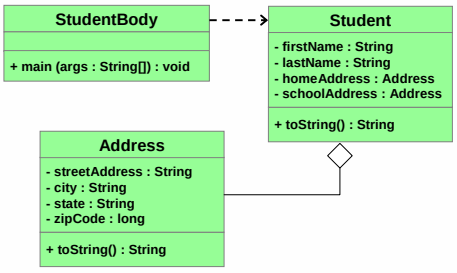
Jones

123 Main Street

800 Lancaster Ave.



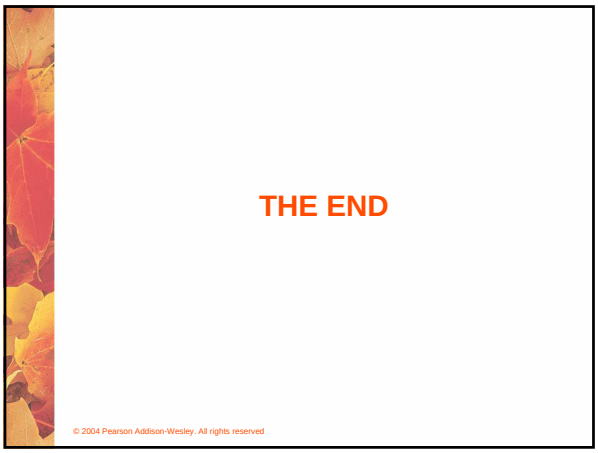
## Aggregation in UML






## Aggregation

- In the following example, a `Student` object is composed, in part, of `Address` objects
- A student has an address (in fact each student has two addresses)
- See StudentBody.java (page 304)
- See Student.java (page 306)
- See Address.java (page 307)
- An aggregation association is shown in a UML class diagram using an open diamond at the aggregate end


THE END
© 2004 Pearson Addison-Wesley. All rights reserved