Static Class Members

November 1, 2006

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

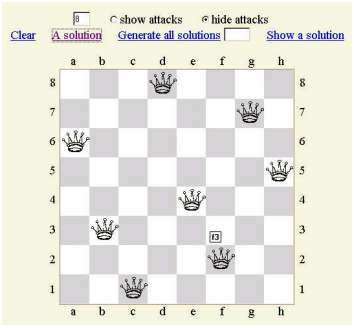
Go Over Midterm2 Solutions

HW 8 is out

- **Also Check out this web page:**

  http://www.cs.princeton.edu/introcs/23recursion/

The Eight Queens Problem



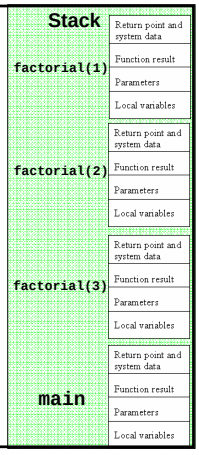
[http://home.earthlink.net/~barnold2002/Acgnj/Csig9908.htm]

Recursion Example

Code

```
public static int factorial(int n)
{
    if(n==1)
       return 1;
    else
       return n* factorial(n-1);
}


public static void main(String[] args)
{
      int result = factorial(3);
      System.out.println(result);
}
```

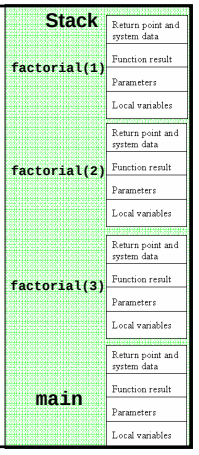


Recursion Example

Code

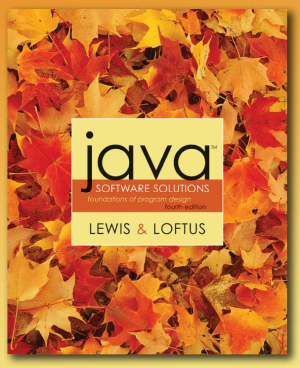
```
public static int factorial(int n)
{
    if(n==1)
       return 1;
    else
       return n* factorial(n-1);
}


public static void main(String[] args)
{
      int result = factorial(3);
      System.out.println(result);
}
```



Chapter 6

Object-Oriented Design



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);
```


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

Visibility Modifiers

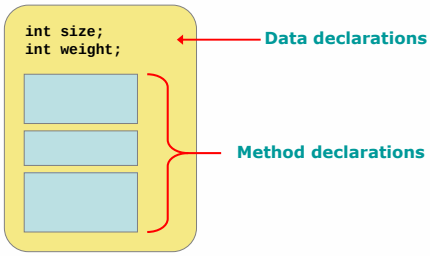
	`public`	`private`
Variables	Violate encapsulation	Enforce encapsulation
Methods	Provide services to clients	Support other methods in the class

Static Class Members

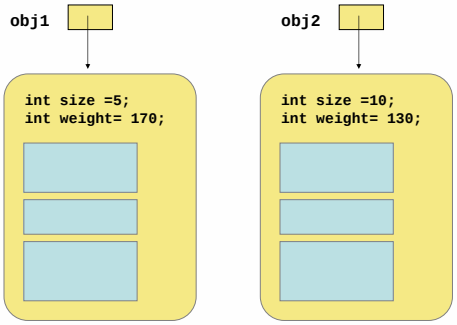
- The order of the modifiers can be interchanged, but by convention visibility modifiers come first
- Recall that the `main` method is static – it is invoked by the Java interpreter without creating an object
- Static methods cannot reference instance variables because instance variables don't exist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

Classes

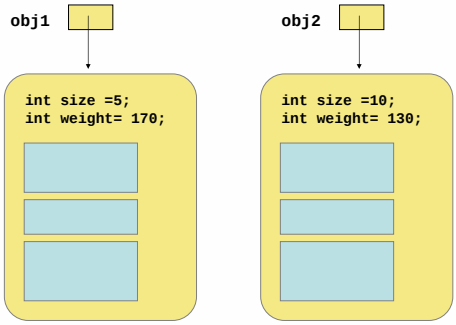
- A class can contain data declarations and method declarations



Objects – instances of classes

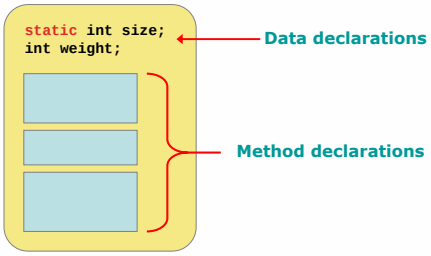


Note that the variables can have different values in the two objects

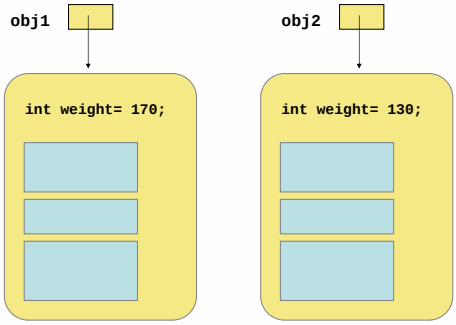


Classes

- Things change if we declare a static variable



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



static int size;

Static allocation

- ***Static allocation*** **means allocation of storage before the program starts and retention until the end.**
- **The locations of objects are basically decided at compile-time, although they might be relocated at load-time. This implies the sizes of the objects must be known then.**

 [http://www.memorymanagement.org/glossary/s.html]

Limitations

Using only static allocation is restrictive, as sizes of data structures can't be dynamically varied, and procedures cannot be recursive. However, it is also fast and eliminates the possibility of running out of memory. For this reason, this scheme is sometimes used in real-time systems.

 [http://www.memorymanagement.org/glossary/s.html]

Historical note

The first high-level language, Fortran, only had static allocation to begin with. Later languages usually offer heap and/or stack allocation, but static allocation is often available as an option.

 [http://www.memorymanagement.org/glossary/s.html]

Static Class Members

- **Static methods and static variables often work together**
- **The following example keeps track of how many `Slogan` objects have been created using a static variable, and makes that information available using a static method**
- **See SloganCounter.java (page 294)**
- **See Slogan.java (page 295)**

Sections 6.1 & 6.2
(dry stuff that you can read on your own)

Program Development

- **The creation of software involves four basic activities:**
  - **establishing the requirements**
  - **creating a design**
  - **implementing the code**
  - **testing the implementation**
- **These activities are not strictly linear – they overlap and interact**

Requirements

- *Software requirements* specify the tasks that a program must accomplish
  - what to do, not how to do it
- Often an initial set of requirements is provided, but they should be critiqued and expanded
- It is difficult to establish detailed, unambiguous, and complete requirements
- Careful attention to the requirements can save significant time and expense in the overall project

Design

- A *software design* specifies how a program will accomplish its requirements
- That is, a software design determines:
  - how the solution can be broken down into manageable pieces
  - what each piece will do
- An object-oriented design determines which classes and objects are needed, and specifies how they will interact
- Low level design details include how individual methods will accomplish their tasks

Implementation

- *Implementation* is the process of translating a design into source code
- Novice programmers often think that writing code is the heart of software development, but actually it should be the least creative step
- Almost all important decisions are made during requirements and design stages
- Implementation should focus on coding details, including style guidelines and documentation

Testing

- *Testing* attempts to ensure that the program will solve the intended problem under all the constraints specified in the requirements
- A program should be thoroughly tested with the goal of finding errors
- *Debugging* is the process of determining the cause of a problem and fixing it
- We revisit the details of the testing process later in this chapter

Identifying Classes and Objects

- The core activity of object-oriented design is determining the classes and objects that will make up the solution
- The classes may be part of a class library, reused from a previous project, or newly written
- One way to identify potential classes is to identify the objects discussed in the requirements
- Objects are generally nouns, and the services that an object provides are generally verbs

Identifying Classes and Objects

- A partial requirements document:

> The user must be allowed to specify each product by its primary characteristics, including its name and product number. If the bar code does not match the product, then an error should be generated to the message window and entered into the error log. The summary report of all transactions must be structured as specified in section 7.A.

Of course, not all nouns will correspond to a class or object in the final solution

Identifying Classes and Objects

- Remember that a class represents a group (classification) of objects with the same behaviors
- Generally, classes that represent objects should be given names that are singular nouns
- Examples: `Coin`, `Student`, `Message`
- A class represents the concept of one such object
- We are free to instantiate as many of each object as needed

Identifying Classes and Objects

- Sometimes it is challenging to decide whether something should be represented as a class
- For example, should an employee's address be represented as a set of instance variables or as an `Address` object
- The more you examine the problem and its details the more clear these issues become
- When a class becomes too complex, it often should be decomposed into multiple smaller classes to distribute the responsibilities

Identifying Classes and Objects

- We want to define classes with the proper amount of detail
- For example, it may be unnecessary to create separate classes for each type of appliance in a house
- It may be sufficient to define a more general `Appliance` class with appropriate instance data
- It all depends on the details of the problem being solved

Identifying Classes and Objects

- Part of identifying the classes we need is the process of *assigning responsibilities* to each class
- Every activity that a program must accomplish must be represented by one or more methods in one or more classes
- We generally use verbs for the names of methods
- In early stages it is not necessary to determine every method of every class – begin with primary responsibilities and evolve the design

THE END