

Lecture 13: Libraries and Encapsulaton

Last time:

1. `for` loops (from last lecture notes)
2. Static variables and methods

Today:

1. Parameter passing
2. Libraries
3. Public vs. private



Parameters and Methods

- Recall that methods / constructors can have parameters

```
public Student (String newName, int IDDesired) {  
    name = newName;  
    id = IDDesired;  
    tokenLevel = 3;  
}
```

- What is printed by the following?

```
String newName = "Joe";  
Student s = new Student(newName + " Schmoe", 123456789);  
System.out.println (s.name);  
System.out.println (newName);
```

- Joe Schmoe
Joe

How Does Java Evaluate Method / Constructor Calls?



```
int newName = "Joe";  
Student s = new Student  
    (newName + " Schmoie", 123456789);
```

1. Arguments are evaluated using stack in effect at **call site** (place where method called)
 - `newName + " Schmoie"`, evaluates to Joe Schmoie
 - `123456789` evaluates to 123456789
2. **Stack frame** (temporary addition to stack) created to associate method parameters with values
3. Stack frame put into stack
4. Body of method executed in modified stack
5. Stack frame removed from stack

Example

- ```

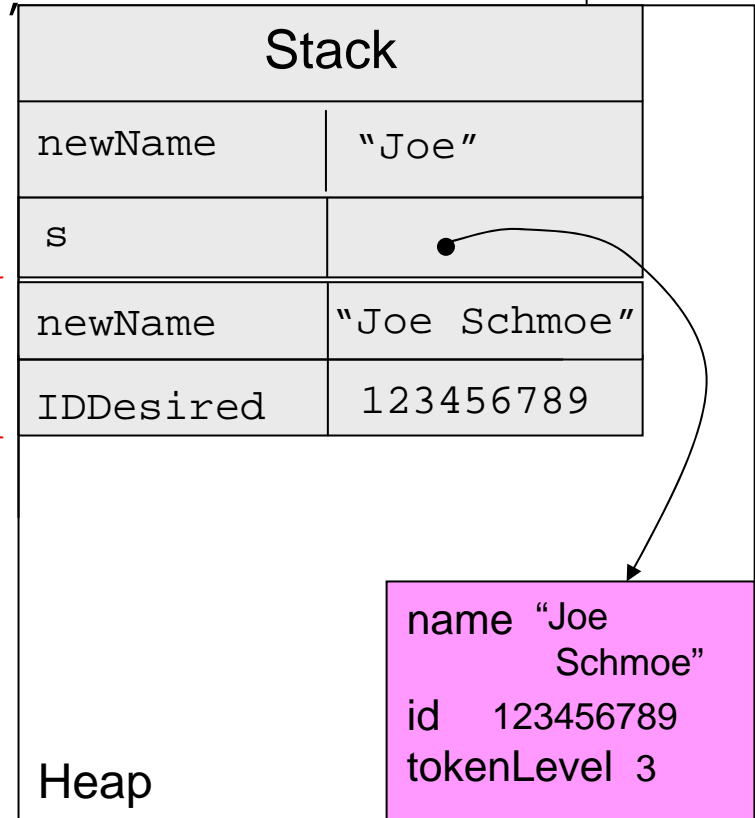
public Student (String newName,
int IDDesired) {
 name = newName;
 id = IDDesired;
 tokenLevel = 3;
}

```
- ```

String newName = "Joe";
Student s = new Student
(newName + " Schmoe",
123456789);
System.out.println (s.name);
System.out.println (newName);

```

stack
frame



Note: When two variables in the stack have the same name, Java uses the one in the **active** (most recently instantiated) frame. Only the variables that appear in the active frame are "in scope". (All other stack variables are "out of scope.")

Libraries in Java

- **Library**: implementation of useful routines that are shared by different programs
- Java mechanism for creating libraries: **packages**
 - Package: group of related classes
 - Example: `java.util` (contains `Scanner` class)
- To use a class from a package, you can use a **fully qualified name** (package name + class name):

```
java.util.Scanner s = new
    java.util.Scanner(System.in);
```
- You can also import the class in the beginning

```
import java.util.Scanner;
```
- To import class in a package:

```
import java.util.*;
```

(Imports `Scanner`, other classes in package)

Package `java.lang`

- A special package containing widely used classes:
 - `String`
 - `Math`
 - `etc.`
- `java.lang.*` is *automatically imported* by every Java program

Package Management

- A class can be added to a package by including:
`package <name of package>;`
in source file (usually very first line)
- The variables / methods provided by a class / package are often called its **API** (= Application Programmers Interface)
- APIs should be documented
- java.lang documentation:
<http://java.sun.com/j2se/1.3/docs/api/java/lang/package-summary.html>

String API Example

```
String s = new String("CAT");
String t = new String("cat");

int i = s.length();

boolean a = s.equals(t);           // false
boolean b = s.equalsIgnoreCase(t); // true

int i = s.compareTo(t);           // some negative number
int j = s.compareToIgnoreCase(t); // 0

char c0 = s.charAt(0);             // 'C'
char c1 = s.charAt(1);             // 'A'
char c2 = s.charAt(2);             // 'T'

String x = t.toLowerCase();        // Creates a NEW String
String y = s.toUpperCase();        // Ditto
```

Math API Example

- `Math` implements lots of mathematical functions
- Its members are *static*

```
double x = Math.abs(-33.7);
```

```
double y = Math.sqrt(7);
```

```
double pi = Math.PI;
```

```
double c = Math.cos(2.3);
```

```
double z = Math.pow(2.0, 3.0);
```

```
double r = Math.random(); // between 0 and 1
```

Public Declarations

- So far all classes / variables / methods have been **public**
 - Keyword `public` used in declaration
 - Every user of an object can access any `public` element
- Sometimes access should be restricted!
 - To avoid giving object users unnecessary info (keep API small)
 - To enforce consistency on instance variables

Example

```
// "acceptTokens" allows the tokenLevel to be set; an
// error is reported if the number of tokens is invalid
public void acceptTokens (int newTokens) {
    if ((newTokens > 0) || (newTokens <= 3)) {
        tokenLevel = newTokens;
    } else{
        System.out.println
            ("Bad argument to acceptTokens: " + newTokens);
    }
}
```

- Goal of `acceptTokens`: ensure `tokenLevel` values are valid
- But `tokenLevel` can still be made invalid

```
Student s = new Student ();
d.tokenLevel = 99;
```

Private Declarations

- Java also allows variables / methods to be declared **private**:

```
private int tokenLevel = 3;
```

- Private variables / members cannot be accessed outside the class definition
- Declaring instance variables private means they can only be modified using public methods

Example

```
public class Student {  
    private String name;  
    private int id;  
    private int tokenLevel = 3;  
    ...  
}
```

What Should Be Public / Private?



- **Class interface** = API = public variables / methods
- Only make something public if there is a reason to
- Why? **Encapsulation**
 - As long as interface is preserved, class can change without breaking other code
 - The more limited the interface, the less there is to maintain
- Rule of thumb
 - Make instance variables private
 - Implement `set` / `get` methods
 - Make auxiliary methods private