

Lecture Set #7: Libraries, Encapsulaton, Packages and “this”

1. Review of Parameter passing
2. Libraries
3. Public vs. private
4. Package Management
5. this



CMSC 131 Fall 2007
Jan Plane (adapted from Bonnie Dorri)

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



CMSC 131 Fall 2007
Jan Plane (adapted from Bonnie Dorri)

1

How Does Java Evaluate Method / Constructor Calls?



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

1. Arguments are evaluated using stack in effect at **call site** (place where method called)
 - `newName + " Schmo"`, evaluates to Joe Schmo
 - `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

CMSC 131 Fall 2007
Jan Plane (adapted from Bonnie Dorri)

2

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 of the file

```
import java.util.Scanner;
```

- To import class in a package:

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

(Imports `Scanner` as well as other classes in package)

CMSC 131 Fall 2007
Jan Plane (adapted from Bonnie Dorri)

3



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>
- On the resources page of the class web site – javadoc generated descriptions.



String API & Math API

- `String` implements lots of string functions
 - `StringExample.java`
- `Math` implements lots of mathematical functions
 - `MathExample.java`



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



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



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

Separate: API and the workings of the class



- Design so that
 - you can change how the class works without having to change the API
 - the only things in the API are things the user will absolutely need (make the interface as simple as possible)
- Demonstrations in Class
 - Significantly Modifying the Student class – without changing the API (or the driver)
 - The Cat class and its drivers
 - with adding a copy constructor
 - Project 3
 - API described – you are using those classes
 - documentation / comments needed

CMSC 131 Fall 2007
Jan Plane (adapted from Bonnie Dorri)

10

Comments



- Explanations you add into your code
- Three forms in Java
 - `/*`
Put text here
`*/`
 - `//` Put text here to end of line
 - `/**`
Put text here
`*/`
- Last form is special comment for `javadoc` (utility for generating documentation from comments)

CMSC 131 Fall 2007
Jan Plane (adapted from Bonnie Dorri)

11



Comments (cont.)

- Some programmers hate them ...
- ... but they are essential for code understanding (it is called “code” for a reason)
- Beginning with Project #3, you will be graded on the content of your comments as well as other readability factors
- Comment:
 - Every variable (what is variable for?)
 - Every method (what does method do?)
 - Every class (what is class for? who wrote it?)



Testing

- insert set #8



Overloading Methods

- Two methods with the same name but distinguishable parameter types
- Terminology:
 - prototype: `public static void f(int x, float y)`
 - signature: `f(int , float )` // names of variables and the return type are not part of it
- Can't differ only in return type
- Can't differ only in types that can be implicitly promoted between (widening conversion) when choices are equal in amount of conversion

CMSC 131 Fall 2007
Jan Plane (adapted from Bonnie Dorri)

14



Implicit Promotions

- On general assignments
- On parameters
- On return value

CMSC 131 Fall 2007
Jan Plane (adapted from Bonnie Dorri)

15



Most SUBTLE example:

- Consider overloading like this:

```
void f(int x, double y)
```

```
void f(double a, int b)
```

- Three Calls:

- `f(3, 10.0)`
- `f(7.7, 5)`
- `f(3, 7)`



this

- a reference to the current object. (Only makes sense in a non-static method.)
- In an instance method, this is the object that is assumed
 - easy to refer to members (data or methods) using the assumed object
 - difficult to refer to the whole object without having a name to call it
- Only use when needed – using it all the time makes the code more difficult to read