Review of classes and subclasses

M fast through this material, since by now all have seen it in CS100 or the Java bootcamp

First packages
Then classes
Then subclasses

Goal: to give you complete understanding about how objects, methods, method calls are implement.

Raise discussion of classes and subclasses above the level of the computer and talk instead about **classes** as drawers in a filing cabinet, **objects** as manila folders that go in file drawers, and **references or pointers** as labels or names placed on manila folders.

This makes concepts easier to grasp, to talk about; loses nothing.

1

Packages

Read Weiss, section 3.6, "packages", for reference

Package: collection of .java source files (and other packages and interfaces) that are grouped together in the same directory (folder).

Package java.lang contains classes that you can automatically use:

•wrapper classes: Integer, Boolean, etc.
•Math: contains functions you can use, like abs and sqrt
•String and StringBuffer
•System
•Throwable, Error, Exception (discuss later)

2

Package java.io contains classes for doing input/ouput. We'll discuss this a bit in recitations.

To use these, you should "import" them.

Put the command

import java.io.*;

at the top of a file that contains a class that will use a class in this package.

```
import java.io.*;
public class Ex {
        public void m(…) {
            ...
        }
}
```

3

Other useful packages

You have to import these packages. We'll use many of these later in the course.

- java.applet: used for applets
- java.awt: used in constructing GUIs
- javax.swing: the more modern classes for constructing GUIs
- java.util: classes for dates, calendars, locales, random numbers. Class Vector. Classes for sets, lists, maps

4

You can make your own packages

Default package, say classes C1, C2, C3

Package mine, say classes K1, K2

File structure:

main directory:
- C1.java
- C2.java
- C3.java
- mypack (a directory)
  - K1.java
  - K2.java

---

file K1.java

```
package mypack;

public class K1 {

}
```

file K2.java

```
package mypack;

public class K2 {

}
```

5

Visibility levels

```
public int w;
private int x;
protected int y;
/* package */ int z;
```

private: visible only in the class.
/* **package** */: visible only in the package in which it appears.
protected: visible in the package in which it appears and in subclasses.
public: visible anywhere.

Note: You cannot use the keyword package as a prefix on a declaration. That is why we have placed comments around it. Visibility "package" is the default, when no access modifier is given.
Note: You can place these modifiers on fields, methods, and classes

6

Review of classes

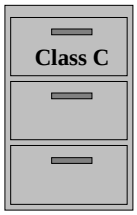
Why review? To make sure that you and I are using the same terminology and the same concepts of class and related issues. Use this example:

```
public class C {
    public static final int ten= 10;
    private int y;

    // Constructor: instance with y = yp
    public C (int yp)  { y=  yp; }

    // specification of method p
    public static void p(int x) {
        // body of method goes here
    }

    // specification of function f
    public int f(int y) {
        // body of function f goes here
    }
}
```

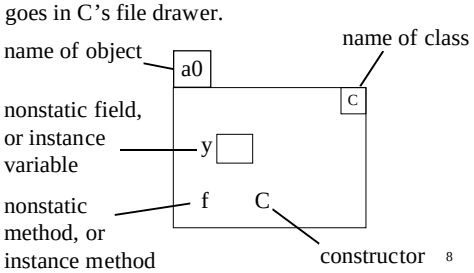
7

Notes on the class on previous slide

1. A class is a drawer of a file cabinet. It contains (at least) the static entities defined in the class. In this case, ten and p.



2. A class is a template for objects of the class. Every object of C is a manila folder of the form given below. The manila folder contains all nonstatic entities declared in class C. The manila folder goes in C's file drawer.



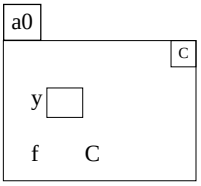
8

Draw objects as manila folders

You MUST draw each object as shown below

As a manila folder with:

- class name in box in upper right corner
- name of object on the tab of manila folder
- each nonstatic field as a variable in the folder
- the name of each nonstatic method in the folder

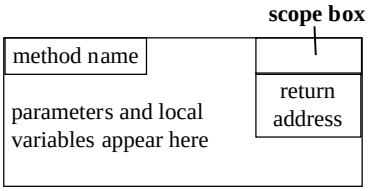


9

The frame for a method call

All method calls occur within methods, which are defined in classes. We now explain the use of the "scope box" in the frame:

Scope box used during execution of method body to find variables and methods that are referenced



Scope box contains, for a:

- **nonstatic method**: the name of the instance in which the method appears
- **static method**: the name of the class in which it is defined
- **constructor**: name of newly created object

10

Execution of procedure call

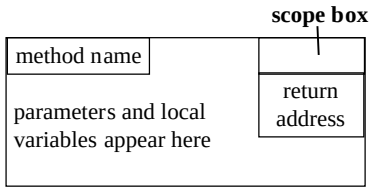
1. Evaluate args and push them onto call stack.

2. Push locations for the rest of the frame for the call onto the stack.

3. Put in frame: name of method, local vars, return address, and scope box --filled in correctly (see slide 10).

4. Execute method body --look in frame at top of stack for variable or method. If not there, use scope box to determine where to look next; **if in an object, search from bottom to top**.

5. Pop frame from call stack; continue executing at the return address in popped frame.



11

Sample execution of proc call --do in class

```
// An instance maintains the number of walks
// and hits of a baseball player
public class C {
    private int y;       // number of walks
    private int x= 0;  // number of hits

    // Constructor: instance with yp walks, 2 hits
    public C (int yp)  { y=  yp; x= 2; }

    // = number of hits
    public int hits()    {  return x;  }

    // = number of hits + number of errors
    public int hitErr()   { return x;}
}
```

```
public class M {
    public static void main (String[] pars) {
        b= new  C(5);
        c= new  C(4);
        d= c.hits();
    }
}
```

12

Memorize for quiz on Tuesday, 18 Sept.

It's important for understanding how method calls and objects work that you memorize:

(1) format of a frame

(2) format of an object --manila folder

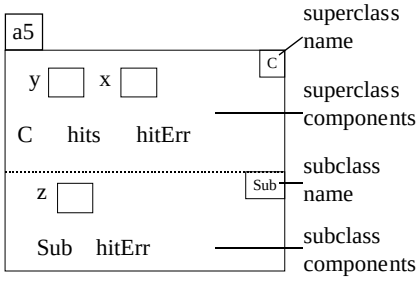
(3) evaluation of a new expression
new C(…)
(a) create a new instance of class C
(b) execute method call C(…), putting in the scope box of the frame the name of the newly created object

(4) Steps in executing a method call (slide 11).

13

Drawing an instance of a subclass

```
// Extends class C on slide 12
public class Sub extends C {
    private int z; // number of errors

    // Constructor: instance with yp walks, 2 hits,
    // and zp errors
    public Sub(int yp, int zp)
        {  super(yp);  z= zp; }

    // = number of hits + number of errors
    public int hitErr() { return hits() + z;}
}
```

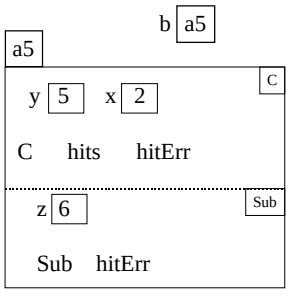


14

Overriding a method

Consider: Sub b= new Sub(5,6);
b.hitErr();

Which method hitErr is called? Our rules for execution of calls (slide 11) say the one below the line --the one in subclass Sub. It overrides the other. In class, we execute this call.



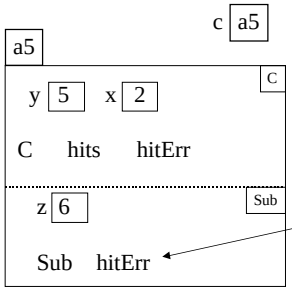
15

Casting

Consider: C c= **new** Sub(5,6);

Instance a5 automatically cast to C, since c is of class-type C. Apparent type of a5 (using c) is C, real type is Sub.

Legal	**Illegal**	**Using c, reference only names accessible in the class of C**
c.y	c.z	
c.x		
c.hits()		
c.hitErr()		



But our rules say that c.hitErr() refers to this!!!

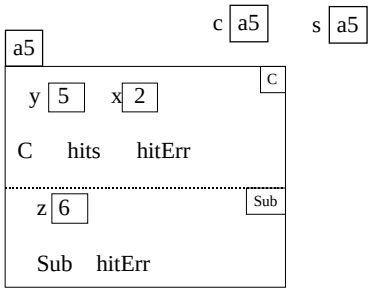
16

Casting

Consider: C c= **new** Sub(5,6);
Sub s= (Sub) c;

Explicit cast of c to subclass s. Using s, one can reference everything in object a5.

Legal	**Illegal**	**Legal**
c.y	c.z	s.z
c.x		s.hits()
c.hits()		s.hitErr()
c.hitErr()		



17