CS 1110
Final Exam: Review Session 1

Drawing frames for
calls, executing method calls

Biggest issue!!! You can't do questions on this topic correctly unless you draw variables, draw objects when they are created, and draw frames for method calls.

Learning to do this will help you to do the same thing when trying to find errors in your programs.

Understanding execution of

- *local variable declaration (in a method body)*
- *new expression (3 steps)*
- *method call (method frames, call stack)*

- examples from previous exams
  - code execution (Q4 from 2008 fall final, modified)
  - method call (Q3 from 2007 fall final)

Important!

- All previous finals included some questions about code execution
  - You need to know how to draw variables, objects, method frames …
- *The purpose of such questions on executing statements with new expressions and method calls is to test your understanding of how java programs are executed*

---

code segment (in a method body)

```
int a= 3;
C x= new C(a);
C y= new C(a);
x= y;
```

The first thing to do?

draw all local variables

---

code segment (in a method body)

```
public class C {
    private int f;
    public C(int k) { f = k; }
}
```
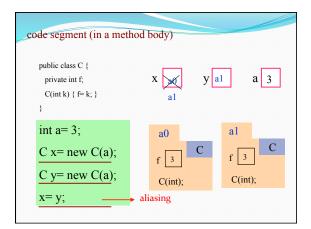
x ☐   y ☐   a 3

```
int a= 3;
C x= new C(a);
C y= new C(a);
x= y;
```

Evaluation of new expression

- 3 steps in evaluating the new expression   **new** C(args)
  - *create a new folder* (object) of class C with a unique name (place it in the class file drawer)
  - Execute the *constructor call* C(args)
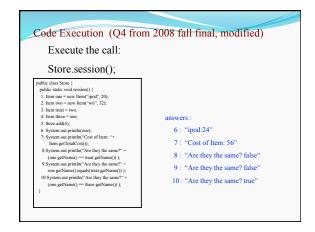 - yield *the name* of the object as *the value of the new expression*

1

Slide 1: code segment (in a method body)

```
public class C {
    private int f;
    C(int k) { f= k; }
}
```

x ⊠ a0   y a1   a 3

a1

```
int a= 3;
C x= new C(a);
C y= new C(a);
x= y;
```

→ aliasing

a0
f 3   C
C(int);

a1
f 3   C
C(int);

Slide 2: variables declared in a loop

```
public void m(int size) {
    for (int i= 0; i< size; i= i+1) {
        int[] b = …;
        …
    }
}
```
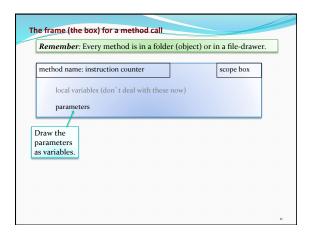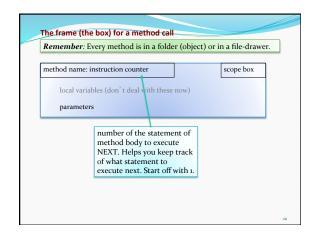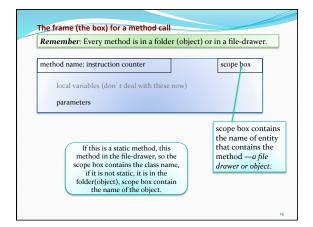
When is local variable b inside the loop created?

During the first step of executing a method call, when the frame for the call is drawn.

Not after the loop starts.

Slide 3: Code Execution (Q4 from 2008 fall final, modified)

Execute the call:
Store.session();

```
public class Store {
    public static void session() {
    1: Item one = new Item("ipod", 20);
    2: Item two = new Item("wii", 32);
    3: Item treat = two;
    4: Item three = one;
    5: three.add(4);
    6: System.out.println(one);
    7: System.out.println("Cost of Item: "+
         Item.getTotalCost());
    8:System.out.println("Are they the same?" +
         (one.getName() == treat.getName()) );
    9:System.out.println("Are they the same?" +
         one.getName().equals(treat.getName()) );
   10:System.out.println("Are they the same?" +
         (one.getName() == three.getName()) );
    }
}
```

```
public class Item {
    /* total cost of all items created */
    private static int totalCost = 0;
    private int cost;  // cost of this item
    private String name;  // title

    /** Constructor: new Item with name t, cost c */
    public Item(string t, int c) {
        name = t;    cost = c;
        totalCost = totalCost + c;
    }
    /** = Cost of this item */
    public int getCost() { return cost; }
    /** = this item's name */
    public String getName() { return name; }
    /** = "<name>:<cost>" */
    public String toString() { return name + ":" + getCost(); }
    /** Add d to this item's cost */
    public void add(int d) {
        cost = cost + d;    totalCost = totalCost + d;
    }
    /** = the total cost of all Items */
    public static int getTotalCost() { return totalCost; }
}
```

Slide 4: Code Execution (Q4 from 2008 fall final, modified)

Execute the call:
Store.session();

```
public class Store {
    public static void session() {
    1: Item one = new Item("ipod", 20);
    2: Item two = new Item("wii", 32);
    3: Item treat = two;
    4: Item three = one;
    5: three.add(4);
    6: System.out.println(one);
    7: System.out.println("Cost of Item: "+
         Item.getTotalCost());
    8:System.out.println("Are they the same?" +
         (one.getName() == treat.getName()) );
    9:System.out.println("Are they the same?" +
         one.getName().equals(treat.getName()) );
   10:System.out.println("Are they the same?" +
         (one.getName() == three.getName()) );
    }
}
```

answers :

6 : "ipod:24"

7 : "Cost of Item: 56"

8 : "Are they the same? false"

9 : "Are they the same? false"

10 : "Are they the same? true"

Slide 5: The frame (the box) for a method call

Remember: Every method is in a folder (object) or in a file-drawer.

method name: instruction counter	scope box

local variables (don't deal with these now)

parameters

Draw the parameters as variables.

Slide 6: The frame (the box) for a method call

Remember: Every method is in a folder (object) or in a file-drawer.

method name: instruction counter	scope box

local variables (don't deal with these now)

parameters

number of the statement of method body to execute NEXT. Helps you keep track of what statement to execute next. Start off with 1.

Slide 13

The frame (the box) for a method call

Remember: Every method is in a folder (object) or in a file-drawer.

method name: instruction counter | scope box

local variables (don't deal with these now)

parameters

scope box contains the name of entity that contains the method —*a file drawer or object*.

If this is a static method, this method in the file-drawer, so the scope box contains the class name, if it is not static, it is in the folder(object), scope box contain the name of the object.

13

Slide 14

To execute the call x.setScore(100);

1. Draw a frame for the call.

2. Assign arguments to the parameters (in the frame).

3. Execute the method body. (Look for variables in the frame; if not there, look in the place given by the scope box.)

setScore: 1 | a0

value | 100

4. Erase the frame for the call.

x | a0 | Score

a0 | Score

score | ~~10~~ 100

setScore(int value) { score= value;}

getScore() {...}

14

Slide 15
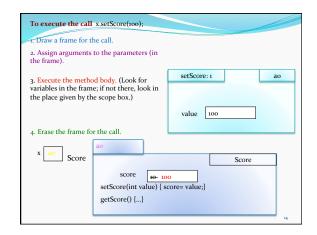
Scope of local variable: the sequence of statements following it within the containing "block".

```
/** = the max of x and y */
public static int max(int x, int y) {
    // Swap x and y to put the max in x
    if (x < y) {
        int temp;
        temp= x;
        x= y;              scope of
        y= temp;           temp
    }
    return x;
}
```

You can't use temp down here
This is an error.

15

Slide 16

Scope of local variable: the sequence of statements following it.

```
/** s contains a name in the form exemplified by "David   Gries".
    Return the corresponding String "Gries, David".
    There may be 1 or more blanks between the names. */
public static String switchFormat(String s) {
    // Store the first name in variable f and remove f from s
    int k;       // Index of the first blank in s
    k= s.indexOf(' ');
        String f; // The first name in s.
        f= s.substring(0, k);
        s= s.substring(k);

    // Remove the blanks from s      scope of k
        s= s.trim();

    return s + ", " + f;
}
```

declaration
assignment

scope of f

16

Slide — Call Stack

Call Stack

Call Stack is the stack of frames for uncompleted method calls, a frame for a method call lasts as long as the method call is being executed. When the call is finished, the frame is erased.

This fact explains why local variables do not retain their values from one call of a method to the next call of the same method:
All the information about the first call is in a frame, and the frame is erased when the call is completed.

Slide — Exercise 1

Question 3 (12 points) Executing method calls. Suppose `Vector` v of Integers contains 3 elements, as shown to the right. The 3 elements are the names of `Integer` objects wrapping the three **int** values 5, 7, and 3.

Execute the method call
        VectorTools.reverse(v);

where class `VectorTools` is given below. (We have labeled the statements with numbers (e.g. 2:), which you can use as program counters in a frame for a call. Stop executing when you are ready to execute the return statement that is labeled 2.

For each call during execution (except the calls on methods in class `Vector`), draw the frame for the call. Note that elements of `Vector` v will change, and you should record those changes in object a1.

Hint: You will have to draw 2, 3, or 4 frames for calls.

```
public class VectorTools {
    // Reverse v[0..v.size()]
    public static void reverse(Vector v) {
        1: reverse(v, 0, v.size()-1);
    }

    // Reverse the segment v[h..k]
    private static void reverse(Vector v, int h, int k) {
        1: if (h >= k)
            2: return;

        3: Integer w= v.get(h);
        4: v.set(h, v.get(k));
        5: v.set(k, w);
        6: reverse(v, h+1, k-1);
    }
}
```

a1
Vector

v | a1

0 | a2
1 | a4
2 | a0

a2 | a4 | a0
Integer | Integer | Integer
5 | 7 | 3

Note: For v a Vector,
v.size() = number of elements in v.
v.get(i) = the value of element v[i]
v.set(i,w) sets v[i] to w.

Exercise 1

3

Step Into vs Step Over

Step Into:
Draw the frame for the call and execute the method call

Step Over:
Assume the function is doing exactly what it should do based on the specifications of the function.