CS 1110
Final Exam: Review Session 1

Drawing frames for
calls, executing method calls

Understanding the 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?

to draw all local variables

code segment (in a method body)

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

x y a 3

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

Execution of new expression

- 3 steps for executing the new expression
  - ***create a new folder*** (object) of the class with a unique name (place it in the class file drawer)
  - initialize the fields in the object by executing the ***constructor***
  - yield ***the name*** of the object as ***the value of the new expression***

code segment (in a method body)

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

x ~~a0~~ a1 y a1 a 3

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

aliasing

a0 C f 3 C(int);

a1 C f 3 C(int);

variables inside a loop

```
public void m(int size) {
  for (int i=0; i< size; i++) {
    int [] b = …;
    …
  }
```

When is the local variable b inside the loop created?

It is created once with other local variables when we start executing the method (in the method frame).

Not after the loop starts.

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; }
```

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"

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.

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.

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

If this is a static method, this method should be in the file-drawer, thus, the scope box should contain the class name, if it is not static, it is in the folder(Object), the scope box should contain the object.

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

13

To execute the call x.setScore(100);

1. Draw a frame for the call.
2. Assign the value of the argument to the parameter (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.)
4. Erase the frame for the call.

setScore: 1 | a0

value 100

x a0 Score

a0 | Score

score ~~10~~ 100

setScore(int value) { score = value;}

getScore() {...}

14

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;
        y= temp;
    }

    return x;
}
```

scope of temp

You can't use temp down here

This is an error.

15

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
declaration  int k;      // Index of the first blank in s
assignment   k= s.indexOf(' ');
t            String f;  // The first name in s.
             f= s.substring(0, k);
             s= s.substring(k);

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

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

scope of k

scope of f

16

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.

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.

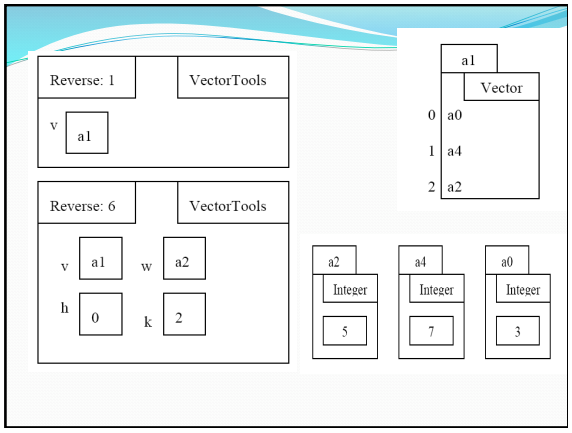
```
public class VectorTools {
    // Reverse v[0..v.size()]
    public static void reverse(Vector v) {
        1: reverse(v, 0, v.size()-1);
    }

    // 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);
    }
}
```

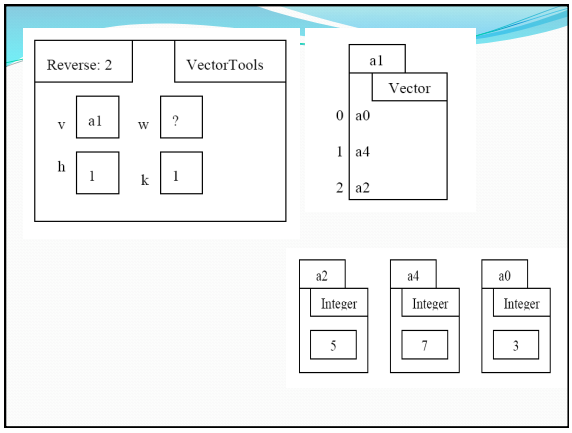
v a1

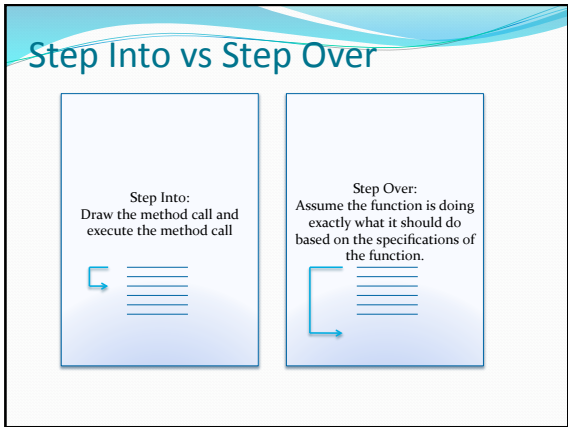
a1 Vector: 0 a2, 1 a4, 2 a0

a2 Integer 5 | a4 Integer 7 | a0 Integer 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.


Reverse: 1
VectorTools
v
a1
Reverse: 6
VectorTools
v
a1
w
a2
h
0
k
2
a1
Vector
0
a0
1
a4
2
a2
a2
Integer
5
a4
Integer
7
a0
Integer
3


Reverse: 2
VectorTools
v
a1
w
?
h
1
k
1
a1
Vector
0
a0
1
a4
2
a2
a2
Integer
5
a4
Integer
7
a0
Integer
3


Step Into vs Step Over
Step Into:
Draw the method 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.