CS61B, Fall 2002 **Discussion #4** **Amir Kamil**
UC Berkeley **9/19/02**

Topics: Review

1 Review

1.1 Assignment

In Java, when one variable is assigned to another, the value contained in the variable on the right side of the assignment is copied over to the variable on the left side. Consider:

```
int x = 4;
int y = x;
y = 1;
```

What is the value of `x` after this code has run? Figures 1 to 3 show the execution of the code. Modifying `y` does not affect `x`'s value, since each variable has its own container.

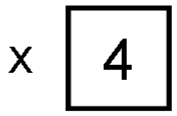

Figure 1: The value of `x` following `int x = 4`.

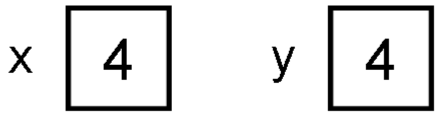

Figure 2: The value of `x` and `y` following `int y = 4`.

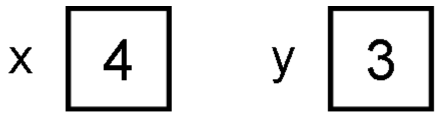

Figure 3: The value of `x` and `y` following `y = 3`.

1.2 Arrays

1.2.1 Simple (1D) Arrays

- See 9/5/02 notes, section 7.
- Arrays hold the same values as variables, but an array can contain multiple values. This means that a primitive array holds primitive values, while an Object array holds pointers.
- Primitive numerical arrays (`int[]`, `double[]`, etc.) are initialized by default to hold all zeroes.

- Primitve `boolean` arrays are initialized by default to hold `false` in each location.
- Object arrays are initialized by default to hold `null` in each location.
- An array of zero length is not `null`; it has a `length` field, and you can call methods on it.

1.2.2 Multi-Dimensional Arrays

- While it is sometimes useful to think of a two dimensional array as a matrix, that is not what it is. It is actually an array of one dimensional arrays. For example, `conifer(3)` returns the structure in figure 4.

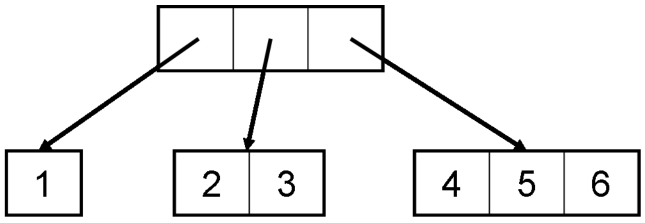

Figure 4: The return value of `conifer(3)`.

- For a general array `x`, `x[i]` returns the `i`th element in the array. The same is true for two dimensional arrays; `x[i]` would return the `i`th one dimensional array that it contains.
- This can be generalized to higher dimensions. For example, an `int[][][]` is an array that holds arrays that hold `int` arrays.

1.3 Inheritance

- See 9/12/02 notes, section 1.
- For convenience, I have reproduced all the computer code we've written so far.

  The initial `Computer` class we wrote:

```
/* A simple example of a Java class */
/* Imports classes from external packages */
import java.util.*;
public class Computer {

  /* Fields */
  int state = 0; // Let 0 = off, 1 = on, -1 = crashed.
  String os; // The operating system on the computer.
  String user;
  // Initially an empty list; no programs running.
  LinkedList programs = new LinkedList();

  /* Turns on the computer, then boots it. */
  public void turnOn() {
    state = 1;
    boot();
    return;
  }

  /* Boots the computer */
  private void boot() {
    if (os.equals("Windows 95")) {
      state = -1; // crash
    } else {
      programs.add(os);
    }
  }

  /* Logs a user into the computer */
  public void login(String user) {
    this.user = user;
  }

  /* Constructor */
  public Computer(String os) {
    this.os = os;
    user = "no one";
  }

}
```

The `Computer` class and its descendents, using abstract classes and interfaces:

```
public abstract class Computer {

  protected int state;

  protected String os;

  protected Vector programs;

  public void turnOn() {
    state = 1;
    boot();
  }

  protected abstract void boot();

  ... // other methods, fields

}

public class PC extends Computer {

  protected void boot() {
    if (os.equals("Windows 95")) {
      throw new Exception("CRASH!");
    } else {
      programs.addElement(os);
    }
  }

}

public interface Laptop {

  public void charge(int min);

  public void discharge(int min);

  public void getStolen();

}

public class PCLaptop extends PC implements Laptop {

  private int remainingTime;

  public void charge(int min) {
    remainingTime += min;
  }

  public void discharge(int min) {
    remainingTime += min;
  }

  public void getStolen() {
    throw new Exception("Finders keepers, loser weepers.");
  }

  ... // other methods, fields

}
```

1.4 Exceptions

- See 9/12/02 notes, section 4.