PURDUE UNIVERSITY

CS18000: Problem Solving And Object-Oriented Programming

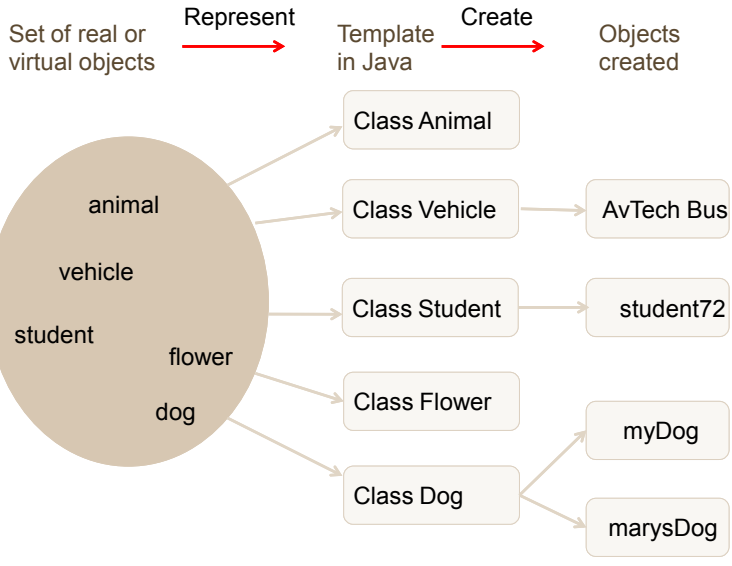
Class (and Program) Structure

31 January 2011

Prof. Chris Clifton

COMPUTER SCIENCES

Classes and Objects

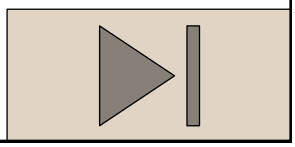


2

Basic Class Structure

- Everything in Java starts with a Class
  - One class has a "main" method
- May have data associated with it
  - variables
  - constants
- Statements oc methods
  - May also have declarations
- Constructor method called when class created
  - Initializes the object
  - May have arguments

```
class Program {
    public static final String Author =
        "Chris";
    Date timeRun;
    Program( String user ) {
        timeRun = new Date();
    }
    System.out.println(
        timeRun.getTime() );
    public static void main (String[ ]
        args) {
        Program p;
        p = new Program( args[0] );
        System.out.println(
            p.timeRun.getTime() );
    }
}
```



Variables

A variable is something whose value may change during program execution.

Every variable has a name and a type.

Every variable must be declared before it is used.

Declarations

```
int age;

float  height, area;

String name

boolean

int x=1, y=0;

String firstName="Harry";
```

5

Names

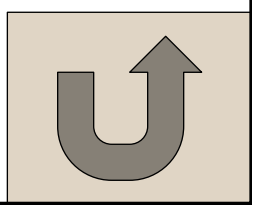
Used to denote classes, objects, data

Contain characters; must start with a letter, or a $ sign or an underscore.

Examples: height, area1, Dog, $great

Length unlimited, case sensitive.
Dog and dog are different names.

Convention: All class names begin with an uppercase letter; all other names begin with a lower case letter.



Constants

A constant is something that cannot change during program execution.

Examples:

Integer constants: 0, 1, -1, +24, 29, 300009998, O14, 0x1B

Floating point constants: 0.0, -2.345e28, -0.000976512

Boolean constants: true, false

Character constants: ' ', 'a', 'A', '$'

String constants: "", " ", "Hi!", "Alice in Wonderland"

7

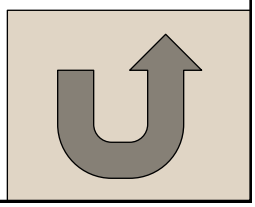
Named Constants

A constant can be named and the name used instead of the constant itself.

Examples:

final float pi=3.14159;

final boolean dogsExist=true;



Simple expressions

Expressions are used to compute "something".

```
float  x, y, z;

x*y+z; // Arithmetic expression, results in float value

x<y;  // Boolean expression, results in boolean
value

String firstName="Mary", lastName= "Jones";

firstName+" "+lastName; // Results in a string
```

More in Chapter 2! And yet more to come!

9

Assignment statement

An assignment statement allows assigning the value of an expression to a variable.

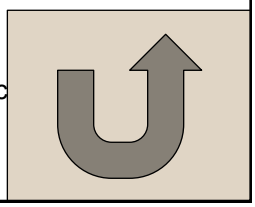
```
float p=x*y+z; // p gets the value of x*y+z

boolean q=x<y;  // q gets the value of x<y

String firstName="Mary", lastName= "Jones";

String name= firstName+" "+lastName;
```

More in Chapter 2! And yet more to c



Parameters and Arguments

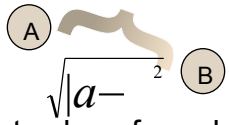
- Program (String user) { … }
  - user is a *formal parameter*
  - Used like a variable inside the method
  - Value provided when called
- Program p = new Program (“Chris”) { … }
  - “Chris” is the *argument* to the method
  - *for this call*, in the method, user has the value “Chris”

Example in a Program: *Concurrent* distance

- How can we speed up the Euclidean distance program?
  - *We have multiple processors, use them to compute parts of the sum simultaneously!*
- Java Thread class:
  - Set up what a thread is supposed to do (constructor)
  - Start it (returns immediately)
  - Do other things
  - Wait for it to end and get result

Problem Solving: Euclidean Distance

- What is the problem?
  - What would be a solution?
- Steps to a solution
  - Input
  - Calculations
  - Output
- Data representation
- Function breakdown
- *Now we start coding*



- Get values for a, b
  - Represent as Array
- Compute distance
  - Sum squares of each dimension
    - Divide into parts
    - Sum each separately
    - Add parts together
  - Take square root
- Display result

Using Threads

- Like running a program
  - "run" instead of "main"
  - takes no arguments
- Object must know what it is supposed to do
  - Constructor
- Result stored in object
- Create object with portion of a, b to sum
  - Constructor takes arrays and saves in object
    - Variable in scope of Class
  - "main" divides and creates objects
- run does the summation
  - Saves result in variable

Example: Euclidean Distance

```
public class Lec3Distc extends Thread
{
  public double distance = 0;
  private double x[], y[]; // Arrays to compute distance on
  private int begin,end; // Dimensions to compute on

  public Lec3Distc(double a[], double b[],int bg, int en)
     // Set up what this instance is supposed to do when it runs.
  {
    x = a;
    y = b;
    begin = bg;
    end = en;
  }

  public void run()
     // What to do when "start" is called (from Thread class)
  {
    distance = SumSquareDiff(x,y,begin,end);
  }

  public static double SumSquareDiff(double[] a, double[] b, int begin, int
        end)
    // Requires:  a.length = b.length; no null values in a or b
    // Produces:  Euclidean distance between a and b (>0)
  {
    double sum = 0;
    for (int i=begin; i<end; i++) {
      sum = sum + (a[i]-b[i])*(a[i]-b[i]);
    }
    return sum;
  }
```

```
public static double EuclideanDistance(double[] a, double[] b)
    // Requires:  a.length = b.length; no null values in a or b
    // Produces:  Euclidean distance between a and b (>=0)
  {
    Lec3Distc first = new
        Lec3Distc(a,b,0,(int)Math.floor(a.length/2));
    Lec3Distc second = new
        Lec3Distc(a,b,(int)Math.floor(a.length/2)+1,a.length);
    first.start();  // Start computation on the first half, but don't
        wait
    second.start();                  // Start computation on the
        second half, don't wait
    try {
      first.join();  // Wait for the first half to finish.
      second.join();                 // Wait for the second half to
        finish.
    } catch(InterruptedException e) { /* Ignore */ }
    return Math.sqrt(first.distance+second.distance);
  }
```

- Rest of class (main) is unchanged
  - *This is Functional Abstraction in action!*