PURDUE UNIVERSITY

CS18000: Problem Solving And Object-Oriented Programming

Types, Methods

24 January 2011

Prof. Chris Clifton

COMPUTER SCIENCES

Today We Learn

- Classes and Objects vs. Primitive Types
- Functions as Abstractions

Primitive Types

Primitive Type

- Integers
  - byte, short, int, long
- Rationals
  - float, double
- Boolean
  - boolean
- Character
  - char

Class

- Number
  - Byte, Short, Integer, Long
  - BigInteger
- Number
  - Float, Double
- Boolean
- Character

1/26/2011 CS18000 3

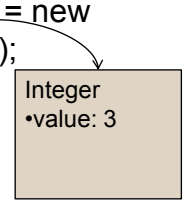
Primitive Type vs. Class

Primitive Type

- *Is* a value
- int a = 3;
- Operators on value
  - +, -

Class

- Create Objects
  - Object has values
- Integer a = new Integer(3);



- Has *methods*
  - compareTo(), …

1/26/2011 CS18000 4

Primitive Type vs. Class

- Primitive types are "special"
  - literals (constant values)
  - infix operators (+, -)
  - …
- Classes used to generate objects
  - double 3. vs. object of type Double containing the value 3.
- *For now, big issue is equality test*
  - a.equals(b) vs. a == b
  - == : variables a and b refer to the same object
  - *Two different objects can have same value!*

1/26/2011 CS18000 5

Methods

- Class has *methods*
  - Operations on objects
  - Specific to objects of that class
- Method works with an object
  - May have arguments
  - May produce result
- *Functional Abstraction*

Integer
value: 3
MAX_VALUE: 2^{31}-1
MIN_VALUE: -2^{31}

int intValue()
float floatValue()
boolean equals(Object)
int compareTo(Integer)

1/26/2011 CS18000 6

Abstraction: How Programming Scales

- Small programs are easy to write
  - You've done this
- Large programs a challenge
  - Too much to keep track of
  - *We can prove it!* (take CS 30700 for more…)
- Solution: Write small programs
  - Combine them to form larger programs
- *This only works if we don't worry how the small programs work!*

1/26/2011 CS18000 7

Functional Abstraction

- Function provides a capability
  - We use that capability
  - We don't care how it is done!
- What do we care about?
  - Input required
  - Output produced
- *Running Example: Euclidean Distance*

  double EuclideanDistance(double[] a, double[] b)

1/26/2011 CS18000 8

Array types

- What is double**[ ]** a ?
  - [ ] is an *array*
  - Contains a sequence of objects of that type
- Creating an array
  - double [ ] doubleArray = new double[20];
  - double [ ] shortArray = { 4.0, 18.5, -3.2 };
- Using values
  - doubleArray[2] = 17.0;
  - doubleArray[1] = doubleArray[2] * 4.0;
  - *Array and array index gives a value*

1/26/2011 CS18000 9

Functional Specification: Input

- What do we specify about input?
  - Number of inputs
  - Data type
  - Permissible values
  - Meaning

```
double EuclideanDistance(double[] a, double[] b)
// Requires:a.length == b.length
//              no null entries in a or b
```

1/26/2011 CS18000 10

Functional Specification: Output

- What do we specify about output?
  - Number of outputs (is input changed?
  - Data type
  - Permissible values
  - Meaning

```
double EuclideanDistance(double[] a, double[] b)
// Requires:a.length == b.length
//          no null entries in a or b
// Returns: Euclidean distance between a and b
//          result > 0
```


PURDUE UNIVERSITY

CS18000: Problem Solving And Object-Oriented Programming

Declarations, Problem Solving

26 January 2011

Prof. Chris Clifton

COMPUTER SCIENCES

Variable Declarations

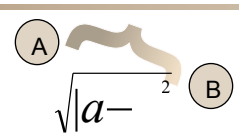
- Variables must be *declared*
  - Defines type variable can hold
  - Must be done before use
- Must be initialized before use
- Use in *scope* of declaration
  - *Statement* where defined
  - Generally { }
- Good Practice: Declare at start of class/method

```
static void main( String[ ] args ) {
    int i = 0;
    double gpa;

    gpa = i + 4;

    for ( int j=0; j<5; j++) {
        i = i + j;
    }
    j = 0;
}
```

1/26/2011 CS18000 13

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
  - Take square root
- Display result

1/26/2011 CS18000 14

Example: Euclidean Distance

```
import java.lang.Math;
import java.util.Date;
import java.util.Scanner;

public class Lec3Dist
{
  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)
  {
    double distance = 0.;
    for (int i=0; i<a.length; i++) {
      distance = distance + (a[i]-b[i])*(a[i]-b[i]);
    }
    return Math.sqrt(distance);
  }
  public static final int DIMENSIONS = 10000000;
  public static void main(String args[])
  {
    double[] a = new double[DIMENSIONS];
    double[] b = new double[DIMENSIONS];

    for (int i=0; i < DIMENSIONS; i++) {
      a[i] = Math.random();
      b[i] = Math.random();
    }

    long time = new Date().getTime();
    double dist = EuclideanDistance(a,b);
    time = new Date().getTime() - time;
    System.out.println("Distance computed is " +
      dist);
    System.out.println("Distance computation took " +
      time + " milliseconds");
  }
}
```

1/26/2011 CS18000 15

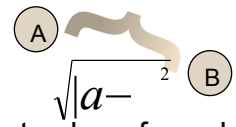
Concurrency

- How can we speed this up?
  - *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

1/26/2011 CS18000 16

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!*