1.00 Lecture 5

More on Java Data Types, Control Structures
Introduction to Methods

Reading for next time: Big Java: 7.1-7.5, 7.8

Floating Point Anomalies

- **Anomalous floating point values:**
  - **Undefined, such as 0.0/0.0:**
    - `0.0/0.0` **produces result** `NaN` **(Not a Number)**
    - **Any operation involving** `NaN` **produces** `NaN` **as result**
    - **Two** `NaN` **values cannot be equal**
    - **Check if number is** `NaN` **by using methods:**
      - `Double.isNaN(double d)` **or** `Float.isNAN(int i)`
      - **Return boolean which is true if argument is** `NaN`
  - **Overflow, such as 1.0/0.0:**
    - `1.0/0.0` **produces result** `POSITIVE_INFINITY`
    - `-1.0/0.0` **produces result** `NEGATIVE_INFINITY`
    - **Same rules, results as for** `NaN` `(Double.isInfinite)`
  - **Underflow, when result is smaller than smallest possible number we can represent**
    - **Complex, not handled very well (represented as zero)**

Example

```
public class NaNTest {
    public static void main(String[] args) {
        double a=0.0, b=0.0, c, d;
        c= a/b;
        System.out.println("c: " + c);
        if (Double.isNaN(c))
            System.out.println("  c is NaN");
        d= c + 1.0;
        System.out.println("d: " + d);
        if (Double.isNaN(d))
            System.out.println("  d is NaN");
        if (c == d)
            System.out.println("Oops");
        else
            System.out.println("NaN != NaN");
        double e= 1.0, f;
        f= e/a;
        System.out.println("f: " + f);
        if (Double.isInfinite(f))
            System.out.println("  f is infinite");
    }
}
```

Doubles as Bad Loop Counters

```
public class Counter {
    public static void main(String[] args) {
        int i= 0;
        double x= 0.0;
        while (x <= 10.0) {
            x += 0.2;
            i++;
            if ( i % 10 == 0 || i >= 48)
              System.out.println("x: " + x + " i: " + i);
        }
    }
}
```

Doubles as Bad Loop Counters

```
i : 10 x : 1.9999999999999998
i : 20 x : 4.000000000000001
i : 30 x : 6.000000000000003
i : 40 x : 8.000000000000004
i : 48 x : 9.599999999999998
i : 49 x : 9.799999999999997
i : 50 x : 9.999999999999996
i : 51 x : 10.199999999999996
```

Notice accumulating, increasing error. Never use floats or doubles as loop counters (well, almost never…)

We went one iteration too many

Exercise

- Create a class AngleTest
  - Loop over angles from θ_{min} to θ_{max} by δ
  - Output the angle
  - Compute and output $1/(\theta_{max} - \theta)$
  - Make sure you get very close to θ_{max}
  - Input $\theta_{min} = 0.1$, $\theta_{max} = 4.0$, $\theta = 0.1$
- How close do you get?
- Does Java catch the zero divide?
- If you have time:
  - Implement this with an int loop counter

Numerical Problems

Problem	Integer	Float, double
Zero divide	POSITIVE_INFINITY, NEGATIVE_INFINITY	POSITIVE_INFINITY, NEGATIVE_INFINITY
0/0	NaN (not a number)	NaN (not a number)
Overflow	No warning. Program gives wrong results.	POSITIVE_INFINITY, NEGATIVE_INFINITY
Underflow	Not possible	No warning, set to 0
Rounding, accumulation errors	Not possible	No warning. Program gives wrong results.

Common, “bad news” cases

More on Control Structures

- Three control structures in Java, or any language, for that matter:
  - Sequence: execute next statement
    - This is default behavior
  - Branching: if, else statements
    - If, else are the primary construct used
    - Switch statement used if many choices
  - Iteration: while, do, for loops
    - Additional constructs exist to terminate loops ‘prematurely’

Switch statement

- Used as substitute for long if-else chains
  - Branch condition must be integer, can't be String, float, etc.
  - No ranges, just single values or expressions in switch
    - C# allows strings as branch condition, but not Java or C++

```
int speed;
switch (speed/10) {        // Limit= 9 mph (bicycle)
  case 3:                  // Drop through to case 2
  case 2:
    System.out.println("Arrest");
  case 1:                  // Drop through to case 1
    System.out.println("Ticket");
    break;                 // Prevent dropping through
  case 0:
    System.out.println("Speed legal");
    break;
  default:
    System.out.println("Invalid radar reading");
}
```

Switch statement

- Used as substitute for long if-else chains
  - Branch condition must be integer, can't be String, float, etc.
  - No ranges, just single values or expressions in switch
    - C# allows strings as branch condition, but not Java or C++

```
int speed;
switch (speed/10) {        // Limit= 9 mph (bicycle)
  case 3:                  // Drop through to case 2
  case 2:
    System.out.println("Arrest");
  case 1:                  // Drop through to case 1
    System.out.println("Ticket");
    break;                 // Prevent dropping through
  case 0:
    System.out.println("Speed legal");
    break;
  default:
    System.out.println("Invalid radar reading");
}
```

Terminating Iteration: Break

- Break statement in for, while or do-while loops transfers control to statement immediately after end of loop

```
public static void main(String[] args) {
    for (int i = 0; i < 20; i++) {
          System.out.println("i: "+i);
          if (i*i > 20)
                break;        // End loop
    }
    System.out.println("Done");
}

// If break in inner, nested loop, control is
// transferred to the outer loop
```

Terminating Iteration: Continue

- Continue statement jumps to end of loop but continues looping

```
public static void main(String[] args) {
    for (int i = 0; i < 7; i++) {
          System.out.println("i: "+i);
          if (i*i < 17)
                continue;     // Skip rest of loop
          System.out.println("i: "+i);
    }
    System.out.println("Done");
}

// If continue in inner, nested loop, control stays
// in inner loop
```

Control exercise

- Write a class LoopExercise:
  - Main() method has:
    - Loop over int i going from 0 to 8
      - Make j = i^2-1
      - If j negative, skip the rest of the loop
      - Find s= square root of j
      - If s > 4, end the loop
      - Output i, j and s to see what's happening
    - Print "Done" at the end of the program

Java Methods

- Methods are discrete units of behavior
  - You've already used some:
    - JOptionPane()
    - Math.sqrt()
    - System.out.println()
  - You'll write your own for the rest of the term, as part of classes
  - Right now, you are writing classes but they only have a main() method and they create no objects
  - We'll write additional methods in our classes
  - (And then create objects that have methods)
  - For now, our methods will have the keywords `public static` in them
    - Treat them as an incantation for this and the next lecture

Why Use Methods

- Methods provide a way to invoke the same operation from many places in your program, avoiding code repetition
- Methods hide implementation details from the user of the method
- Variables defined within a method are not visible to users of the method; they have local scope within the method. (More on scope next time)
- The method cannot see variables in the program component (e.g., main()) that calls it either. There is logical separation between the two, which avoids variable name conflicts.
- Methods (and objects) allow programs to scale

Method example

```
public class MethodExample {
   public static void main(String[] args) {
       double boxWeight= 50;
       double boxCube= 10;
       String boxID= “Box A”;
       double density= getDensity(boxWeight, boxCube);
       System.out.println("Density: "+ density);
       printBox(boxWeight, boxCube);
   }
   public static double getDensity(double bw, double bc) {
        double result= bw/bc;  // 'result' could be 'density'
        return result;
   }
   public static void printBox(double w, double c) {
        System.out.println("Box weight: "+w+" cube: "+c);
        System.out.println(" Density: "+getDensity(w,c));
//      System.out.println(“ ID: “+boxID);  // No access to ID
   }                                        // Won’t compile!
}
```

Passing Arguments

```
main(…){
   double boxWeight= 50;
   double boxCube=10;
   String boxID= "Box A";
   double density=getDensity
      (boxWeight, boxCube);
   …
```

Return value

Argument 1

Argument 2

```
double getDensity(double bw, double bc)
{        // Method makes its own copy
         // of arguments bw and bc
Double result= bw/bc;
return result;}
```

Communi-cation only via arg list, return value

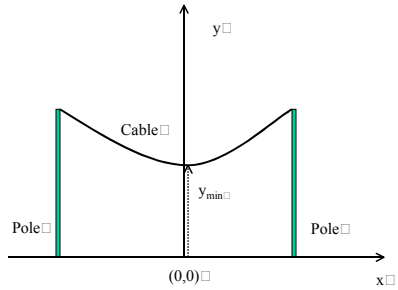
Arguments matched by position

Assume method is written first: can't know main() vars

Method exercise

- Write a class MethodExercise
  - Main() method:
    - Declares String name, int age, double height
    - Sets variables to your name, age, height
    - Calls isOldEnough() method
    - Calls printInfo() method
  - Method isOldEnough() returns true if age >= 21, false otherwise
  - Method printInfo prints name, age, height
  - Choose appropriate arguments, return values

Homework 2

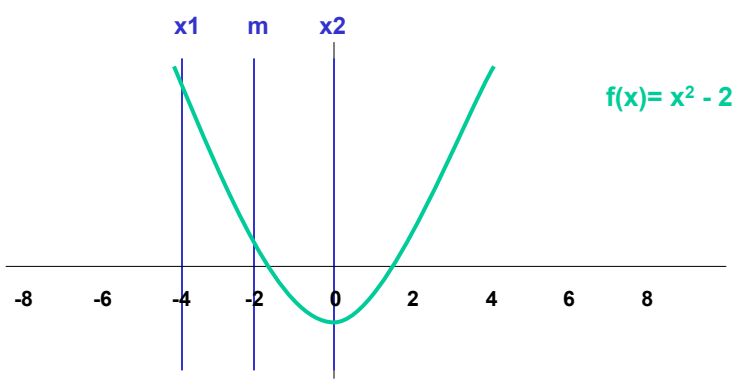


$$y = \frac{T}{w}\cosh\left(\frac{wx}{T}\right) + y_{min} - \frac{T}{w}$$

$$\cosh(x) = 0.5(e^{x} + e^{-x})$$

Solve for T using bisection: find T s.t. f(T)= 0
Then, calculate y for each x and print them out

Bisection



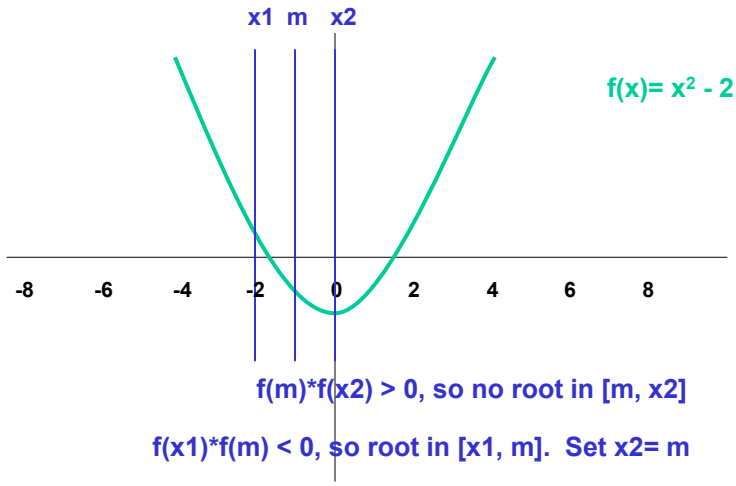
$f(x) = x^2 - 2$

f(x1)*f(m) > 0, so no root in [x1, m]

f(m)*f(x2) < 0, so root in [m, x2]. Set x1=m

Assume/analyze only a single root in the interval (e.g., [-4.0, 0.0])

Bisection



f(m)*f(x2) > 0, so no root in [m, x2]

f(x1)*f(m) < 0, so root in [x1, m]. Set x2= m

Continue until (x2-x1) is small enough