

# Lecture 4

## 1.00 – Fall 2005

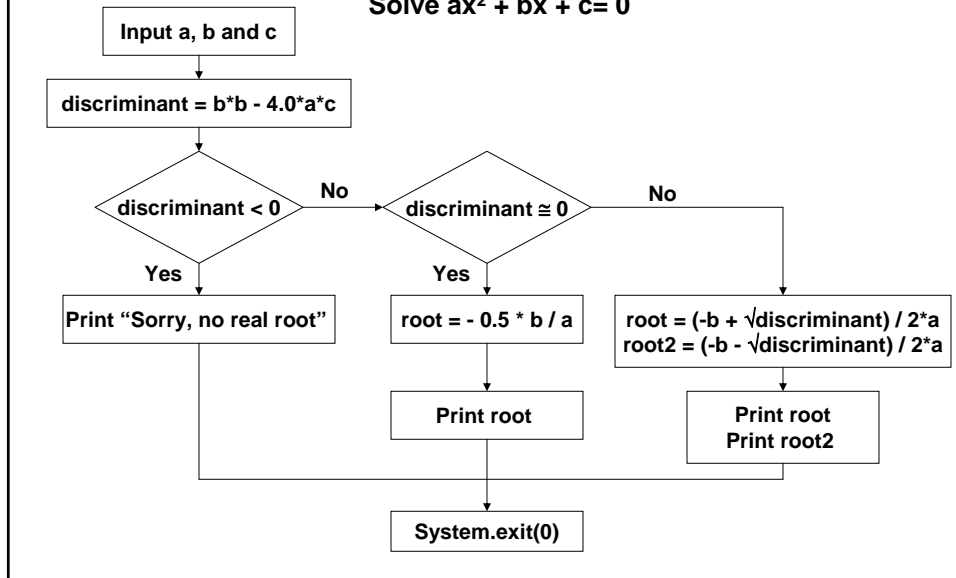
### More on Java Data Types and Control Structures

## Control Structures: Branch

General form	Example
<pre>if (boolean) statement;</pre>	<pre>if (x == y)   a = 20; if (x ==z) {   b = 10;   c = 20; }</pre>
<pre>if (boolean)   statement1; else   statement2;</pre>	<pre>if ( x == y ) {   a = 10;   b = 20; } else   x = y;</pre>
<pre>if (boolean1)   statement1; ... else if (booleanN)   statementN; else   statement;</pre>	<pre>if ( x &gt; 60)   y = 20; else if (x &lt; 30) {   z += y;   y = 25; } else   y= 40;</pre>

## Control example

Solve  $ax^2 + bx + c = 0$



## Control example

```
import javax.swing.*; // To support simple input
public class Control { // Quadratic formula
    public static void main(String[] args) {
        final double TOL= 1E-15; // Constant(use 'final')
        String input;
        input= JOptionPane.showInputDialog("Enter a");
        double a= Double.parseDouble(input);
        input= JOptionPane.showInputDialog("Enter b");
        double b= Double.parseDouble(input);
        input= JOptionPane.showInputDialog("Enter c");
        double c= Double.parseDouble(input);
```

## Control example

```
double discriminant= b*b - 4.0*a*c;
if ( discriminant < 0)
    System.out.println("Sorry, no real root");
else if (Math.abs(discriminant) <= TOL) {
    double root= -0.5 * b / a;
    System.out.println("Root is " + root); }
else { // Redefine 'root'; blocks have own scopes
    double root=(-b + Math.sqrt(discriminant))/
        (2.0*a);
    double root2=(-b- Math.sqrt(discriminant))/
        (2.0*a);
    System.out.println("Roots" + root + ", " +
        root2); }
System.exit(0); }
```

## Control example

- **The previous program has a deliberate, subtle bug**
  - Can you see it?
  - Is it likely that you'd find it by testing?
  - Is it likely you'd find it by using the debugger and reading the code?
- **Fix the error by rearranging the order of the if-else clauses**

## Control structure: Iteration

General form	Example
<pre>while (boolean)   statement;</pre>	<pre>while (x &gt; 0) {   System.out.println("x= " + x);   x--; }</pre>
<pre>do   statement; while (boolean); // Always executes stmt at least once</pre>	<pre>do {   System.out.println("x=" + x);   x--; } while (x &gt; 0)</pre>
<pre>for (start_expr; end_bool; cont_expr)   statement;</pre>	<pre>for ( x= 20; x &gt; 0; x--)   System.out.println("x=" + x);</pre>

## for loops

<pre>for (start_expr; end_bool; cont_expr)   statement;</pre>	<pre>for (j= 0; j &lt; 20; j++)   z += j;</pre>
<p><b>is equivalent to:</b></p>	
<pre>start_expr; while (end_bool) {   statement;   cont_expr; }</pre>	<pre>j= 0; while (j &lt; 20) {   z += j;   j++; }</pre>

## Example Method-Computing ln(x)

The natural logarithm of any number x  
is approximated by the formula

$$\ln(x) = (x-1) - (x-1)^2/2 + (x-1)^3/3 \\ - (x-1)^4/4 + (x-1)^5/5 + \dots$$

## Iteration Example: ln (x)

```
import javax.swing.*;
public class Iteration {
    public static void main(String[] args) {
        String input= JOptionPane.showInputDialog("Enter x (0-2)");
        double x= Double.parseDouble(input);
        // Compute 20 terms of
        // ln x= (x-1) - (x-1)^2/2 + (x-1)^3/3 - ...
        final int ITERATIONS= 20;          // Fixed no of iterations
        double logx= 0.0;
        double x1= x-1;
        for (int i= 1; i <= ITERATIONS; i++)
            if (i % 2 == 0)                // i even
                logx -= Math.pow(x1, i)/i;
            else
                logx += Math.pow(x1, i)/i;
        System.out.println("Ln x= " + logx);    } }
```

## import directive

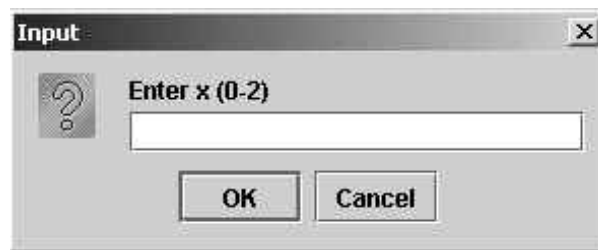
```
import javax.swing.*;
```

- Tells Java to import all classes (\*) in a package named javax.swing
- Not an executable statement. That's why it's outside any class
- Java starts search at a "known" locations (folders) configured in Eclipse or in a CLASSPATH environmental variable

## Dialog Box

```
String input = JOptionPane.showInputDialog(  
    "Enter x (0-2)");
```

This line of code pops up a dialog box with space to enter text. The String argument appears in the box.



## Conversion of String to double

```
double x= Double.parseDouble(input);
```

Double is the name of a class that is a non-primitive form of double type.

Double has a method called `parseDouble( )` that takes a String as argument

`parseDouble( )` returns a double value that is corresponds to the String argument

## Math class and pow( ) method

```
Math.pow(x1, i)/i;
```

- Math is a class that comes with Java system and has range of numerical functions
- `pow( )` is a method in Math class that takes 2 doubles as arguments
- `pow( )` computes the first argument to the power of the second argument
- Second argument is an int in this example, but is promoted to a double automatically by Java compiler
- Division is a double/int, but int in denominator is promoted before division, so result is a double

## Iteration Example 2: Ln (x) (part)

```
// Compute 20 terms of
// Ln x= (x-1) - (x-1)^2/2 + (x-1)^3/3 - ...
final int ITERATIONS= 20; // Fixed no of iterations
double logx= 0.0;
double x1= x-1;
for (int i= 1; i <= ITERATIONS; i++)
    if (i % 2 == 0) // i even
        logx -= Math.pow(x1, i)/i;
    else
        logx += Math.pow(x1, i)/i;
System.out.println("Ln x= " + logx);
}
```

## Java Arithmetic Operators

Table in precedence order, highest precedence at top

Operators	Meaning	Associativity
++ -- + (unary) - (unary)	increment decrement unary + ( x = +a) unary - ( x = -a)	Right to left
* / %	multiplication division modulo	Left to right
+ -	addition subtraction	Left to right

## Precedence, Association

- Operator precedence is in the order of the previous table

- Operators in same row have equal precedence

```
int i=5, j= 7, k= 9, m=11, n;  
n= i + j * k - m;           // n= ?
```

- Associativity determines order in which operators of equal precedence are applied

```
int i=5, j= 7, k= 9, m=11, n;  
n= i + j * k / m - k;       // n= ?
```

- Parentheses override order of precedence

```
int i=5, j= 7, k= 9, m=11, n;  
n= (i + j) * (k - m)/k;     // n= ?
```

## Precedence, Association, p.2

- Operator precedence is in the order of the previous table

- Operators in same row have equal precedence

```
int i=5, j= 7, k= 9, m=11, n;  
n= i + j * k - m;           // n= 57
```

- Associativity determines order in which operators of equal precedence are applied

```
int i=5, j= 7, k= 9, m=11, n;  
n= i + j * k / m - k;       // n= 1
```

- Parentheses override order of precedence

```
int i=5, j= 7, k= 9, m=11, n;  
n= (i + j) * (k - m)/k;     // n= -2
```

## Operator Exercises

- What is the value of int n:
  - n= 1 + 2 - 3 / 4 \* 5 % 6;
  - n= 6 + 5 - 4 / 3 \* 2 % 1;
  - i = 5; j = 7; k = 9;
  - n= 6 + 5 - ++i / 3 \* --j % k--;
  - i = 5;
  - n= i + ++i;

## Operator Exercises

- What is the value of int n:
  - n= 1 + 2 - 3 / 4 \* 5 % 6; // n=3
  - n= 6 + 5 - 4 / 3 \* 2 % 1; // n=11
  - i = 5; j = 7; k = 9;
  - n= 6 + 5 - ++i / 3 \* --j % k--; // n=8
  - i = 5;
  - n= i + ++i; // n=11 in Java  
// n=12 in C++
  - // Don't ever do any of these!

## Mixed Arithmetic

- **Promotion:**

- When two operands have different types, Java converts the 'low capacity' to the 'high capacity' type

```
int i;           // Max 2147483647
short j;        // Max 32767
i = j;          // Ok
```

- **Casting:**

- When you wish to convert a 'high capacity' type to a 'low capacity' type, you must indicate that explicitly

```
int i; short j;
j = i;           // Illegal -won't compile
j = (short) i;  // Casts i to short int
```

- **Binary operators (+, -, \*, /):**

- If either operand is (double, float, long), other will be converted to (double, float, long)
- Otherwise both are converted to int

## Mixed Arithmetic Example

```
public class TypePromotion {
    public static void main(String[] args) {
        byte b= 5, bval;
        short s= 1000, sval;
        int i= 85323, ival;
        long l= 999999999999L, lval;
        float f= 35.7F, fval;
        double d= 9E40, dval;

        bval = b + s;           // Won't compile
        bval = (byte) (b + s);  // Ok-cast, overflows
        sval = (short) (b + s); // Cast required!
        fval = b + s + i + l + f; // Ok-float
        lval = b + s + l + l;    // Ok-long
        ival = (int) f;         // Ok-truncates
    }
}
```

## Integer Arithmetic Properties

- **Overflows occur from:**
  - **Division by zero, including 0/0 (undefined)**
    - Programmer has responsibility to check and prevent this
    - Java will warn you (by throwing an exception) if it can't do an integer arithmetic operation (discussed later)
  - **Accumulating results that exceed the capacity of the integer type being used**
    - Programmer has responsibility to check and prevent, as in zero divides
    - No warning is given by Java in this case

## Integer Overflow Example

```
public class IntOverflow {  
    public static void main(String[] args) {  
        int biGval = 2000000000;  
        System.out.println("biGval : " + biGval);  
        biGval += biGval;  
        System.out.println("biGval : " + biGval); } }
```

```
// Output  
biGval : 2000000000  
biGval : -294967296
```

**It's necessary to analyze the range of your results, under worst case circumstances. You often use a long to hold sums of ints, etc.**

## Floating Point Properties

- **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(float f)`
    - Returns boolean which is true if argument is NaN

## Floating Point Properties

- **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)
- **Rounding errors– see following examples**

## 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");
    }
}
```

## Float Rounding Program

```
public class Rounding {
    public static void main(String[] args) {
        System.out.println("Number times inverse != 1");
        for (int test=2; test < 100; test++) {
            float top= test;
            float bottom= 1.0F/test;
            if (top*bottom != 1.0F)
                System.out.println(test + " " +
top*bottom);
        }
    }
}
```

## Float Rounding Program- Output

41	0.99999994	
47	0.99999994	<b>Occurs with doubles too; it's virtually the same, because 1.0 is more precise as a double</b>
55	0.99999994	
61	0.99999994	
82	0.99999994	
83	0.99999994	
94	0.99999994	
97	0.99999994	

## Doubles as Bad Loop Counters

```
public class Counter {
    public static void main(String[] args) {
        int i = 0;
        double c = 0.0;
        while (c != 10.0 && i < 52) {
            c += 0.2;
            i++;
            if (i % 10 == 0 || i >= 50)
                System.out.println("c: " + c + " i: " + i);
        }
    }
}
```

## Doubles as Bad Loop Counters

//Output

```
c: 1. 9999999999999998 i: 10
c: 4. 0000000000000001 i: 20
c: 6. 0000000000000003 i: 30
c: 8. 0000000000000004 i: 40
c: 9. 9999999999999996 i: 50
c: 10. 1999999999999996 i: 51
c: 10. 3999999999999995 i: 52
```

Notice accumulating, increasing error. Never use floats or doubles as loop counters

## Numerical Problems

Problem	Integer	Float, double
Zero divide (overflow)	Exception thrown. Program crashes unless caught.	POSITIVE_INFINITY, NEGATIVE_INFINITY
0/0	Exception thrown. Program crashes unless caught	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**
  - **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++

## switch statement-Example

```
int speed;
switch (speed/10) { // Limit= 9 mph (bicycle)
  case 3:
  case 2:
    System.out.println("Arrest"); // Drop thru
  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

## Terminating Iteration: break

```
int low= 8;
int high= 12;
for (i=low; i < high; i++) {
    System.out.println("i= " + i);
    if (i >= 9) {
        System.out.println("Too high");
        break;
    }
}
System.out.println("Next statement");

// Output is
i = 8
i = 9
Too high
Next statement
```

## Terminating Iteration: continue

- **Continue statement jumps to end of loop but continues looping**

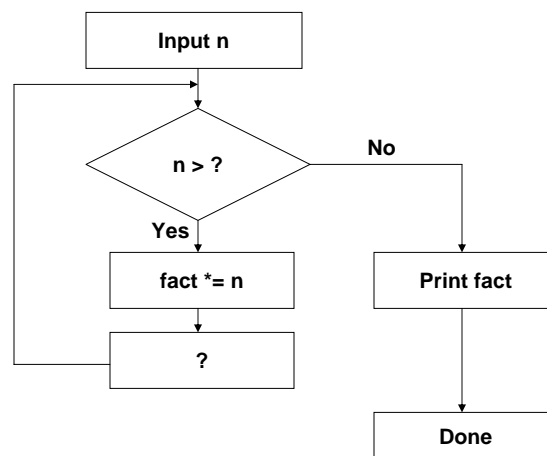
```
int sum= 0;
int low= -1;
int high= 2;
for (i=low; i < high; i++) {
    if (i == 0) {
        System.out.println("0 divide");
        continue; }
    int q= 1/i;
    sum += q;
}
System.out.println("Sum= " + sum);    // Sum = 0
```

## Iteration Exercise

- Recall the definition of a factorial:  
$$n! = n * (n-1) * (n-2) * \dots * 1$$
  
For example:  $4! = 4 * 3 * 2 * 1 = 24$
- A factorial has the following properties:
  - $0! = 1$
  - $n$  is a positive integer
- Write a main() that calculates the value of  $n!$  for a given  $n$ . Use, for example:

```
int n= 6;           // Assume n >=0; don't check
```

## Calculating the result



## **Sentinel Controlled Loop (Optional)**

**Suppose that the user doesn't want the program to run just once. Instead he wants to be prompted again to enter a number and, when done, prompted to enter another number.**

**In order to do that, a sentinel controlled loop will be used.**

**The idea of a sentinel controlled loop is that there is a special value (the "sentinel") that is used to say when the loop is done.**

**In this example, the user will enter "-1" to tell the program to end.**

**Assume the user enters the number via a JOptionPane. If you're writing the code by hand, don't worry about the exact syntax of JOptionPane; just assume the user enters a valid number.**

**Revise your program.**