# Interactive Applications (CLI) and Math

- Interactive Applications
- Command Line Interfaces
- The Math Class
- Example: Solving Quadratic Equations
- Example: Factoring the Solution
- Reading for this class: L&L, 3.5

# Interactive Applications (CLI)

- An interactive program with a command line interface contains a sequence of steps to:
  - Prompt the user to enter input data
  - Read and save the user's responses
  - Process the data after all input(s) are received
- We can prompt the user:

```
System.out.println("prompt text");
```

- We can read and format user responses:

```
type variable = scan.nextType();
```

# Interactive Applications (CLI)

- Similar to Quadratic.java (Page 129)

```
int a, b, c; // integer coefficients
Scanner scan = new Scanner(System.in);

System.out.println(“Enter coefficient A”);
a = scan.nextInt();
System.out.println(“Enter coefficient B”);
b = scan.nextInt();
System.out.println(“Enter coefficient C”);
c = scan.nextInt();
// we have the data to solve the equation
// ax-squared + bx + c = 0 for it’s roots
```

# We have the input values, now what?

- To solve the quadratic equation, we need to program in Java the formulas learned in high school algebra:

  discriminant = b squared – 4ac

  root1 = (-b + squareroot of discriminant)/2a

  root2 = (-b  - squareroot of discriminant)/2a
- How do we program those equations?
- We need to use the Math Class Library, Expression Evaluation, and Assignment

# The Math Class

- The `Math` class is part of the `java.lang` package
- The `Math` class contains methods that perform various mathematical functions
- These include:
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

# The Math Class

- The methods of the `Math` class are *static methods* (also called *class methods*)
- Static methods can be invoked through the class name – no object of the `Math` class is needed

```
value = Math.cos(90) + Math.sqrt(delta);
```

- Similar to Quadratic.java (page 130)

```
discriminant = Math.pow(b, 2) – 4.0 * a * c;
root1 = (-1.0 * b + Math.sqrt(discriminant))/(2.0 * a);
root2 = (-1.0 * b – Math.sqrt(discriminant))/(2.0 * a);
```

- Note: We can't program the ± in the formula on page 128 in Java. We need to calculate each root separately

# Solving Quadratic Equations

- However, the textbook's program to solve for the roots of a quadratic equation is deficient!
- The equations for calculating the roots are correct but are not used correctly in the program
- Since the user can enter any combination of three integer values for the coefficients, we need to analyze the possible special cases where just computing the formula based on the input values of "a", "b", and "c" is not correct
- This is the introduction to your Project #1

# Solving Quadratic Equations

- User can enter any values for “a”, “b”, and “c”
- If the user enters values that cannot be computed properly using the formulas, the program will fail to operate correctly
- Let’s try a = 1, b = 0, and c = 1
- The program generates two answers
  - NaN stands for “Not a Number”
- What happened?

# Solving Quadratic Equations

- With those coefficient values, the formula for calculating the discriminant results in a negative number

```
discriminant = b * b – 4 * a * c
discriminant = 0 * 0 – 4 * 1 * 1
discriminant = -4
```

- Later in calculating the roots, the formula takes the square root of the discriminant
- Mathematically, a negative number does not have a “real” square root

# Solving Quadratic Equations

- The Math.sqrt() method can't provide any "real" number that is the square root of -4
- In this case, it returns the result "NaN"
- However, in algebra we learned to "fake" the square root of a negative number by using the "imaginary" number i (the square root of -1)

  `Math.sqrt(-4)` can be shown as:

  `Math.sqrt(-1 * 4)` which equals:

  `Math.sqrt(-1) * Math.sqrt(4)` which equals:

  `i * 2.0` where i is the "imaginary" square root of -1
- How can we get our program to print this answer?

# Solving Quadratic Equations

- We need to write the program so that our code checks the value of the discriminant before trying to take the square root of it
- If the value of the discriminant is negative, we need to construct the correct answer

  `= “i * ” + Math.sqrt(Math.abs(-4));`
- That code will provide the resulting String

  `= “i * 2.0”`

# Solving Quadratic Equations

- There are other possible values of “a”, “b”, and “c” that can result in NaN or no valid result
- Suppose the user enters a value of 0 for “a”?
- The formula for the roots divides by (2.0 * a)
- If the value of “a” is 0, the division is impossible
- The expression evaluations will provide the results NaN or –Infinity
- Again, we need to write the program so that our code checks the value of “a” before trying to do the division

# Solving Quadratic Equations

- If the value of “a” is 0, is there a solution?
- Yes, let’s look at the equation with “a” = 0

  `0 * Math.pow(x, 2) + b * x + c = 0` is the same as:

  `b * x + c = 0` which can be solved as a linear equation:

  `x = - c / b` as long as `b` is not equal to 0!

  Note: There is now only one root - not two
- Based on the above, we can see another special case, if “b” is equal to 0 (but only if “a” is also equal to 0)
- It is OK for “b” to be equal to 0, if “a” is not 0

# Solving Quadratic Equations

- Suppose both “a” = 0 and “b” = 0?
- The remaining equation looks like this:

  0 + 0 + c = 0
- If the user entered a value of 0 for “c”, then any value of x is a solution, i.e. 0 + 0 + 0 always = 0
- But, suppose the user had entered a value for “c” that was not equal to 0?
- Now, there is no possible solution for x
- No value of x can make a non-zero value of “c” be equal to 0

# Solving Quadratic Equations

- Now that we have covered all these cases, what does it mean for our programming of a program for solving quadratic equations based on values for “a”, “b”, and “c”?
- We need to write the program so that our code makes decisions about each of these possible special cases before just trying to calculate a result based on the formulas

# Control Flow

- Up until now, all of our programs just ran sequentially through a sequence of steps
- Each statement did something and then continued to the next statement in sequence
- To make decisions while solving a quadratic equation, we need to control the flow of the execution of statements in our program
- We will see how to do that in the next lecture

# Factoring our Program

- But before we do that, let's see how to divide our program into smaller parts
- This is called *factoring* the program
- If we think about it, we can envision two different things that our program has to do
  - Gather input from the user and display results
  - Calculate the results based on the formulas
- At a top level, we can create one class for each of those two parts of the problem

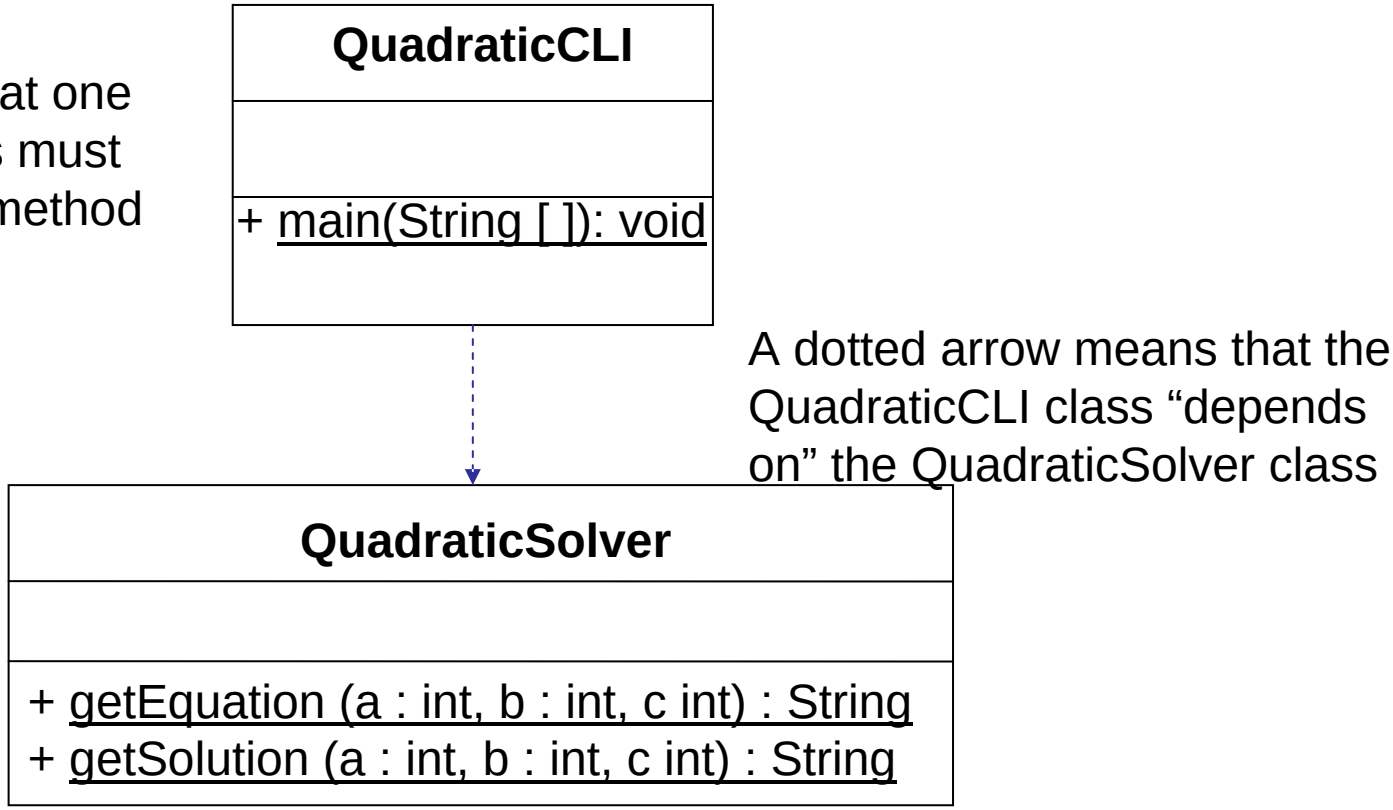
# Factoring our Program

- Why would we want to break our program down into two parts or classes?
- There are many possible reasons, two are:
  - We may assign two programmers to the job Each programmer can write one of the classes
  - We may be able to reuse one part separately from the other part, e.g. use the calculation class with a CLI class initially and re-use it later with a GUI class to make it more “user friendly”

# Factoring our Program

- Proposed “class diagram” for our program:

Remember that one of our classes must have a main method



# Factoring our Program

- Note that the method names in both classes are underlined in the class diagram
- That means that they are specified to be static just like the Math class methods
- We call the QuadraticSolver methods with the class name and pass the specified parameters to the method by listing them inside the ( )
- Each method returns a String to be printed

# The QuadraticCLI Class

- Passing the inputs to the QuadraticSolver class

```
Scanner scan = new Scanner(System.in);

System.out.println(“Enter coefficient A”);
a = scan.nextInt();
System.out.println(“Enter coefficient B”);
b = scan.nextInt();
System.out.println(“Enter coefficient C”);
c = scan.nextInt();

// we have the data to display and solve the equation
// pass a, b, and c as parameters in the method calls
System.out.println(QuadraticSolver.getEquation(a, b, c));
System.out.println(QuadraticSolver.getSolution(a, b, c));
```

# The QuadraticSolver Class

- A method to display a quadratic equation with coefficients a, b, and c as a String

```
public static String getEquation(int a, int b, int c)
 {
   String result = null;

   // code to concatenate string for returning equation text
   result = "Solving: " + a + "x\u00b2 " + ( (b >= 0) ? "+ " : "")
            + b + "x " + ( (c >= 0) ? "+ " : "") + c + " = 0";

   return result;
 }
```

- **The above method uses string concatenation**
- **There is an embedded Unicode value \u00b2 to get the superscripted 2 to display the $x^2$ term**

# The QuadraticSolver Class

- A method to provide the solution as a String

```
public String getSolution(int a, int b, int c)
{
  String solution = null;

  // Use if-else to choose the correct formulas
  // using the provided parameters a, b, and c.
  // Use string concatenation to create a
  // solution String that can be returned
  // to the QuadraticCLI class.
  . . .   ß Your Java statements go here
  return solution;
}
```