Interactive Applications (CLI) and Math

- Interactive Applications
- Command Line Interfaces
- The Math and Random classes
- Example: Solving Quadratic Equations
- Reading for this class: L&L, 3.4, 3.5

How input works

- The data will enter as a “stream”
- It is broken up by spaces
- Data is read in according to the type (integer, decimal, string)
- 12 24.7 hello goodbye hey[press Enter]

scan.nextInt(); => gets the 12

scan.nextDouble(); => gets the 24.7

scan.next(); (x3) => gets “hello”, “goodbye”, “hey”

scan.nextLine(); => gets the rest (nothing!)

Commands must be compatible!

Interactive Applications (CLI)

- An interactive program with a command line interface contains a sequence of steps to:
  - Prompt the user
  - Get the user's responses
  - Process the data as input is received (or after)
- **`System.out.println("Enter text:");`**

```
int i = scan.nextInt();
String str = scan.next();
type variable = scan.nextType();
```

The `Math` Class

- The `Math` class is part of the `java.lang` package. It is like String (and unlike Scanner) in that we do not have to import it.
- The `Math` class contains methods that perform various mathematical functions
- These include:
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

http://www.tutorialspoint.com/java/lang/java_lang_math.htm

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. It is **like** an object behavior but **not** tied to a specific object.

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

int i = -654; System.out.println(Math.abs(i)); => 654

System.out.println(Math.pow(-65, 3)); ==> -274625.0

(Because the Math.pow method sends back a decimal)

Random int 1 to 10: (int)(Math.random()* 10) + 1

OR (int)(Math.ceil((Math.random()* 10))
```

Random

- The Random class is for creating random-number generators
- Is part of java.util, so you have to import it: import java.util.Random;
- Gives methods such as:
  - nextFloat(): 0.0 <= x < 1.0
  - nextInt(): any possible int value
  - nextInt(int num): 0 <= x < num
  - More versatile than Math.random()

Interactive Applications (CLI)

- Consider Quadratic.java

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

System.out.println("Enter coeff. A");
a = scan.nextInt();
System.out.println("Enter coeff. B");
b = scan.nextInt();
System.out.println("Enter coeff. C");
c = scan.nextInt();
```

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

  roots = (-b ± squareroot of discriminant)/2a

- How do we program those equations?
- We need to use
  - The **`Math`** Class Library,
  - Expression Evaluation, and
  - Assignment

Solving Quadratic Equations

```
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);
```

- 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
- It only gives correct answers so long as the coefficients entered actually belong to a **quadratic** equation with **real** roots

Solving Quadratic Equations

- User can enter any values for “a”, “b”, and “c”, which can create special cases that the formula cannot accommodate
- Let’s try a = 2, b = 3, and c = 4 (demo)
- What happened?
- **Answer**: A negative discriminant, which has no real square root

```
discriminant = 3 * 3 - 4 * 2 * 4
discriminant =  9 - 32
discriminant = -23
```

The Math.sqrt method cannot handle this!

Solving Quadratic Equations

- However, there is **the "imaginary" number i** (the square root of -1)

In math: $\sqrt{-7} \Rightarrow i * \sqrt{7}$

String: "i * " + Math.sqrt(7); => "i * 2.6457513110645907"

Equation may have **complex** roots (e.g., `5 + i√7` and `5 – i√7`)

- How do we accommodate such user input?
- **Answer:** check discriminant value:
  - *Positive*: Use given formula
  - *Negative*: Construct complex root strings
  - *Zero*: -b/2a (Need not print value twice!)

Solving Quadratic Equations

- Other possible problems:
  - *a = 0 (but not b)*: Formula divides by 2 * a, leading to an error if a equals 0. (Equation is **linear, not quadratic**, so the only root is the **y-intercept**)
  - *a and b (but not c) are 0*: A horizontal line that **never touches the x-axis**, so no roots
  - *All three are 0*: The x-axis itself, so all values are roots (in the sense that **any** value of x would satisfy `0*x^2 + 0*x + 0 = 0`
- Our program must account for all these possibilities – by **making decisions!**

Control Flow

- Up until now, each program has been a linear sequence of steps
- First statement, second, and so forth…in sequence
- To make decisions while solving a quadratic equation, we need to direct the program to different statements based upon contingencies of user input
- We will see how to do that shortly