# Getting started with Java

# Magic Lines

```
public class MagicLines {

        public static void main(String[] args) {









        }

}
```

# Comments

- Comments are lines in your code that get ignored during execution.
- Good for leaving explanations of your code
  - For other programmers
  - For yourself  in 5 months
- Good for suppressing one snippet of code when you want to try an alternative snippet
- Indicated by
  - `//your comment here` (single line)
  - `/* your comment here */` (multiple lines)

# Comments

```
public class MagicLines {

        public static void main(String[] args) {

                // Comment: Execution begins here

                /* Comment: Continues here */

                /* Comment:
                 * Ends
                 * here
                 */
        }

}
```

# Text output

- `System.out.print`(*your_output_here*)
  - Prints your output in the console
- `System.out.println`(*your_output_here*)
  - Subsequent output will be on the next line
- `System.out.format`(*your_output_here*)
  - Apply fancy formatting to output before printing
  - Like printf from C
  - (You don't need to know this one)

# Text output

```
public class MagicLines {

        public static void main(String[] args) {

                System.out.println("Start here");

                System.out.println("Continue");

                System.out.println("Stop "+"here");

        }

}
```

# Strings of Text

- A sequence of letters is called a *String*:
  - “Start here”
  - “Continue”
  - “Stop ”
  - “here”
- Strings can be concatenated with +:
  - “Stop “ + ”here” is the same as “Stop here”
  - Numbers can be concatenated too. These are the same:
    - “Stop “ + 3
    - “Stop 3”

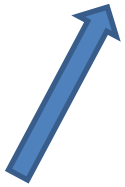
# Strings of Text

- Punctuation and other characters allowed
  - "\tStart here!" (\t results in a tab)
- Strings are a *type of data*.
- Your output in `System.out.print()` and `println()` can be other types (like numbers), not just Strings:
  - `System.out.print(3);`

# Managing Data

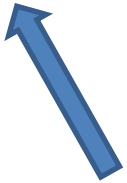
Save the data “Hello”, which is a string, in a variable named str:

```
String str = “Hello”;
```



Data type

Variable



Literal

# Data types

- “Primitive” types
  - Basic data like numbers or letters
  - Require just a few contiguous bytes
- Objects
  - More complicated data like Strings of letters or arbitrarily precise numbers (BigDecimals)
  - Require many bytes of storage
  - Defined with many more lines of code
  - More complex behavior

# “Primitive” Data-types

- Logical
  - `boolean` : Two values, true or false
- Textual
  - `char` : Single character (`‘a’`, `‘b’`, …)
    - 16 bits (65536 possible characters)

# Primitive Data-types

- Integral
  - `byte` : 8-bit integer in [-128, 127]
  - `short` : 16-bit integer in [-32768, 32767]
  - `int` : 32-bit integer in
    [-2,147,483,648, 2,147,483,647]
  - `long` : 64-bit integer in
    [-9,223,372,036,854,775,808,
    9,223,372,036,854,775,807 ]

# Representing negative numbers

| Binary | Decimal (Unsigned) | Decimal (Signed) |
|---|---|---|
| 000 | 0 | ? |
| 001 | 1 | ? |
| 010 | 2 | ? |
| 011 | 3 | ? |
| 100 | 4 | ? |
| 101 | 5 | ? |
| 110 | 6 | ? |
| 111 | 7 | ? |

# Signed two's complement

| Binary | Decimal (Unsigned) | Decimal (Signed) |
|---|---|---|
| 000 | 0 | 0 |
| 001 | 1 | 1 |
| 010 | 2 | 2 |
| 011 | 3 | 3 |
| 100 | 4 | -4 |
| 101 | 5 | -3 |
| 110 | 6 | -2 |
| 111 | 7 | -1 |

- Leftmost digit indicates sign
- Splits in the middle
- "Wraps around" at the ends

# Primitive Data-types

- Floating point
  - `float` : 32-bit rational numbers
    - Ranges from $\sim \pm 10^{-45}$ to $\sim \pm 10^{38}$
  - `double` : 64-bit rational numbers
    - Ranges from $\sim \pm 10^{-324}$ to $\sim \pm 10^{308}$
- How to represent in binary?

# Primitive Data-types

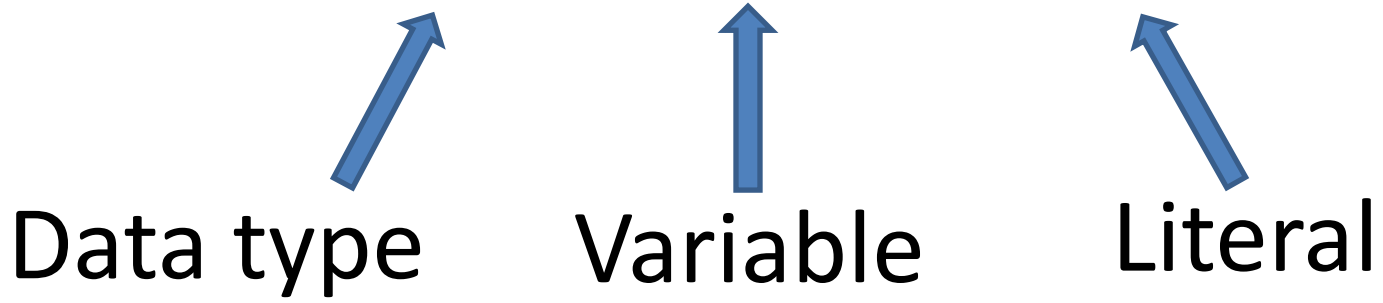
- Floating point
  - `float` : 32-bit rational numbers
    - Ranges from $\sim \pm 10^{-45}$ to $\sim \pm 10^{38}$
  - `double` : 64-bit rational numbers
    - Ranges from $\sim \pm 10^{-324}$ to $\sim \pm 10^{308}$
- Uses the IEEE 754 floating point standard
- Think scientific notation in base 2:

$1.2 \times 10^{3}$ $1.01101 \times 2^{10010100}$

# Literals

Literals are constant values that are “hard-coded” into the program:

```
int x = 123;
```



# Literals

- `boolean` (case sensitive):
  - `true`
  - `false`
- `char`
  - Single quote, single letter: `'a'`
  - Single quote, single escape sequence.
    - Backspace: `'\b'`
    - Tab: `'\t'`
  - Single quote, Unicode escape sequence:
    - `'\u00F1'` (n with a tilde)
  - 16-bit *positive* integer in [0, 65535]

# Literals

- `String`
  - Double quote, multiple letters and escape sequences:
    - “Hello”
    - “\t”
    - “\u00F1”
    - “Hello\t\u00F1”

# Literals

- Integral: `byte, short, int`
  - Base 10: `123`
  - Base 2 (prefix with `0b`): `0b01111011`
  - Base 16 (prefix with `0x`): `0x7B`
  - Underscores allowed between digits: `999_999_999`
- `long:`
  - suffix with `L` (or `l`): 2_147_483_648L

# Literals

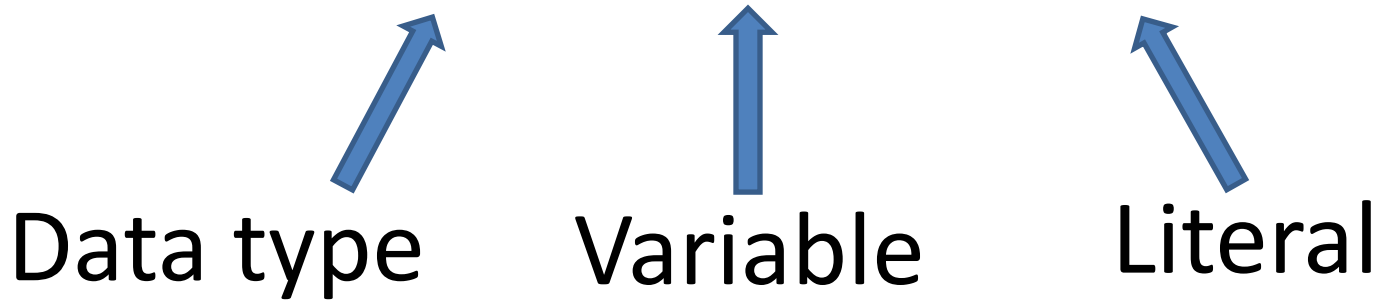
- Floating point (`float`)
  - Suffix with `F` (or `f`): `123.4f`
- Floating point (`double`):
  - With decimal point: `123.4`
  - Optional suffix `D` (or `d`): `123.4d`
  - Scientific notation using `E` (or `e`):

$1.234 \times 10^{20}$  `1.234e20`

# Variables

A name that denotes some value. The value of a variable can change over the course of program execution.

```
int x = 123;
```



# Declaring variables

- All variables must be *declared*, before they can refer to a value. The declaration determines the data type.
- The value of a variable can change, but the data-type cannot.

```
int x;
float y;
String z;
```

# Assigning to variables

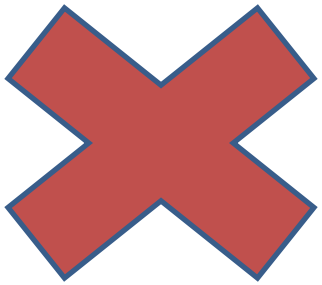
- To reset the value of a variable, a new value must be *assigned*.
- Assignment is done using the “assignment operator” (equal sign):

```
x = 3;
y = 3.0;
z = “3.0”;
```

# Assigning to variables

- Assignment is not the same as logical equality!!!!

```
int x; //Declare x
x = 3; //The value of x is 3…
x = 4; //It is also 4???
```



# Assigning to variables

- Assignment is not the same as logical equality!!!!

```
1 int x; //Now x is declared
2 x = 3; //Now the value of x is 3
3 x = 4; //Now the value of x is 4
```

# Initializing variables

- You can declare a variable and assign a value to it in the same line. This is called *initialization*:

```
int x = 3;
float y = 3.0;
String z = "3.0";
```

# Type casting

- Converting data from one type to another is called “type casting”.
- Syntax: precede your value with the new type (in parentheses).

```
float y;
y = (float) 3; //Now y is 3.0
```

# Type casting

- Type casting can destroy data:
  - `int i = (int) 3.3;`
    - Removes fractional part
      - (3.3 becomes 3)
  - `byte b = (byte) 0b101_1000_0001;`
    - Removes most significant bytes
      - `0b101_1000_0001` becomes `0b1000_0001`
- Only necessary when data might be destroyed

```
float f = 3; //No error
int i = 3.0; //Error
```

# Lab tomorrow

- Bring your laptops!
- You can get started early by visiting the UMCP CS Department Eclipse tutorial:
  http://www.cs.umd.edu/eclipse/
- (This is also linked on the resources page of the class website)
- Follow the steps under the “Installing Eclipse” section