• Hardware
  – CPU runs a program’s statements one at a time
    • Starts from the “Entry Point”
    • Left to Right then Top to Bottom
  – Memory stores information that can be accessed and modified by the CPU

• Java Programs
  – Organized by Projects, then Classes, then Methods
    • Body of something is in between curly braces “{}”
  – The main method is Java’s “Entry Point”
    • Code should be written in the body of the Main Method for now

```
1 //A simple Java Program written by JJ Shepherd
2 //Also these are comments, and are ignored by the compiler
3 public class SimpleJavaProgram
4 {
5   //Body of the Class
6   //Main method or entry point
7   public static void main(String[] args)
8   {
9     //Body of the Main Method
10    //Here is where we are going to be writing most code for now
11
12 }
13
14 }
```
Variables

- Variables store data such as number or characters
  - Containers or Boxes
  - Implemented using Memory

```java
int numberOfCats = 1;
```
Value is the name we called the stored data
  – Values are stored in a memory location
• Its value can be changed

```
int numberOfCats = 2;
```
We must *declare* variables before using them.

- Spoken: “I need a container of this size called this name”

- Declaring a variable requires
  - Type
  - Identifier (name)

### Declaring Syntax

```
<<type>> <<identifier>>;
```

### Example

```
int numberOfCats;
```

- Type
- Identifier
Types
- Type corresponds to the type of data and the number of bytes in memory
- Programming Languages may be
  - Strongly Typed
  - Loosely Typed
- Only use the Type when *Declaring*
- 2 Major Types
  - Class (Object)
  - Primitive

- **Primitive types** are used for simple values such as a number or single character
- **Class Types** are used for a *class* of objects and combine both data and methods (functionality)
  - Reference
  - Contents
Primitive Types

- Integer (Whole Number) Types
  - byte
  - short
  - int (Most Common)
  - long
- Floating-point (Decimal) Types
  - float
  - double (Most Common)
- Character Type
  - char
- Boolean Type
  - boolean

Primitive Types

- Integer (Whole Number) Values
  - Examples: 0 -1 365 12000
- Floating-point (Decimal) Types
  - Include the Decimal Point
  - Examples: 0.99 -22.8 3.14159
- Character Type
  - Single Quotes NOT Double Quotes
  - Examples: ‘a’ ‘A’ ‘#’ ‘ ‘
- Boolean Type
  - Only 2 values
  - Examples: true false
### Primitive Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>Stores whole numbers from -128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>Stores whole numbers from -32,768 to 32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>Stores whole numbers from (-2,147,483,648) to (2,147,483,647)</td>
</tr>
<tr>
<td>long</td>
<td>8 bytes</td>
<td>Stores whole numbers from (-9,223,372,036,854,775,808) to (9,223,372,036,854,775,807)</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>Stores fractional numbers. Sufficient for storing 15 decimal digits</td>
</tr>
<tr>
<td>boolean</td>
<td>1 bit</td>
<td>Stores true or false values</td>
</tr>
<tr>
<td>char</td>
<td>2 bytes</td>
<td>Stores a single character/letter or ASCII values</td>
</tr>
</tbody>
</table>
Identifiers
An **identifier** is a name, such as the name of a variable.

- Identifiers should be meaningful
- Identifiers may contain ONLY
  - Letters
  - Digits (0 through 9)
  - The underscore character ( `_ `)
  - And the dollar sign symbol ( `$` ) which has a special meaning

- Identifiers CANNOT contain
  - Spaces of any kind
  - Digit as the First Character
  - Dots ( `.` )
  - Asterisks ( `*` )
  - Other types of special characters

- Identifiers are Case Sensitive
  - “Stuff”, “stuff”, “STUFF”, and “sTuFF” would all be considered different identifiers

- Identifiers CANNOT be a **reserved word**
  - Example Reserved Words: `int`, `public`, `class`
### Naming Conventions

- **Class Types** start with an **Uppercase** character
  - Example: `String`
- **Primitive Types** start with a **Lowercase** character
  - Example: `int`
- **Variables identifiers of both** start with a **Lowercase Character**
- **Multiword identifiers** are “punctuated” using **uppercase characters**

<table>
<thead>
<tr>
<th>Good Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int test01;</code></td>
</tr>
<tr>
<td><code>double largeValues;</code></td>
</tr>
<tr>
<td><code>boolean inClass;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bad Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int 1Test;//Started with a digit</code></td>
</tr>
<tr>
<td><code>double big vals;//Used a space</code></td>
</tr>
<tr>
<td><code>boolean class;//Class is a reserved word</code></td>
</tr>
</tbody>
</table>
Example
int i;
double j;
char o;

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example

```c
int i;
double j;
char o;
```
Example

```c
int i;
double j;
char o;
```

### Memory

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### Declaring Variables

**Example**

```plaintext
int i;
double j;
char o;
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example

```java
int i;
double j;
char o;
```

### Memory

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>0.0</td>
<td>32</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example
int i;
double j;
char o;

Memory

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>0.0</td>
<td>32</td>
</tr>
</tbody>
</table>

...               | ...      | ...          |
```c
Example
int i;
double j;
char o;
```

### Memory

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>0.0</td>
<td>32</td>
</tr>
<tr>
<td>o</td>
<td>\u0000</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### Declaring Variables

**Example**

```c
int i;
double j;
char o;
```

**Memory**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>0.0</td>
<td>32</td>
</tr>
<tr>
<td>o</td>
<td>\u0000</td>
<td>40</td>
</tr>
<tr>
<td>???</td>
<td>???</td>
<td>42</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Assigning Values
The equals symbol "=" is the assignment operator.

Stores values found on the right hand side (RHS) of the operator into the identifier found on the left hand side (LHS).

Assignments are valid if the type matches are is at least compatible:
- Primitive types can be stored in other primitive types as long the type’s byte amount is less than or equal to value being stored.
- Otherwise “type casting” is required.
- Type casting does not round it cuts off everything past the decimal point ".".

Spoken:
- “Store this value in this container”

**Syntax**

```
<<identifier>> = <<value>>;
```

**Examples**

```
i = 0;
j = 22.3;
o = ‘h’;
i = (int)j; // Type cast from double to int
// Value stored in “i” is 22
```
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Memory
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example

```java
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example

```java
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>22.3</td>
<td>32</td>
</tr>
<tr>
<td>o</td>
<td>‘\u0000’</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;

Memory

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>22.3</td>
<td>32</td>
</tr>
<tr>
<td>o</td>
<td>‘h’</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;

Memory

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>22.3</td>
<td>32</td>
</tr>
<tr>
<td>o</td>
<td>‘h’</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>22.3</td>
<td>32</td>
</tr>
<tr>
<td>o</td>
<td>‘h’</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>22.3</td>
<td>32</td>
</tr>
<tr>
<td>o</td>
<td>‘h’</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
• Declare and assigning initial values
  – Good programming practice to assign initial values
  – Shortens two statements into one
  – Types are not still used after the declaration

Example
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Contents</th>
<th>Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td>j</td>
<td>22.3</td>
<td>32</td>
</tr>
<tr>
<td>o</td>
<td>‘h’</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
• Establishes a value that cannot change
• MUST assign a value initially
• Great for avoiding "magic numbers"
• Good programming practice
  – Make the scope public
  – Make it static
  – Capitalize all characters in the identifier

**Syntax**

```
public static final <<type>> <<identifier>> = <<value>>;
```

**Examples**

```
public static final double PI = 3.14159;
public static final int BOARD_SIZE = 10;
```
• Performs computation and then assigns the results
• Order of Operations
• Basic Math Operations
  – Addition “+”
  – Subtraction “-”
  – Multiplication “*”
  – Division “/”
• Mod Operator “%”
  – Returns the remainder after division
  – Ex: 15 % 2 = 1

Syntax

<<identifier>> = <<value>> <<operator>> <<value>>;

Examples

//Variables
int value = 64 % i + 32;

//Constants
public static final double PI = 3.14159;
public static final double PI_SQ = PI*PI;
**Math Operators**

- **Compute and Assign (C&A) Operators**
  - Shorthand for applying some operator and value to a variable
  - Same as:
    - `<<identifier>> = <<identifier>> <<operator>> <<value>>;`
    - Ex: `i = i+1; i+=1; i++;` //Same statements

- **Common Versions**
  - `+=` – add and assign
  - `-=` – subtract and assign
  - `*=` – multiply and assign
  - `/=` – divide and assign
  - `%=` – mod and assign

- **Special versions**
  - `++` – Increase by 1
    - Same as `+= 1`
  - `--` – Decrease by 1
    - Same as `-=1`

---

**Syntax**

```
<<identifier>> <<C&A operator>> <<value>>;
```

**Examples**

```
i  += 128;  //If i = 32 now it is 160
j  %= 2;   //If j = 28.0 now it is 0.0
```
• Integers are ALWAYS Integers
  – Anything past the decimal point is cut off
  – Also can be considered “rounding down” or “taking the floor”
  – Example: $1/3 = 0$
  – Logic Errors

• eNotation
  – Allows number to be written in scientific notation
  – Example: 865000000.0 can be written as 8.65e8

• Imprecision with Floating-Point Numbers
  – Floating point numbers are approximations as they are finite
  – Example: $1.0/3.0$ is slightly less than $1/3$ ergo $1.0/3.0 + 1.0/3.0 + 1.0/3.0 < 1.0$
  – Logic Errors
For now, input and output is done in the Console

- Command Line Interface
- Console Outputs (Writes)
  - Left to Right
  - Up to Down
- Console Inputs (Reads)
  - Left to Right
  - Up to Down

### Syntax

```
System.out.println(<value>);
```

### Examples

```java
int i = 22;
System.out.println(i);
```
• **System.out.println(<<argument>>);**
  - Statement used to output the argument and adds a new line after

• **System.out.print(<<argument>>);**
  - Statement used to output the argument but stays on the same line

• “Prints” to the standard system output (the console)
• Use Scanner to read from Console
• Must import type Scanner from "java.util" package
  – import java.util.Scanner;
• Create an instance of type Scanner that "scans" the standard system input
  – Scanner keyboard = new Scanner(System.in);
• Useful methods
  – next()
  – nextLine()
  – nextInt()
  – nextDouble()
• Also can be used to "scan" Strings, files, network traffic, etc.

Examples
Scanner keyboard = new Scanner(System.in);
String name = keyboard.nextLine();
int i = keyboard.nextInt();
keyboard.nextLine();//Useful "fix-up"
double j = keyboard.nextDouble();
keyboard.nextLine();//Useful "fix-up"
System.out.println(name + " " + i + " " + j);

Console
JJ
64
3.14
JJ 64 3.14