



1

Basic Computation Part 01

2

Procedural Programming

<ul style="list-style-type: none"> • Hardware <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 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 	<pre> 1 //A Simple Java Program written by JJ Shephard 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 </pre>
---	---

3

Problem Solving

<ol style="list-style-type: none"> 1. Identify your Data. 2. Determine how the data changes over time. 3. Consider structures for both behavior and data. 4. Group together (encapsulate) related information into Classes or Objects. 5. Develop functionality / methodologies that relates to the behavior of your Objects. 6. Further identify relationships between the Classes and optimize the structure. 7. Determine if there exists software patterns that may assist. 	
--	--

4

Problem Solving

<ol style="list-style-type: none"> 1. Identify your data. <ul style="list-style-type: none"> – What information do we need to keep in our memory? – What information is variable and what information is constant? – What is the type of data? Numeric? Words/Strings? Collections? Other Structures? 	
--	--

INPUT
(Data)

Output
(Information)

5


Variables

<ul style="list-style-type: none"> • Variables store data such as number or characters – Containers or Boxes – Implemented using Memory 	<p style="font-family: monospace; font-size: 1.2em; margin-top: 10px;">int numberOfCats = 1;</p>
--	--

6

Variables

- Value is the name we called the stored data
 - Values are stored in a memory location
- Its value can be changed



```
int numberOfCats = 2;
```

7

Variables

- 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

8

Types

9

Types

- Type corresponds to the type of data and the number of bytes in memory
- Programming Languages may be
 - Strongly Typed
 - Weakly (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

10

Types

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

11

Types

Primitive Types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII value

12

Identifiers

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

13

14

Identifiers

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

Good Examples

```
int test01;
double largeValues;
boolean inClass;
```

Bad Examples

```
int 1Test; // Started with a digit
double big vals; // Used a space
boolean class; // Class is a reserved word
```

Declaring Variables

Example

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

Memory

Identifier	Contents	Byte Address
...
		28
...

15

16

Declaring Variables

Example

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

Memory

Identifier	Contents	Byte Address
...
		28
...

Declaring Variables

Example

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

Memory

Identifier	Contents	Byte Address
...
i	0	28
...

17

18

Declaring Variables

Example

int i;
double j;
char o;

Memory

Identifier	Contents	Byte Address
...
i	0	28
...

19

Declaring Variables

Example

int i;
double j;
char o;

Memory

Identifier	Contents	Byte Address
...
i	0	28
j	0.0	32
...

20

Declaring Variables

Example

int i;
double j;
char o;

Memory

Identifier	Contents	Byte Address
...
i	0	28
j	0.0	32
...

21

Declaring Variables

Example

int i;
double j;
char o;

Memory

Identifier	Contents	Byte Address
...
i	0	28
j	0.0	32
o	'\u0000'	40
...

22

Declaring Variables

Example

int i;
double j;
char o;

Memory

Identifier	Contents	Byte Address
...
i	0	28
j	0.0	32
o	'\u0000'	40
???	???	42
...

23

Declaring Variables

Example

int i;
double j;
char o;

Memory Concept

Identifier	Contents	Byte Address
...
i	0	28
j	0.0	32
o	'\u0000'	40
Future identifier	...	42
...

Physical Memory Concept

0	28	32	36	40
00	00000000	00000000	00000000	00000000

24

Assigning Values

Assignment Operator

- 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 or is at least compatible
 - Primitive types can be stored in other primitive types as long as 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
```

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
...

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
...

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
...

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
...

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
...
...
...
...

31

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
j	0.0	32
...
...
...

32

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
j	22.3	32
...
...
...

33

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
j	22.3	32
o	'\u0000'	40
...
...
...

34

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
j	22.3	32
o	'h'	40
...
...
...

35

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
j	22.3	32
o	'h'	40
...
...
...

36

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	0	28
j	22.3	32
o	'h'	40
...

37

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	22	28
j	22.3	32
o	'h'	40
...

38

Assignment Operator

- 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		
Identifier	Contents	Byte Address
...
i	22	28
j	22.3	32
o	'h'	40
...

39

Constants

- 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;
```

40

Math Operators

- 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_50 = PI*PI;
```

41

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 += 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
```

42

More Math Notes

<ul style="list-style-type: none"> • eNotation <ul style="list-style-type: none"> – Allows number to be written in scientific notation – Example: 865000000.0 can be written as 8.65e8 • Imprecision with Floating-Point Numbers <ul style="list-style-type: none"> – 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 	<ul style="list-style-type: none"> • Integers are ALWAYS Integers <ul style="list-style-type: none"> – Anything past the decimal point is cut off – Also can be considered “rounding down” or “taking the floor” – Example: 1/3 = 0 – Logic Error
---	--

43

Basic Input and Output

<ul style="list-style-type: none"> • For now, input and output is done in the Console • Command Line Interface • Console Outputs (Writes) <ul style="list-style-type: none"> – Left to Right – Up to Down • Console Inputs (Reads) <ul style="list-style-type: none"> – Left to Right – Up to Down 	<p style="text-align: center;"><u>Syntax</u></p> <pre>System.out.println(<<value>>);</pre> <hr/> <p style="text-align: center;"><u>Examples</u></p> <pre>int i = 22; System.out.println(i);</pre>
---	---

44

Basic Output

<ul style="list-style-type: none"> • System.out.println(<<argument>>); <ul style="list-style-type: none"> – Statement used to output the argument and adds a new line after • System.out.print(<<argument>>); <ul style="list-style-type: none"> – Statement used to output the argument but stays on the same line • “Prints” to the standard system output (the console) 	<p style="text-align: center;"><u>Syntax</u></p> <pre>System.out.println(<<argument>>); System.out.print(<<argument>>);</pre> <hr/> <p style="text-align: center;"><u>Examples</u></p> <pre>int i = 22; System.out.println(i);</pre>
---	--

45

Basic Input

<ul style="list-style-type: none"> • Use Scanner to read from Console • Must import type Scanner from “java.util” package <ul style="list-style-type: none"> – import java.util.Scanner; • Create an instance of type Scanner that “scans” the standard system input <ul style="list-style-type: none"> – Scanner keyboard = new Scanner(System.in); • Useful methods <ul style="list-style-type: none"> – next() – nextLine() – nextInt() – nextDouble() • Also can be used to “scan” Strings, files, network traffic, etc. 	<p style="text-align: center;"><u>Examples</u></p> <pre>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);</pre> <hr/> <p style="text-align: center;"><u>Console</u></p> <pre>J> 64 3.14 J> 64 3.14</pre>
--	---

46