JAVA UNIT 1 REVIEW

Chapter1: Review

- What is Java?
- Object Oriented Programing
- Eclipse Environment

What is Java

- Java is a robust, object-oriented, platform independent, and secure programming language that is considered easier then C++ and less prone to programing mistakes then other C variants languages.

Object Oriented Programming

- **Object Oriented Programming (OOP)** is a software development methodology where the components of the software are conceptualized as a set of objects that work together to achieve a goal. These objects are created from a template, called a class, and they contain data and actions (statements) that are required to use the associated data within the class.

Eclipse Environment -Review

- Workspace
- Preferences
- Auto-Save
- Ctl-Space
- Marketplace

Chapter 2: Review

- Statements and Expressions
- Variables and Primitive Data Types
- Constants
- Comments
- Literals
- Math Statements
- Logical Operators

Statements and Expressions

- Statement: Simple command that causes something to happen
- Terminate with a semi-colon ;
- Statement blocks – things that must execute as a set grouped together with curly braces

```
{
….statements
 }
```

Variables and Primitive Data Types

- Variable: a container for the information we need
- Declare a variable: give it a name and type
- Initialize a variable: give the variable its initial value
- **Instance Variables:** define an objects attributes
- **Class Variables**: define the attributes of an entire class
- Local Variables: (right here right now) declared and used locally then destroyed

Primitive Data types

- A primitive type is predefined by the language and is named by a reserved keyword
  - Byte
  - Short
  - Int
  - Long
  - Float
  - Double
  - Boolean

Constants

- A variable that never changes
- Need to define a value that is shared
- Declared with the keyword final

Comments

- Describe how you program functions
- Remove code to test
- Compiler ignores them
- // single line comment
- /* */ block comment

- Javadoc begins with /** ends with */

Official documentation, can be read by utility programs to form web page records

Literals

- Literal is a number, character or combination that represent a value
- Non printable characters
- String literals

Escape Sequence	Character
\n	newline
\t	tab
\b	backspace
\f	form feed
\r	return
\"	" (double quote)
\'	' (single quote)
\\	\ (back slash)
\uDDDD	character from the Unicode character set (DDDD is four hex digits)

Comparison

op	meaning	true	false
==	equal	2 == 2	2 == 3
!=	not equal	3 != 2	2 != 2
<	less than	2 < 13	2 < 2
<=	less than or equal	2 <= 2	3 <= 2
>	greater than	13 > 2	2 > 13
>=	greater than or equal	3 >= 2	2 >= 3

Math Statements

Operator	Name	Example expression	Meaning
*	Multiplication	a * b	a times b
/	Division	a / b	a divided by b
%	Remainder (modulus)	a % b	the remainder after dividing a by b
+	Addition	a + b	a plus b
-	Subtraction	a - b	a minus b

Assignment Operators

=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR(exclusive or)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	NOT equal to
?:	Ternary if-then-else

Chapter 3 Review

- New Objects
- Calling Methods
- Object References
- Object Casting and Primitive Types
- Object Comparisons Values and Classes

New Objects

- To create an object we must use the *new()* operator
- Cannot leave the parenthesis off-even with no arguments to pass
- Can pass arguments to a class based on the classes Constructor
- The Constructor can be used to initialize a new object and its variables
- Get and set instance variables with dot notation
- ClassVaribles apply to every object in the class

Calling Methods

- Also use dot notation
  - object.method();
  - Customer.cancelOrder();
- Class Methods
  - Defined in and apply to the class as a whole
  - Used for utility programs

Object References

- Reference is an address that indicates where an objects variables and methods are stored
- RefTester.java
- pt2=pt1; creates a reference from pt2 to pt1

Object Casting and Primitive Types

- Java methods and constructors require a specific form and will not accept alternatives
- Must use the correct data types
- Casting: changing the value to the right type
  - Primitives smaller to larger, not Boolean
- Java.lang contains objects that correspond to primitive data types
- Integer – int Character-char Float-float

Object Comparisons Values & Classes

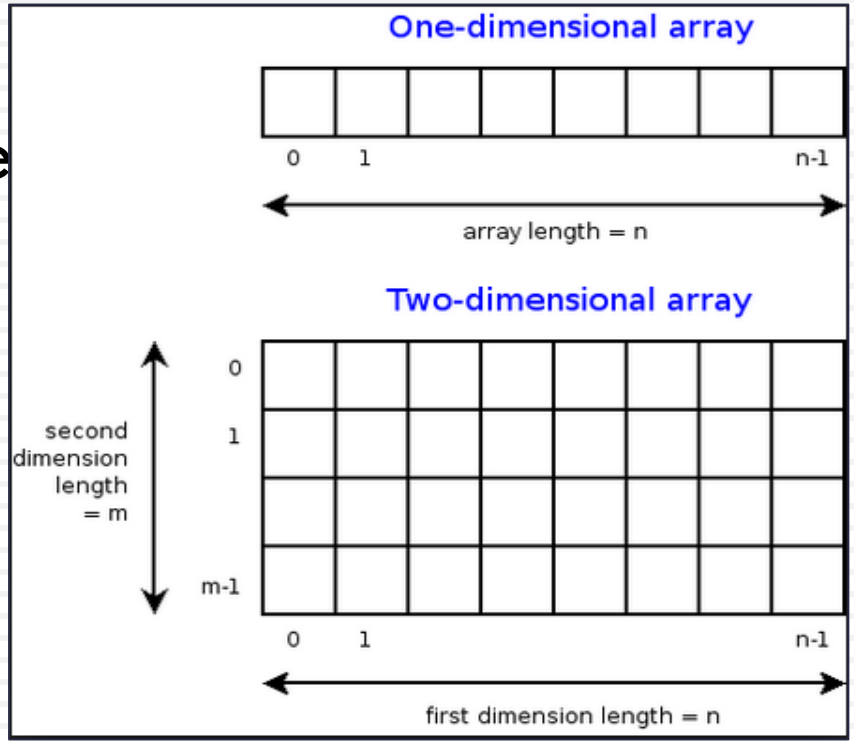
- == and =! work for and object but compare the objects to see if they are the same not the values of its instance variables

Chapter 4: Lists, Logic and Loops

- Arrays
- Block Statements
- If Conditional
- Ternary Operator
- For Loops
- While and Do Loops

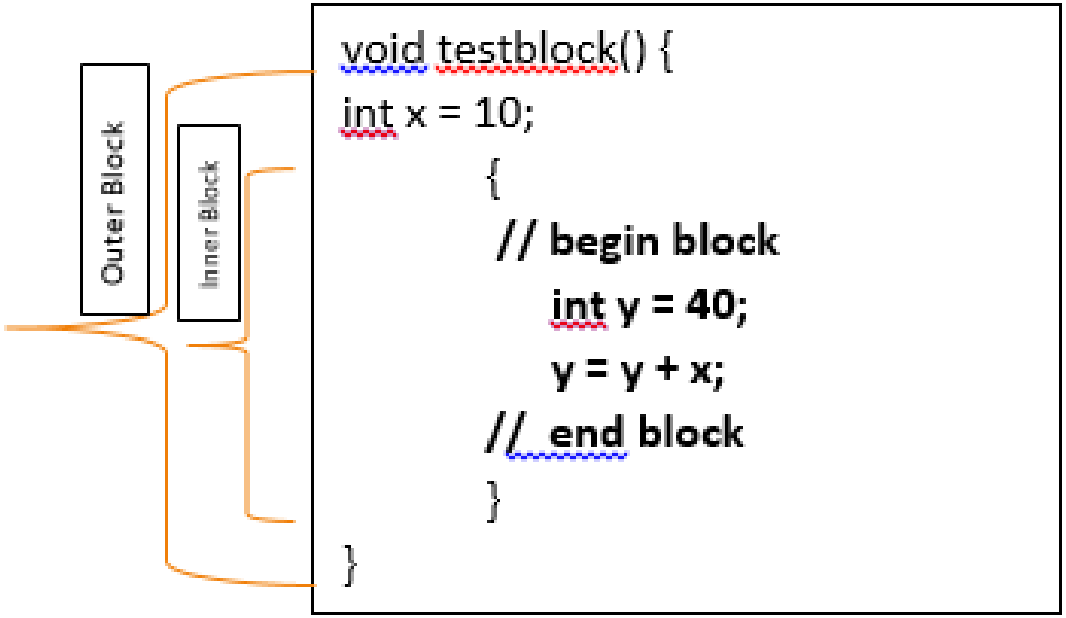
Arrays

- Store a list of items
- Declare the array variable
- Create array object
- Store information
- String[ ] requests; or
- String requests[ ];
- Index starts at 0
- Multidimensional String [ ][ ] requests;



Block Statements

- Create a scope for local variables
- Used in classes and methods, logic and loops structures



If Conditionals

- The key to any programing lanquage is its decision capabilities
- If (Boolean) ie x>=3, outOfGas
- Else for when we want something else to happen on false

```
if (arguments.length < 3) {
        System.out.println("Not enough arguments");
} else {
        System.out.println("Just Enough!");
}
```

Switch Conditionals

- Common to test a variable against a value and if it doesn't match check against another

```
char selection;
bool invalid = true;

cout << "What flavor would you like?\n
Enter O for orange, A for apple or P for pear: ";

while(invalid) {

    cin >> selection;

    switch(selection) {

        case 'O':
            cout << "Orange selected.\n";
            invalid = false;
            break;
        case 'A':
            cout << "Apple selected.\n";
            invalid = false;
            break;
        case 'P':
            cout << "Pear selected.\n";
            invalid = false;
            break;
        default:
            cout << "Invalid Selection. Please try again: ";
        }

}
```

Ternary Operator

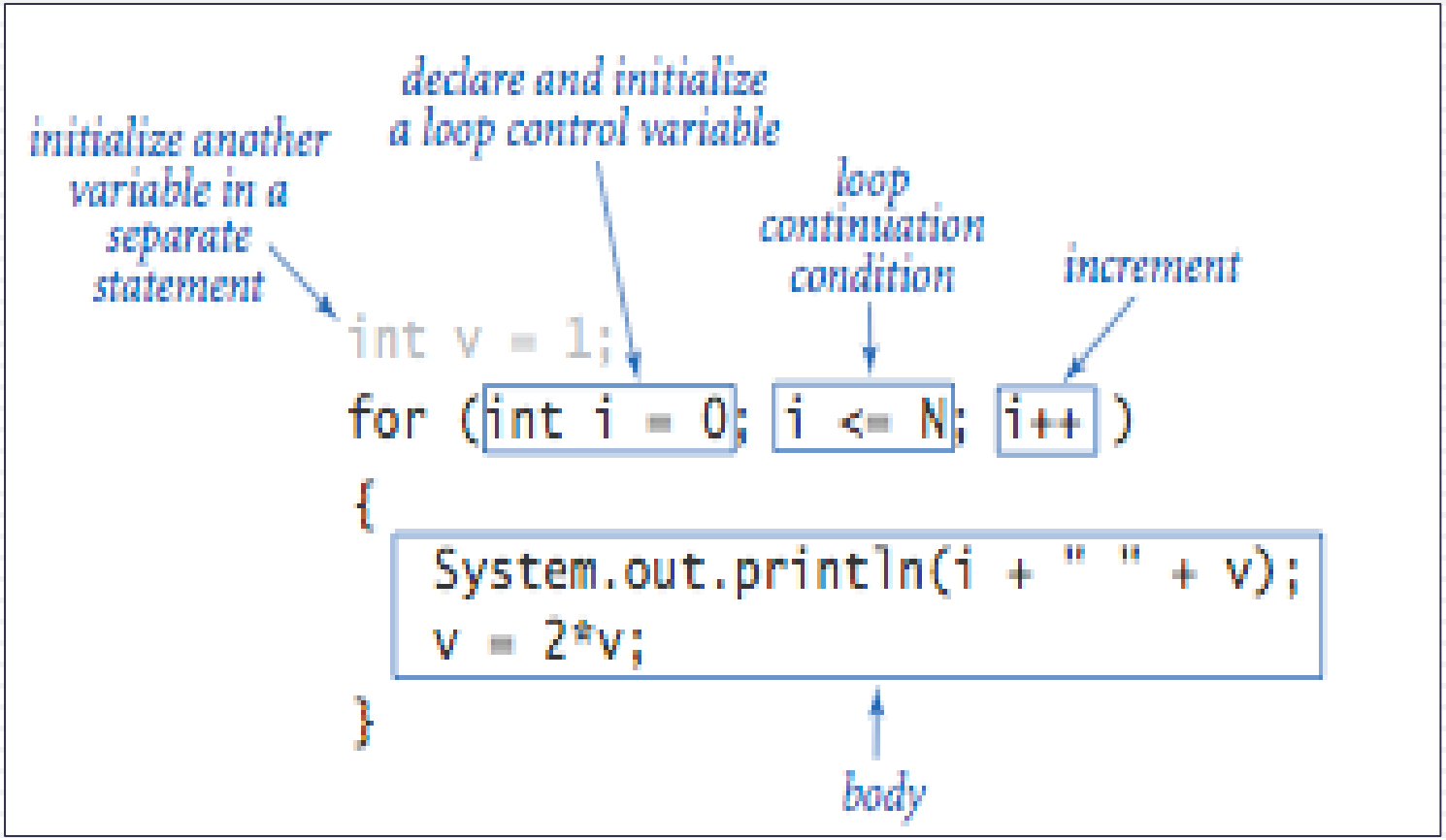
- Three parts

```
test ? trueResult : falseResult;
```

```
int ourBestScore = myScore > yourScore ? myScore: yourScore
```

- The larger value of myScore and yourScore is copied to ourBestScore

For Loops


initialize another variable in a separate statement
declare and initialize a loop control variable
loop continuation condition
increment
int v = 1;
for (int i = 0; i <= N; i++)
{
System.out.println(i + " " + v);
v = 2*v;
}
body

The for Statement

- An example of a `for` loop:

```
for (int count=1; count <= 5; count++)
   System.out.println (count);
```

- The initialization section can be used to declare a variable
- Like a while loop, the condition of a for loop is tested prior to executing the loop body
- Therefore, the body of a for loop will execute zero or more times

The for Statement

- The increment section can perform any calculation

```
for (int num=100; num > 0; num -= 5)
    System.out.println (num);
```

- A for loop is well suited for executing statements a specific number of times that can be calculated or determined in advance
- Each expression in the header of a for loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed

`for` loop Exercises:
How many times is the loop body repeated?

- ```
  for (int x = 3; x <= 15; x++)
      System.out.println(x);
  ```
- ```
  for (int x = 1; x <= 5; x++)
      System.out.println(x);
  ```
- ```
  for (int x = 12; x >= 2; x++)
      System.out.println(x);
  ```

- Write the `for` statement that print the following sequences of values.
  - 1, 2, 3, 4, 5, 6, 7
  - 3, 8, 13, 18, 23
  - 20, 14, 8, 2, -4, -10
  - 19, 27, 35, 43, 51
  ```

Nested Loops

- Loops can be nested inside other loops; that is, the body of one loop can contain another loop.

```
int price;
for (int width = 11; width <=20; width++){

  for (int length = 5; length <=25; length+=5){

    price = width * length * 19; //$19 per sq. ft.
    System.out.print ("  " + price);
  }
  //finished one row; move on to next row
  System.out.println("");
}
```

OUTER

INNER

for (initialization; test; increment)
statement

Nested Loop: Example

- This array has just been populated with daily production. Write code to summarize the annual output of the plant.

Week Day

```
int[ ] [ ] dayValue = new int [52][7];
```

While Loops

- Execute a block repeatedly **while** a specific condition exists (remains true)

```
// Demonstrate the while loop.
class WhileDemo {
  public static void main(String args[]) {
    char ch;

    // print the alphabet using a while loop
    ch = 'a';
    while(ch <= 'z') {
      System.out.print(ch);
      ch++;
    }
  }
}
```

Do While Loops

- Do something while the condition exists (is true)
- Will execute the body of the loop at least once before testing the condition – even if the condition is false at the outset
  - While will always check first then decide whether or not to execute the body

Statements **break** and **continue** allow us to break out of a loop early – Break drops out of the structure; continue starts the loop over with the next iteration

```
Initialization;
do
   {
   Statement 1 ;
   Statement 2 ;
   Statement 3 ;
   ............
   ............
   if ( If Condition)
     break;

   Statement N-1 ;
   Statement N ;
   Increment;
   } while (condition);

OutsideStatement 1;
```

Independent Exercise – 2 Hours

- A. Using countDays() method from the DayCounter.java application, create an application that displays every date in a given year in a single list from January 1 to December 31.
- Name the Java file as YearDisplayer.java
- Run as a Java Application
- B. Create a class that takes words from the first 10 numbers ("one" – "ten") and converts them into a single long integer. Use the switch Statement for the conversion and command-line arguments for the words.
- Name the file WordNumber.java
- Run as command line with your word selection (shown next)

While Loops

- Execute a block repeatedly **while** a specific condition exists (remains true)

```
// Demonstrate the while loop.
class WhileDemo {
  public static void main(String args[]) {
    char ch;

    // print the alphabet using a while loop
    ch = 'a';
    while(ch <= 'z') {
      System.out.print(ch);
      ch++;
    }
  }
}
```

Labeled Loops

- Both break and continue can have an optional label that tells Java where to resume execution of the when you use break or continue, add the name of the label after the keyword itself.

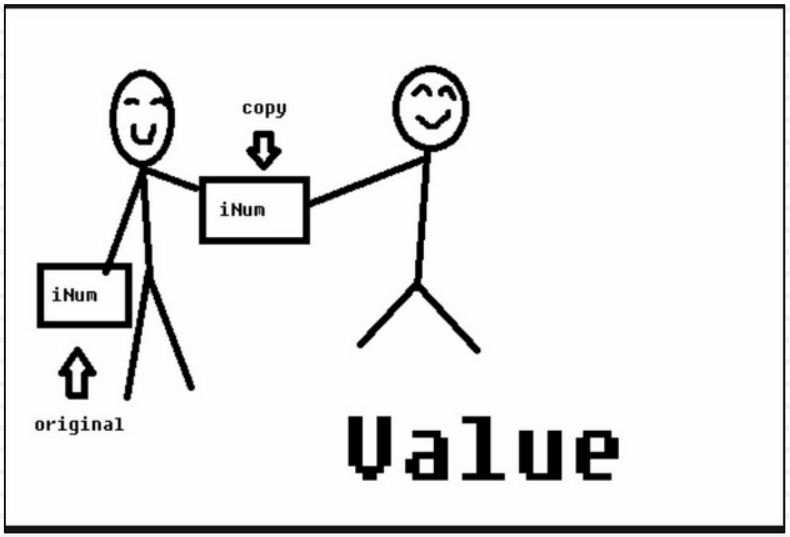
```
out:
for (int i = 0; i < 10; i++) {        //outer loop
while (x < 50) {       // inner loop
        if (i * x++ > 400)
                break out;
}
}
```

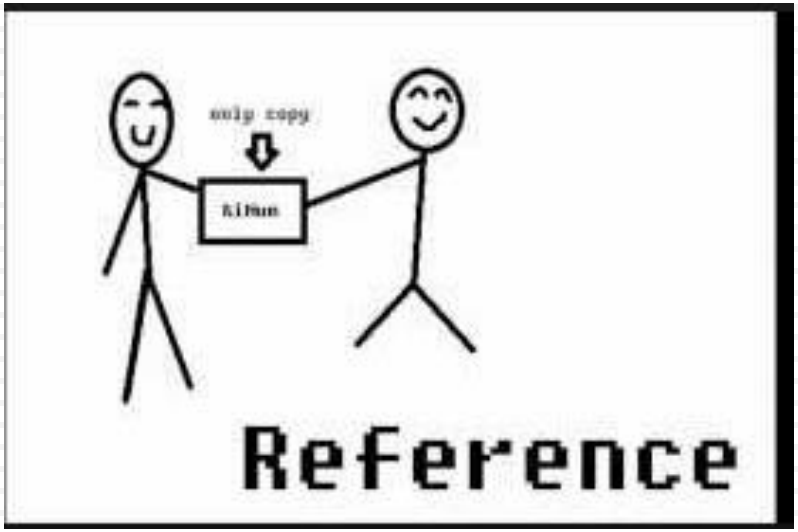
Passing by Reference Vs. by Value

- What is passed into a function is the actual argument and where it is received is the formal arguments
  - Recall that when you call a function, a chunk of memory_is allocated. Critical to the discussion here is that this memory holds the formal parameter values and function local variables.
  - **<u>Pass *by value*</u> means you are making a copy in memory of the actual argument's value** that
    - Use pass by value when you are only "using" the parameter for some computation and not going to change it
  - **<u>*Pass by reference*</u> (also called pass by address), a copy of the address of the actual argument is stored.**
    - Use pass by reference when you are changing the parameter passed in by the sending program.

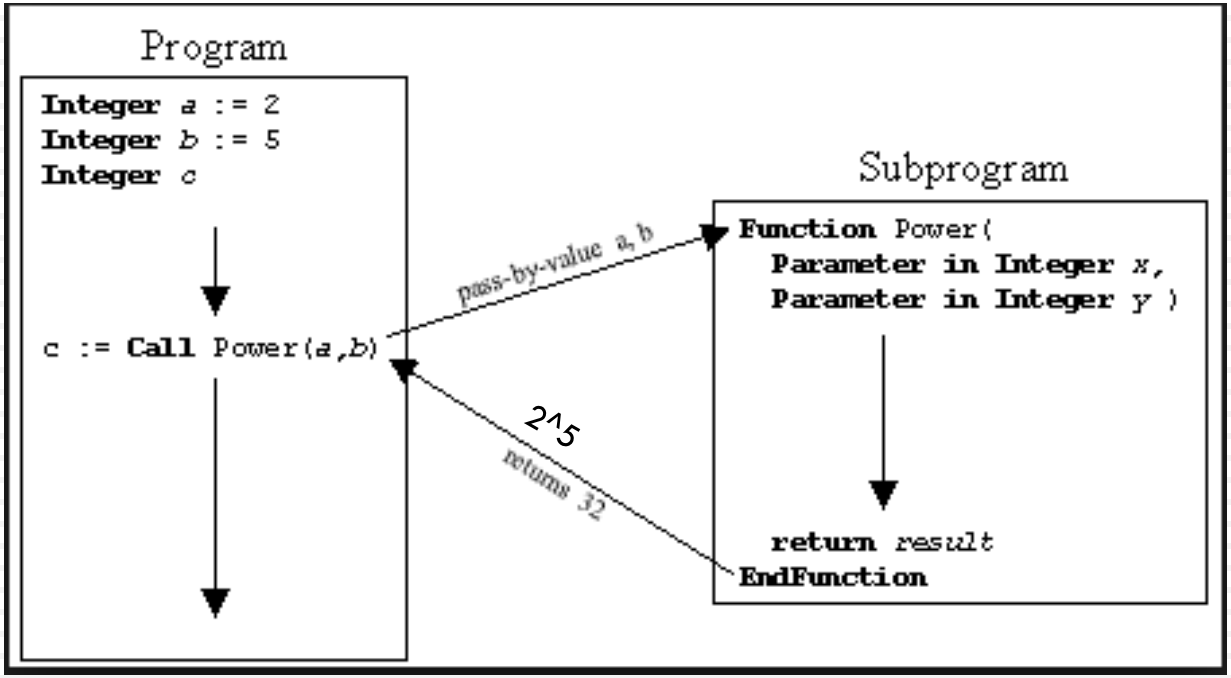
COPY OF the value of the argument

the address of the argument





Pass by Value – Used in Java



X = 2
y = 5

Coding Conventions

- Conventions are important for read-ability
- Code is read more often than it is written
  - Sample from Google:

4.1.1 Braces are used where optional
Braces are used with if, else, for, do and while statements, even when the body is empty or contains only a single statement

4.3 One statement per line
Each statement is followed by a line-break