In **Java**, all code should be placed inside a class declaration. We do this by using the class keyword. In addition, the class uses an access specifier public, which indicates that our class in accessible to other classes from other packages (packages are a collection of classes). We will be covering packages and access specifiers later.

Ex: public class Hello (note that Java is Case sensitive)

Note: The last two lines which contains the two curly braces is used to close the main method and class respectively.

Java Comments: - A comment is indicated by the delimiters "/*" and "*/". Anything within these delimiters are ignored by the Java compiler, and are treated as comments.

Example :

Single Line comments : Only needs a Double slash "//" like this and then you have the Single line comment.

```
Public Class Hello {

Public static void main (String [] args){

        // this is a single line Comment

        System.out.println("Hello World");

        }

}
```

Multi-Line Comments: - A comment is indicated by the delimiters "/*" and "*/". Anything within these delimiters are ignored by the Java compiler, and are treated as comments.

```
Public class Hello {

Public static void main (String[] args0 {

                System.out.println("Hello World") ;

                /* This section is the start of the multi-
line comment)

(And this will be the end part of comment) */

        }

}
```

Coding Guidelines :

- Your Java programs should always end with the .java extension.
- Filenames should match the name of your public class. So for example, if the name of your public class is Hello, you should save it in a file called Hello.java.
- You should write comments in your code explaining what a certain class does, or what a certain method do.

Java Statements and blocks

A **statement** is one or more lines of code terminated by a semicolon.

Ex: System.out.println("Hello world");

A **block** is one or more statements bounded by an opening and closing curly braces that groups the statements as one unit. Block statements can be nested indefinitely.

```
Ex:     public static void main( String[] args ){

        System.out.println("Hello");

        System.out.println("world");

}
```

Java Identifiers

Identifiers are tokens that represent names of variables, methods, classes, etc. Examples of identifiers are: Hello, main, System, out.

Java identifiers are case-sensitive. This means that the identifier: Hello is not the same as hello. Identifiers must begin with either a letter, an underscore "_", or a dollar sign "$". Letters may be lower or upper case. Subsequent characters may use numbers 0 to 9.

Identifiers cannot use Java keywords like class, public, void, etc. We will discuss more about Java keywords later.

Java Keywords

Abstract	double	int	super
Boolean	else	interface	switch
Break	extend	long	synchronized
byte	false	native	this
byvalue	final	new	threadsafe
case	finally	null	throw
catch	float	package	transient
char	for	private	true
class	goto	protected	try

Const	if	public	void
continue	implements	return	while
default	import	short	

Java Literals

Literals are tokens that do not change or are constant. The different types of literals in Java are: Integer Literals, Floating-Point Literals, Boolean Literals, Character Literals and String Literals.

Integer literals come in different formats: decimal (base 10), hexadecimal (base 16), and octal (base 8). In using integer literals in our program, we have to follow some special notations. For decimal numbers, we have no special notations. We just write a decimal number as it is. For hexadecimal numbers, it should be preceeded by “0x” or “0X”. For octals, they are preceeded by “0”.

example:

Floating-Point Literals

Floating point literals represent decimals with fractional parts. An example is 3.1415. Floating point literals can be expressed in standard or scientific notations. For example, 583.45 is in standard notation, while 5.8345e2 is in scientific notation.

Floating point literals default to the data type double which is a 64-bit value. To use a smaller precision (32-bit) float, just append the “f” or “F” character.

Boolean Literals

Boolean have only two values, **TRUE or FALSE.**

Character Literals

Character Literals represent single Unicode characters. A Unicode character is a 16-bit character set that replaces the 8-bit ASCII character set. Unicode allows the inclusion of symbols and special characters from other languages. To use a character literal, enclose the character in single quote delimiters.

For example, the letter a, is represented as ‘a’.

String Literals

String literals represent multiple characters and are enclosed by double quotes. An example of a string literal is, “Hello World”.

Primitive data types

The Java programming language defines eight primitive data types. The following are, boolean (for logical), char (for textual), byte, short, int, long (integral), double and float (floating point).

Logical - boolean

A boolean data type represents two states**: true and false.**

Example: boolean result = true;

Textual – char A character data type (char), represents a single Unicode character. It must have its literal enclosed in single quotes(’ ’).

example: ‘a’ //The letter a ‘\t’ //A tab

To represent special characters like ' (single quotes) or " (double quotes), use the escape character \.

'\'' //for single quotes '\"'
//for double quotes

Although, String is not a primitive data type (it is a Class), we will just introduce String in this section. A String represents a data type that contains multiple characters. It is not a primitive data type, it is a class. It has it’s literal enclosed in double quotes(“”). For example, String message=“Hello world!”

Variables

A variable is an item of data used to store state of objects.

A variable has a **data type** and a **name**. The data type indicates the type of value that the variable can hold. The **variable name** must follow rules for identifiers.

Declaring and Initializing Variables

To declare a variable is as follows

<data type> <variable name> [=initial value];

Note: Values enclosed in <> are required values, while those values enclosed in [] are optional.

Here is a sample program that declares and initializes some variables

```
public class VariableSamples {

public static void main( String[] args ){

 //declare a data type with variable name

// result and boolean data type

boolean result;

//declare a data type with variable name

// option and char data type

char option; option = 'C';

 //assign 'C' to option //declare a data type with variable name

//grade, double data type and initialized

//to 0.0 double grade = 0.0; } }
```

Outputting Variable Data

In order to output the value of a certain variable, we can use the following commands

```
Ex:         int value = 10;
            char txt = x;
            String word = “Hello”;
            Float decVal = 2.5;

System.out.println(“Value” + value);

System.out.println(“Text: “ + txt);

System.out.println(“Greetings: “ + word);

System.out.println(“Decimal Value: “ + decVal);
```

The output will be:

Value: 10

Text: x

Greetings: Hello

Decimal Value: 2.5

System.out.println() vs. System.out.print()

What is the difference between the commands System.out.println() and System.out.print()? The first one appends a newline at the end of the data to output, while the latter doesn't.

```
Ex:         System.out.println(“Hello);
            System.out.println(“World”);
Output:     Hello
            World
```

while in the other one

```
            System.out.print(“Hello “);
            System.out.print(“World”);
Output:     Hello World
```

Operators

In Java, there are different types of operators. There are arithmetic operators, relational operators, logical operators and conditional operators. These operators follow a certain kind of precedence so that the compiler will know which operator to evaluate first in case multiple operators are used in one statement.

Arithmetic Operators

Operator	Use	Description
+	op1+op2	Adds op1 and 2
*****	op1*op2	Multiplies op1 and op2
/	op1/op2	Divies op1 and op2
%	op1%op2	Computes the remainder of op1 and op2
-	op1-op2	Subtracts op1 and op2

Increment and Decrement operators

Aside from the basic arithmetic operators, Java also includes a unary increment operator (++) and unary decrement operator (--). Increment and decrement operators increase and decrease a value stored in a number variable by 1.

```
Example:        count = count + 1;
//increment the value of count by 1

// It’s the same as count++;
```

(Next Page)

Operator	Use	Description
++	op++	Increments op by 1; evaluates to the value of op before it was incremented
++	++op	Increments op by 1; evaluates to the value of op after it was incremented
--	op--	Decrements op by 1; evaluates to the value of op before it was decremented
--	--op	Decrements op by 1; evaluates to the value of op after it was decremented

Example:

int i = 10;

int j = 3;

int k = 0;

k = ++j + i; //will result to k = 4+10 = 14

another Example:

int i = 10;

int j = 3;

int k = 0;

k = j++ + i; //will result to k = 3+10 = 13

Relational operators

Relational operators compare two values and determines the relationship between those values. The output of evaluation are the boolean values true or false.

Operator	Use	Description
>	op1>op2	op1 is greater than op2
>=	op1>=op2	op1 is greater than or equal to op2
<	op1<op2	op1 is less than op2
<=	op1<=op2	op1 is less than or equal to op2
==	op1==op2	op1 is equal to op2
!=	op1!=op2	op1 is not equal to op2

Logical Operators

Logical operators have one or two boolean operands that yield a boolean result. There are six logical operators: && (logical AND), & (boolean logical AND), || (logical OR), | (boolean logical inclusive OR), ^ (boolean logical exclusive OR), and ! (logical NOT)

Operator Precedence

Operator precedence defines the compiler's order of evaluation of operators so as to come up with an unambiguous result.

.	[]	()	
++	--	!	~
+	-		
<<	>>	>>>	<<<
==	!=		
&	|		
^			
&&			
||			
?:			
=			

MODULE 2

Using the Scanner Class to get input

To put data into variables from the standard input device, Java provides the class Scanner. Using this class, first we create an input stream object and associate it with the standard input device.

Sample code:

```
import java.util.Scanner;
//always import the Scanner first before using it
public class Bio {
public static void main(String[] args){
  Scanner sc = new Scanner(System.in);
System.out.print("Name: ");
String name = sc.nextLine();
System.out.print("Age: ");
int age = sc.nextInt();
sc.close();
System.out.println("Name: " + name);
System.out.println("Age: " + age);
  }
}
```

The Output will be:

Name: (The user will input their name, ex. Raven)

Age: (The user will input their age , ex. 19)

Name: Raven

Age: 19

Note: if you pasted this code into an IDE make sure to replace quotation marks like these “ “ because they are different from quotation marks in MS Word and can cause a lot of errors.

Using JOptionPane to get input

```
import javax.swing.JOptionPane;
 public class Welcome{
 public static void main( String args[] ){
 String name;
 name=JOptionPane.showInputDialog("Enter your name");
 JOptionPane.showMessageDialog( null, "Hello "+name +
"\nWelcome to Java Programming!" );
  System.exit(0);
    }
}
```

MODULE 3

Decision Control Structures

The decision control structure are Java statements that allows us to select and execute one set of statements if a condition is true and another set of actions to be executed if a condition is false.

if statement

The if-statement specifies that a statement (or block of code) will be executed if and only if a certain condition statement is true.

```
if(condition)
{
 // Statements to execute if
 // condition is true
}
```

Example:

```
int x = 10;
if(x >=10)
System.out.println(“Pogi si Raven”);
```

if-else statement

The if-else statement is used when we want to execute a certain statement if a condition is true, and a different statement if the condition is false.

```
int x = 10;
        if(x >= 11)
          System.out.println("Pogi si Joshua");
        else
          System.out.println("Pogi si Angelicious");
```

if-else if-else statement

The statement in the else-clause of an if-else block can be another if-else structures. This cascading of structures allows us to make more complex selections.

```
int x = 10;
if(x >= 11)
System.out.println("Pogi si Joshua");
else if(x == 9)
System.out.println("Pogi si Angelicious");
```

```
else if(x == 10)

System.out.println("Raven is the Most Pogi in the PSU Lingayen Campus");
```

switch statement

Switch case statement is used when we have number of options (or choices) and we may need to perform a different task for each choice.

```
import java.util. * ;

public class SwitchStatement {

public static void main(String[] args) {

Scanner sc = new Scanner(System. in );

System.out.println("You have three options of who is the Most Handsome in II-BSCS-A");

System.out.println("1. Raven");

System.out.println("2. Angelicious");

System.out.println("3. Joshua");

System.out.println("Enter your choice:");

int ch = sc.nextInt();

switch (ch) {

case 1:

System.out.println("Congrats you have chosen Raven!");

break;

case 2:

System.out.println("Congrats you have chosen Angelicious");

break;

case 3:

System.out.println("Congrats you have chosen Joshua!");

break;

default:

System.out.println("Wrong input!");

break;

        }

}
```