Java & ACM Libraries

Graphics Program	Console Program	Dialog Program

`import acm.graphics.*;`
`import acm.program.*;`
`public class HelloProgram extends GraphicsProgram`
`{`
`  public void run()`
`    {`
`      add(new GLabel("hello, world", 100, 75));`
`    }`
`}`
	`import acm.program.*;`
	`public class Add2Integers extends ConsoleProgram`
	`{`
	`  public void run()`
	`   {`
	`    print("This program adds two integers\n");`
	`    int n1 = readInt("Enter first number:  ");`
	`    int n2 = readInt("Enter second number: ");`
	`    int total = n1 + n2;`
	`    println("The total is " + total + ".");`
	`   }`
	`}`
		`import acm.program.*;`
		`public class Add2Integers extends DialogProgram`
		`{`
		`  public void run()`
		`   {`
		`    print("This program adds two integers\n");`
		`    int n1 = readInt("Enter first number:  ");`
		`    int n2 = readInt("Enter second number: ");`
		`    int total = n1 + n2;`
		`    println("The total is " + total + ".");`
		`   }`
		`}`

identifier : a name for an object or other entity in a Java program. Must start with either a letter or the underscore symbol, and the remaining characters must all be letters, digits, or the underscore symbol. Examples:

`thirty` (but not 30) `num1` `SubTotal` `NumberOfPods` `ID_A_5` `An_Integer`
`AverageAge` `UserName` `AgeOfTree` `tag_14` `LastWord` `A_Double`

case-sensitive : distinguishes between uppercase and lowercase letters in the spelling of identifiers. Each of the following are distinct identifiers: `rate` `RATE` `Rate`

statement/executable statement : an instruction followed by the computer

object : a memory location (with identifier or name) which can hold numbers or other types of information (`data members`) and may have associated messages available (`member functions`). Simple objects are sometimes called `variables`.

keyword / reserved word : identifiers which have predefined meaning in Java and which you cannot or should not use as names for objects. Examples: `int, double, void, class, if, static`

object declaration/instantiation : a statement which assigns an identifier to a storage location and informs the computer what *type* of data will be stored in the object, as well as what methods are available. Examples:

Declaration only	**Declare & Initialize**
`int NumPeople, count;`	`int DailyGaugeReading = 0.0;`
`double x, y, z, Distance;`	`double wWidth = getWidth();`

assignment statement : an order to the computer telling it to set the value of the object on the left-hand side of the `" = "` to the value of the expression on the right-hand side. Examples:

`count = 0;` `x = y + 2.0;` `MyPoint = new GPoint(1.0, 2.14);`

constant : a value which cannot change during execution of the program

- *unnamed*: literals such as 4.5, 29, 'A', or "This is a literal string"
- *named*: identifiers declared as constants. Examples:

```
private static final  int      MAX = 100;
private static final  double   TAXRATE = 0.085;
```

increment : to increase the value. Examples: `num = num + 1;` `num += 1;` `num++;`

decrement : to decrease the value. Examples: `num = num - 1;` `num -= 1;` `num--;`

arithmetic Operators : `+, - , *, /, %`

Integer Division : when both operands are integer, division (/) yields the whole number (integer) quotient. Examples: `15 / 6 is 2` `10 / 12 is 0`

Modulus operator (`%`): both operands *must* be integer, and `%` yields the whole number (integer) remainder. Examples: `15 % 6 is 3` `10 % 12 is 10`

For `+, -, *, and /`, if both operands are integer, the resulting value will be an integer. If either or both operands are floating point, the result will be floating point.

mixed mode arithmetic : integer and floating point operands occur in the same expression. Only at the point where a floating point operand occurs will the result be a floating point value.

type casting / type conversion : the *explicit* (programmer forced) conversion of a value from one data type to another. Used to avoid errors and for readability. Examples:

`(double) AnIntValue` and `(char) AnIntValue`

Precedence of Arithmetic Operators :

`( )`	*highest* - anything in parentheses
`*, /, %`	at same level: *evaluated left to right*
`+, -`	*lowest: evaluate left to right*

Output : `print()` and `println()`, literals, constants, objects and expressions. Example:

```
print("This is a literal string \n"
      + 29 + "\n"                    // an unnamed constant
      + MAX + "\n"                   // a named constant
      + anIntVar);                   // an integer object
println();
```

Formatting Floating–Point Values — to display **dollar** amounts

```
Import:       import java.text.NumberFormat;
Declaration:  NumberFormat formater = NumberFormat.getCurrencyInstance();
Use:          formater.format(amountAsDouble);
```

Formatting Floating–Point Values — to enforce specific number of digits in output

```
Import:       import java.text.DecimalFormat;
Declaration:  DecimalFormat pattern = new DecimalFormat("####0.00");
Use:          pattern.format(amountAsDouble);
```

Input : `readInt()`, `readDouble()`, and `readLine()` are used to prompt the user and get values entered on the keyboard. Examples:

```
int value = readInt("Please enter an integer:  ");
double x = readDouble("Enter x-value:  ");
String line = readLine("Enter phrase to check: ");
```

flow of control : the order in which the computer executes statements in a program

control structure : a statement used to alter the normally sequential flow of control (loop, selection)

selection statement : a branching control structure that decides between alternative actions. Examples: `if` and `switch` statements

logical or Boolean expressions : expressions which evaluate to either `true`(1) or `false`(0)

relational operators : used to compare two values: `<  <=  >  >=  ==  !=`

Boolean operators : used to join two Boolean expressions: AND (`&&`), OR (`||`), and NOT (`!`)

unary operator : requires single operand

binary operator : requires two operands

Extended precedence of operators :

```
( )
! (negation)   unary+  unary-
*  /  %
+  -
<  <=  >  >=
==   !=
&&
||
=
```