Lecture # 4

Arithmetic Operators and Constants

Introduction to Programming

Variables

- A *variable* is a named location in a computer's memory.
- A *variable* is required for every piece of information that your program will store.
- A program can manipulate the values stored in variables to solve specific problems.

Terminology

- ***Operators*** are used to perform an action
- ***Operands*** are the data on which the action is performed
- An ***expression*** is a statement made up of ***operators*** and ***operands***
- For example:

  Expression: `balance = balance - 100`
  Statement: `balance = balance - 100;`
  Operators: `= -`
  Operands: `balance 100`

Assignment operator =

- The *assignment operator* is used to assign a value to a variable.
- When evaluating an expression, read from right to left.
- For example:

```
double balance;  //declare a variable
balance = 100.0;  //assign a value to variable
balance = balance + 20;   //????
```

Arithmetic Operators

- The arithmetic *operators* are used to perform *operations* on variables

Operation	**Java Operator**
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus (remainder)	%

Addition operator +

- The addition operator (+) is used with number variables to add those values together.
- The result of an addition may be stored in a variable.
- For example:

```
double balance = 100.0;
double interest = 12.42;
balance = balance + interest;
```

Program to add two numbers

```
public class Add2Numbers
  {
     public static void main(String[ ] args)
     {
        int no1;  //declare variables
        int no2;
        int result;

        no1= 2;   //assign values
        no2= 3;
        result = no1 + no2; //perform calc

        System.out.print(result); //display result
     }
  }
```

Concatenation operator +

- The + symbol used as *addition operator* when used with numeric values or variables
- used as *concatenation operator* when used with strings
  - to *concatenate* strings together. (Concatenation operator – overloaded operator)
- This is very useful in `System.out.print()` statements.
- For example:

```
balance = balance + interest;
System.out.print("Balance: " + balance);
```

Program to add two numbers

```
public class Add2Numbers
{
    public static void main(String[ ] args)
    {
        int no1 = 2;
        int no2 = 3;
        int result = 0;

        result = no1 + no2;
        System.out.print ("The result is " + result);
    }
}
```

Addition operator

Concatenation operator

Subtraction operator -

- The ***subtraction operator*** is used with variables to subtract one value from the other.
- Again, the result of a subtraction may be stored in a variable.
- For example:

```
double balance = 100.0;
double bankCharge = 1.42;
balance = balance - bankCharge ;
```

Multiplication operator *

- The ***multiplication operator*** is used with variables to multiply those values together.
- For example:

```
double balance = 100.0;
double interest = 0;
double interestRate = 0.05;

interest = balance * interestRate;
balance = balance + interest;
```

Division operator /

- Important difference between integer division and decimal (floating point) division
- If both operands are *integers*, the result is an *integer*. The remainder is disregarded

  `7 / 4` yields `1`

- If one or both operands are *real numbers (double)*, the result is a *double*.

  `7.0 / 4` yields `1.75`

Division operator /

- The ***division operator*** is used with variables to divide one value by another.
- For example:

```
int mathsMark = 75;
int progMark = 58;
int average;

average = (mathsMark + progMark) / 2;
```

What value is assigned to average?

Division operator /

- When used with `int` variables, the division operator will disregard any remainder.
- For example:

```
int answer;
answer = 11 / 4;     // answer is 2
```

- For example:

```
double answer;
answer = 11 / 4.0;   // answer is 2.75
```

Modulus operator %

- The ***modulus operator (%)*** is used with `int` variables to find the remainder after division.
- For example:

```
int answer;
answer = 7 % 4;      // answer is 3
```

- The modulus operator (%) returns the remainder after *integer division*

Modulus operator %

Examples of the modulus operator in Java	
Expression	**Value**
29 % 9	2
6 % 8	6
40 % 40	0
10 % 2	0

Modulus operator example

A film last 132 minutes. How long is this in hours and minutes?

```
int hours;              //declare variables
int mins;
int filmLength;

filmLength = 132;       //assign values

hours = filmLength /60;
mins = filmLength % 60;
```

Operator precedence

- The multiplication, division, and modulus (*, /, %) operators take precedence over the addition and subtraction (+, -) operators.
- For example:

```
int answer;
answer = 5 + 2 * 2     // answer is 9
```

Operator precedence

- Use brackets to override operator precedence. Expressions in brackets are always evaluated first.
- For example:

```
int answer;
answer = (5 + 2) * 2;   //answer is 14
```

- The Assignment operator (=) has the lowest level of precedence, so it is always evaluated last

Associativity of arithmetic operators

Precedence

/ % * have higher precedence than + -,
i.e. 3 + 2 * 5 gives 13

Associativity

- if operators have same precedence then look at associativity
- Arithmetic operators work from left to right
  i.e. 13 % 2 * 5 gives 5
- Associativity of assignment operator is right to left

Constants

- There will be times where data items in a program have values that do not change
- Examples:
  - Maximum score in exam 100
  - Number of hours in day 24
  - Value of π (*pi*) in Maths 3.14159
- Values that remain constant throughout a program should be named and declared as **constants**

Constants

- Constants are declared like variables, but are preceded by the keyword **final**
- They are always initialised to their fixed value

```
final int HOURS = 24;
```

```
final double PI = 3.14;
```

Constants

- It is conventional to use all upper case letters in a constant name
- If the name contains two or more words, they may be joined using an underscore

```
final int DAYS_IN_WEEK = 7;
```

Why use Constants?

1. A constant name is much easier to remember than a value. E.g. PI
2. Once declared in a program, constants cannot be modified. Attempting to do so will result in a compilation error. (PI = 4; //will cause error)
3. Constants make code easier to update, as the value is defined in one place only.
4. Constants greatly improve the readability of code.

Program Example

```
public class AreaOfCircle
{
     public static void main(String args[])
     {
        final double PI = 3.14159;
        double radius = 10;

        double area;

        area = PI * radius * radius;

        System.out.print("Area is  " + area);
     }
}
```

Program Example

```
public class WeeksInYear
{
   public static void main(String args[])
   {
      final int DAYS_IN_YEAR = 365;
      final int DAYS_IN_WEEK = 7;
      int weeksInYear;

      weeksInYear = DAYS_IN_YEAR / DAYS_IN_WEEK;
      System.out.print("There are " + weeksInYear);
      System.out.print(" weeks in a year");
   }
}
```