Variables, Stored Data and Calculations

Learning Objectives

- To introduce and become familiar with the following:
  - **Store data in the computer's memory using variables**
  - **Declare, initialise and assign values to variables**
  - **Output variable values to the screen**
  - **Use variables in numerical calculations**
  - **Casting**
    - **More advanced and worthy of self study when you are confident you fully understand the earlier material**

Some Thoughts

- What is the first thing a typical ATM cash machine asks for following a card being inserted?
  - What does it do with this information?
  - How does it do it?
- The programs that we have implemented so far print text (messages) that we have “hard coded” to the screen
- The actual text is included as part of the program
  - What are the implications of this?
  - What if I wanted to carry out calculations?
  - What if I want store information to print out later?

Types of Information

- Java provides the facility for storing **data** in the memory of the computer
- A variety of different types of data may be stored
  - Positive and negative whole numbers ➔ **integers**

    e.g. -2 3 -9 100 20097543
  - Positive and Negative decimal numbers ➔ **real numbers**

    e.g. -0.1 3.2 1.16 209.105 7.0 -32.18
  - Letters, numerals or other symbols ➔ **characters**

    e.g. 'A' 'a' 'c' 'D' '~' '$' '?'
  - **Sequence of characters** ➔ **Strings**

    e.g. "Dublin" "Programming" "I.T.T."

Storing Data

- Before attempting to store data in the memory of a computer the programmer must first:

A Indicate the **TYPE of data** to be stored

WHY? Each different kind of data requires different amount of storage

B Provide a NAME (identifier) as a label under which the data is stored

WHY? This is the way in which the computer accesses the data

Variables and Variable Identifiers (1)

- Suppose we store data (such as an amount) at a particular named location
- Normally we may want the value of this data to be changed at some stage in the program
- Hence the data is described as **changeable** or **variable**
- For example, we may wish to store a **password** and give it some initial value.
  - Later, we might like to change the value of our **password**

Variables and Variable Identifiers (2)

- When storing information we need to provide a name of where we wish to store the information
- This name we give it is referred to as the **identifier**
- Summary:
  - An **identifier** is used to store data in a named memory location
  - This is then referred to as a **variable identifier**
  - This is shortened to the term **variable**

Visualisation – VERY IMPORTANT

age	initial	height	bankBalance
18	'E'	1.76	-30.00

mark	currency	average	area
62	'$'	32.6	130.00

Identifiers

- Identifiers are names or labels chosen by YOU (the programmer)

- They are used as names of (for example):
  - Programs (or classes in Java) e.g.

    LetterE1 is the program identifier in **class LetterE1**

    LetterE2 is the program identifier in **class LetterE2**
  - Sections of a program (or methods) in Java

    main(String args [])
  - Items of data stored in memory

RULES for Variable Identifiers

- Variable Identifiers should begin with a **letter**
  - I INSIST you ALWAYS use a lower-case letter
- This may be followed by any number of letters or numerals or the underscore character '_'
- Spaces are not permitted
- Reserved words may not be used as an identifier
  - Reserved words have a particular/special meaning in Java
  - Reserved words include: public, class, main, args …

GOOD PRACTICE for Variable Identifiers

- Variable identifiers **should** begin with a **lower-case letter**
  - So important it is almost a rule
- Use a mixture of upper and lower case letters
- Use a variable name that has a meaningful association with the data you are representing
- Where a variable identifier is invented by combining a series of English words, capitalise the first letter of each new word e.g.
  - `currentValue` (store current value of a stock market share)
  - `myFavouriteClub` (store the name of my favourite club)
  - `myPetsName` (store the name of my pet dog)

Caution – Case Sensitivity

- REMEMBER Java is **Case Sensitive**

- Upper and lower case versions of the same letter are treated as different letters
  - Hence `myValue` and `myVALUE` are different

- Good examples of valid identifiers in Java include:

```
length      breadth     area
hoursWorked
side1       side2       side3
firstName   surname     address     homeTown
```

Test Yourself …

- Suggest suitable identifiers for the following variables:
  - someone's height **(real number e.g. 1.80)**
  - someone's home town **(String e.g. "Dublin")**
  - the name of this module **(String i.e. "Software Dev I"**
  - someone's age **(int e.g. 31)**
  - the colour of someone's car **(String e.g. "white")**
  - someone's average mark in a series of subjects **(real number e.g. 81.3.**

Data Types

- Java allows the programmer to store a variety of different types of data in memory
- Some of these are often referred to as simple or **primitive data types** (non-object-orientated)
- The main types of data to be stored are numbers and text

Kinds of Numbers

- **Integers**
  - positive or negative whole numbers, i.e. numbers with no figures after the decimal point (4, 77, -20)
- Java allows for 4 types of integer:

  `byte` `short` `int` `longint`
- **Real numbers**
  - positive or negative numbers with figures after the decimal point (4.6, 77.35, -20.4167)
  - Java allows for 2 types of real number:

    `float` `double`
- The most commonly used are `int` and `double`

Text

- **Characters**
  - single letters or numerals or other symbols
  - In Java a single character is placed INSIDE SINGLE QUOTES

    e.g. **'D'** **'e'** **'$'** **‘*’**

- **Strings**
  - A **`String`** is a sequence of characters such as a name, an address, a sentence, etc.
  - In Java a String is held inside double quotes

    eg “Martin McKinney" **"Programming"** **"Room MS1001"**
  - Strictly speaking **Strings** are NOT a primitive data type, they are **objects (deal with this later)**

Boolean

- The **`boolean`** represents the value **`true`** or the value **`false`**

- These true and false values are defined using the reserved words **`true`** and **`false`** respectively

Variable Declarations

Declaring Variables

- The process of **Variable Declaration** involves:
  - indicating the **type of data** to be stored AND
  - providing an **identifier** to hold data in memory
- It is also known as **DECLARING a Variable**
- In Java a **variable declaration statement** takes the form:

```
datatype variableIdentifier(s);
```

A reserved word indicating the data type to be stored

A sequence of one or more identifiers, one for each of the variables to be used

The declaration must be terminated by a semi-colon (;)

Examples

Datatype	Variable identifier
`int`	`length;`
`int`	`breadth, digit;`
`double`	`rate;`
`double`	`side1, side2, side3;`
`char`	`letter;`
`char`	`initial, startLetter, endLetter;`
`String`	`name;`

Self-test

What is the difference between the following sections of variable declaration statements?

```
int breadth, digit;
```

AND

```
int breadth;
int digit;
```

Variables – more information

- Java variables can be declared anywhere inside the curly brackets **{ . . . }** of a class

- Once a variable has been declared, data of the appropriate type may be stored in it by means of a simple assignment statement of the form:

```
variable Identifier = item of data or value;
```

NB Read the '=' symbol as 'is assigned the value'

Variable Identifier = Data Item or Value;

Examples:

```
length = 25;
breadth = 35;
digit = 5;
```

} Integers (int)

```
side1 = 4.7;
side2 = 6.3;
side3 = 9.8;
```

} Real Numbers (double)

```
letter = 'B';
```

← Character (char)

```
name = "Martin";
```

← String (String)

- Statements of this type are often referred to as an **assignment statement** i.e. data is being assigned to the variable using the **assignment operator '='**

Initialisation

- On the first occasion in a program that data is stored in a variable the process is known as **initialising the variable** i.e. giving it its first value

```
//    Declare an integer variable called length
int length;

//    Assign a value of 25 to length
//    length 'is assigned the value' 25
length = 25;
```

Combined Declaration & Initialisation

- Declaring a variable AND initialising it with data can be combined into a single process

- Hence, data is assigned to a variable as part of the declaration statement

```
int length = 25;
```

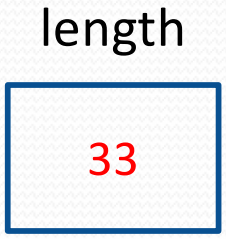
```
double side1 = 4.7;
```

One Value & One Value Only

- Different values can be assigned to the same variables later in the program

- Assigning a new value to a variable will overwrite (i.e. wipe out) any value stored earlier

```
int length = 25;
length = 33;
```



The previous value of 25 is lost forever

Displaying Data

- Data which has been stored in variables can be output to the screen using the output instructions:

```
System.out.print()
System.out.println()
JOptionPane()
```

Displaying Data

Consider a statement to declare and initialise a variable:

```
int length = 25;
```

This could be followed later in the program with an output statement:

```
System.out.println(length);
```

This would display the value **25** onscreen AND leave the cursor at the start of the next line:

```
25
```

However ….

Printing a number on its own to the screen is NOT very useful for the end user!!

- Preferable to provide some message along with the data to provide context

- For example:

```
The room is 25 metres in length
```

We will now look at combining textual information and variables together

Concatenation

- A **String** and a **Variable** can be output to the screen together, to ensure output is informative to the user
- The **String** is really a message to provide context
- We do this in Java by putting a plus sign (**+**) between the String (message) and the variable inside brackets in the output statement
- This process can be referred to as concatenation

Example of Concatenation

```
int size = 25;

System.out.println("The value stored in the variable
                    called size is " + size);
```

The above code would display the following message onscreen:

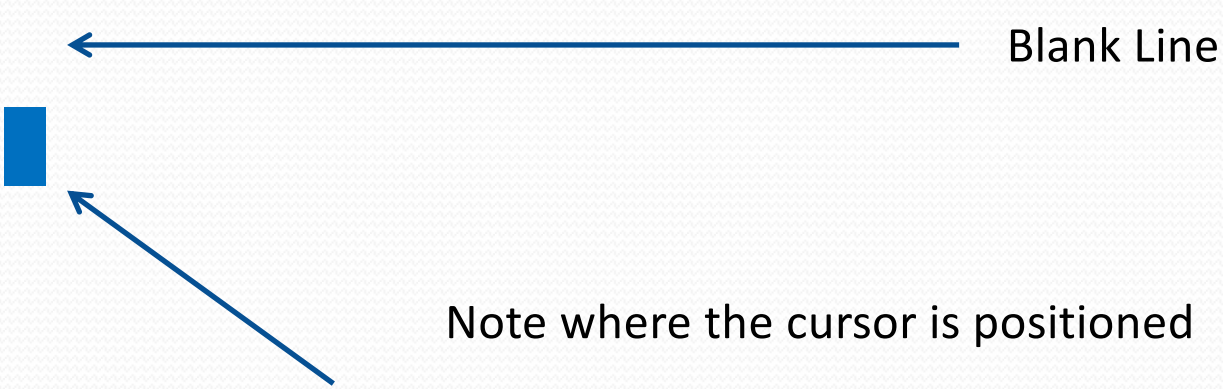
```
The value stored in variable called size is 25
```

UsingVariables1.java

```
7   class UsingVariables1 {
8
9     public static void main (String [] args) {
10
11      // Declaration and initialisation of 3 variables
12      char letter = 'A';
13      int number1 = 25;
14      double number2 = 35.67;
15
16      // Values stored in variables can be output to screen
17      System.out.println("Value stored in letter is " + letter);
18      System.out.println("Value stored in number1 is " + number1);
19      System.out.println("Value stored in number2 is " + number2);
20      System.out.println();
21
22    }//main
23  }//class
```

Program Output

```
Value stored in letter is A
Value stored in number1 is 25
Value stored in number2 is 35.67
```



Poor Code – why??

```
public class Welcome {

   public static void main (String [] args) {

      int number2;
      System.out.println("Number 1 is " + number1);
      System.out.println("Number 2 is " + number2);
   }//main

}//class
```

Answers:

Variable `number1` is never declared (and hence never initialised)

Variable `number2` is declared, but never initialised

Rules

- **A** variable must be declared (using a variable declaration statement) before it can be used in a program.
- A variable cannot be assigned a value or printed out etc. until the **variable declaration statement** has been completed
- The **data** assigned to a variable **must** be of the **same type** as the type indicated in the variable declaration
  - Integer variables need integer information
  - Character variables need character information
- The '=' symbol is known as the **assignment operator**

UsingVariables2.java

```
22      // Values of the variables can be changed
23      letter = 'B';
24      number1 = 157;
25      number2 = -157.69;
26
27      // Values stored in variables output to screen
28      System.out.println("Value now in letter is " + letter);
29      System.out.println("Value now in number1 is " + number1);
30      System.out.println("Value now in number2 is " + number2);
31      System.out.println();
32
33    }//main
34  }//class
```

Program Output

```
Value stored in letter is A
Value stored in number1 is 25
Value stored in number2 is 35.67

Value now in letter is B
Value now in number1 is 157
Value now in number2 is -157.69
```

Assigning Values Between Variables

- It is possible to transfer data from one variable to another using an assignment statement
- **Note - The two variables must have been declared to be of the same data type**

```
int length, breadth;  //  declare two variables
                      //       to hold two integers

length = 25;          //  store the value 25 in the
                      //       variable length

breadth = length;     //  place a copy of the value held
                      //       in length into breadth
```

- What values do **length** and **breadth** hold now?

Similar – but different

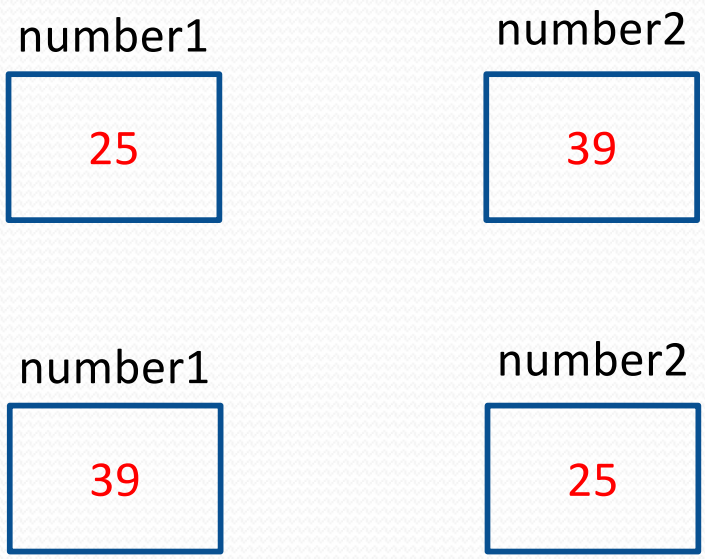
- A similar section of program (with different results) might be as follow:

```
int length, breadth;    //  declare two variables
                        //      to hold two integers

length = 25;            //  store the value 25 in the
                        //      variable length

breadth = 39;           //  store the value 39 in the
                        //      variable breadth

length = breadth;       //  place a copy of the value
                        //      held in breadth into length
```

- What values do **length** and **breadth** hold now?

Swapping the Values of 2 Variables

- Suppose we wish to SWAP the values held in two variables number1 and number2



- We MUST use a third variable as shown overleaf

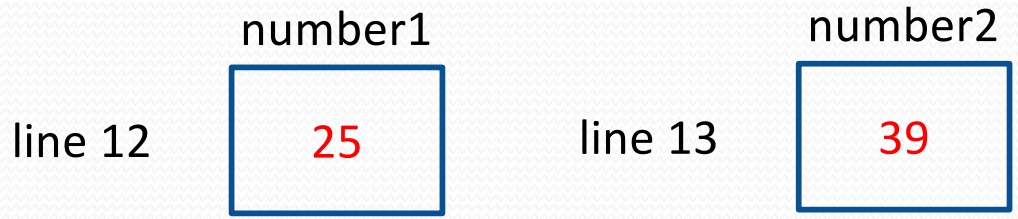
SwapNumber.java

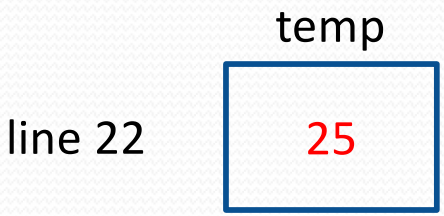
```
7   public class SwapNumbers {
8
9     public static void main(String [] args) {
10
11      int number1, number2;  // Declare 2 integer variables
12      number1 = 25;          // Store 25 in number1
13      number2 = 39;          // Store 39 in number2
14      int temp;              // Declare third variable to act
15                             //         as a temporary variable
16
17      // Print out initial values to screen
18      System.out.println("The value of number1 is " + number1);
19      System.out.println("The value of number2 is " + number2);
20      System.out.println();
```

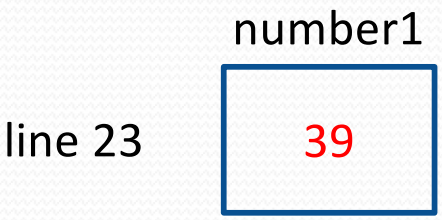
SwapNumber.java (contd.)

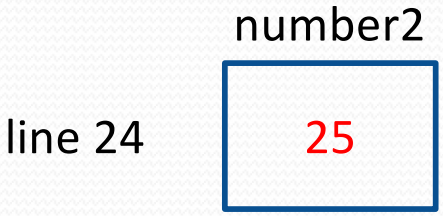
```
22      temp = number1;     //  Copy value of number1 into temp
23      number1 = number2; //  Copy value in number2 into number1
24      number2 = temp;     //  Copy value in temp into number2
25
26      // Print out new values to screen
27      System.out.println("Value of number1 is now " + number1);
28      System.out.println("Value of number2 is now " + number2);
29
30      }//main
31  }//class
```

Visualisation









Program Output

```
The value of number1 is 25
The value of number2 is 39
```

Output from lines 18-20

```
Value of number1 is now 39
Value of number2 is now 25
```

Output from lines 27-28

Mathematical Calculations in Java
SELF READING

Mathematical calculations

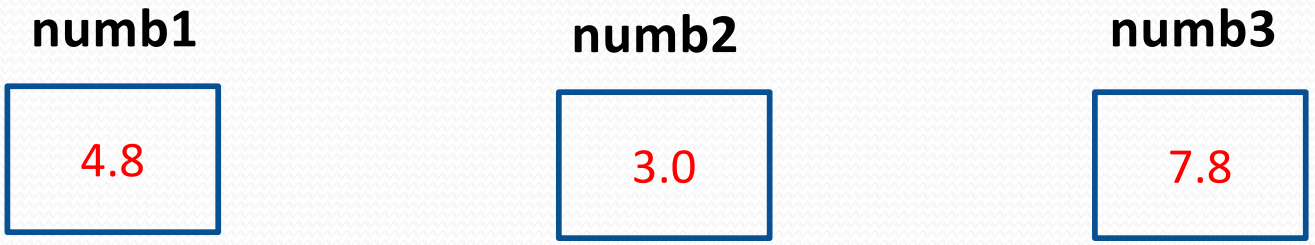
- Once numerical values have been stored in (appropriate) variables they can be used within the program for mathematical operations
- The following mathematical operations can be performed:
  - add (+)
  - subtract (-)
  - multiply (*) and
  - divide (/)
  - Also remainder on division (%)
- Results or outcomes of a mathematical operation can then also be stored in a variable

Operations & Operators

Mathematical Operation	Mathematical Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder on Integer Division	%

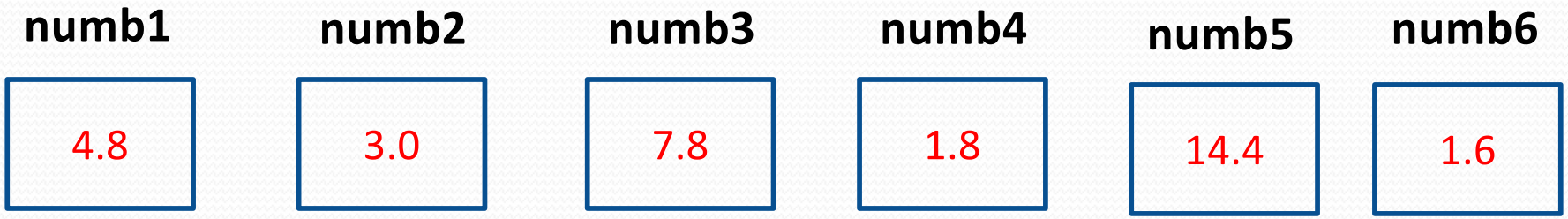
Code (1)

```
// Declare 3 variables to hold decimal numbers
double numb1,numb2,numb3;

// Store value 4.8 in numb1
numb1 = 4.8;

// Store value 3.0 in numb2
numb2 = 3.0;

// Add numb1 to numb2 - store the result in numb3
numb3 = numb1 + numb2;
```



Code (2)

```
// Declare 3 variables to hold decimal numbers
double numb4, numb5, numb6;

// Subtract numb2 from numb1 - store result in numb4
numb4 = numb1 - numb2;

// Multiply numb1 by numb2 - store result in numb5
numb5 = numb1 * numb2;

// Divide numb2 into numb1 - store result in numb6
numb6 = numb1 / numb2;
```



Difference Between / and %

- If we use the symbol **'/'** to divide two integers the outcome will only be the **whole number part of the result**

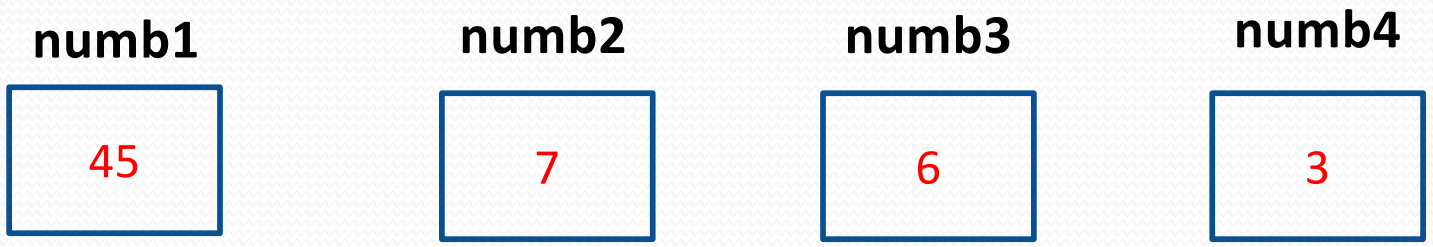
So:	**18 / 7**	**is simply**	**2**
	18 / 6	**is simply**	**3**

- If we use the symbol **'%'** to divide two integers this will **ONLY** return the **remainder**

So:	**18 % 7**	**is simply**	**4**
	18 % 6	**is simply**	**0**
	18 % 4	**is simply**	**2**

Dividing Integers

```
// Declare four variables to hold integers
int numb1, numb2, numb3, numb4;

// Store the value 45 in the variable numb1
// AND store the value 7 in the variable numb2
numb1 = 45;
numb2 = 7;

// Divide numb2 into numb1 - store result in numb3
numb3 = numb1 / numb2;

// Divide numb2 into numb1 - store remainder in numb4
numb4 = numb1 % numb2;
```



Mathematical Expressions

- A mathematical expression is the name given to the process of combining a mixture of values and variables using mathematical operators to produce a result

- For example, the formula for the area of a circle

  Area = Πr^2 rewritten in Java results in:

```
areaOfCircle = 3.1416 * radius * radius;
```

OR

```
areaOfCircle = Math.PI * Math.pow(radius,2);
```

Order of Precedence

- Where there is a mixture of mathematical operators the normal laws of operator precedence apply:

Multiplication & Division	*	/	%
Addition & Subtraction		+	-

- Example 1

3 * 7 – 6 + 2 * 5 / 4 + 6

➔ **21 – 6 + 10 / 4 + 6**

➔ **21 – 6 + 2 + 6**

➔ **23**

- Example 2

7.0 + 3.5 * 2.0

➔ **7.0 + 7.0**

➔ **14.0**

Brackets Change Precedence

- If we wanted the addition to take place before the multiplication we can use round brackets ( ) in the normal way

- Example 1:

3 * ((7 – 6) + 2) * ((5 / 4) + 6)

= 3 * (1 + 2) * (1 + 6)

= 3 * 3 * 7

= 63

- Example 2:

(7.0 + 3.5) * 2.0

= 10.5 * 2.0

= 21.0

Examples

- What is the result of each of the following?

a)	3 + 5 – 3 * 8 / 6	Answer: 4
b)	4 + 5 * 4 % 3	Answer: 6
c)	-8 * 4	Answer: -32
d)	-4 / 3 + 5	Answer: 4
e)	8 + 7 % 2	Answer: 9
f)	6.3 + 2.7 / 3	Answer: 7.2
g)	(6.3 + 2.7) / 3	Answer: 3.0
h)	6.3 + (2.7 / 3)	Answer: 7.2

Example (Rectangle1.java)

- A Java application (Rectangle1.java) has values stored to represent the length and breadth of a rectangle. Calculate and output the perimeter and the area

Calculations:

perimeter = 2 x (length + breadth)

area = length x breadth

Output:

"Rectangle Area = " + area

"Rectangle Perimeter = " + perimeter

```
8  public class Rectangle1 {
9
10    public static void main(String[] args) {
11
12      // Declare 4 variables to hold length,
13      //     breadth, perimeter and area
14      double length, breadth, area, perimeter;
15
16      // Assign values to length and breadth
17      length = 24.7;
18      System.out.println("Rectangle length  = " + length);
19
20      breadth = 25.9;
21      System.out.println("Rectangle breadth = " + breadth);
22
23      // Calculate, store and print out the area
24      area = length * breadth;
25      System.out.println("Rectangle Area = " + area);
26
27      // Calculate, store and print out the perimeter
28      perimeter = (length + breadth) * 2.0;
29      System.out.println("Rectangle Perimeter = " + perimeter);
30
31   }//main
32 }//class
```

Program Output

```
Rectangle length  = 24.7
Rectangle breadth = 25.9
Rectangle Area = 639.7299999999999
Rectangle Perimeter = 101.19999999999999
```

The output looks untidy.

How would you format the output from the program so that all numbers are output with 2 decimal places?

Augmented Assignment Operators in Java

(MORE ADVANCED)

Can be left until you are more confident

Augmented Assignment Operators

- Effectively shortcut operators that allow you to take a variable as one of its arguments and then assigns the result back to the same variable

Use with caution until you are totally comfortable with their meaning.

Augmented Assignment Operators

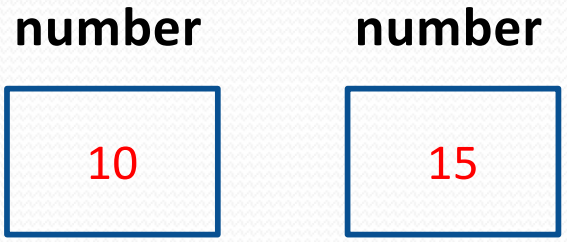
The operators **+, -, *, / and %** can be used with =

Operator	Name	Example	Equivalent
+=	Addition assignment	i += 8	i = i + 8
-=	Subtraction assignment	i -= 8	i = i - 8
*=	Multiplication assignment	i *= 8	i = i * 8
/=	Division assignment	i /= 8	i = i / 8
%=	Remainder assignment	i %= 8	i = i % 8

Augmented Assignment Operators

```
int number = 10;

// Add 5 to number
number += 5;
```

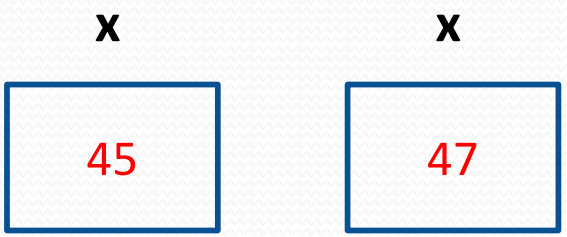
```
number = number + 5;
```




```
int x = 45;

x = x + 2;

System.out.println("x = " + x + ", x + 2 = " + (x += 2));
```

Output:

```
x = 45, x + 2 = 47
```



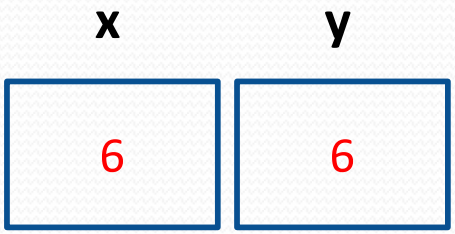
Increment and Decrement Operators

Increment Operator (++)

```
int x = 5, y;
y = ++x;
```

pre-increment operator

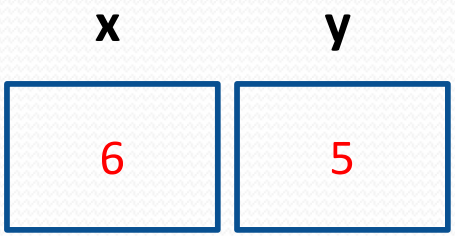
First the value of x is incremented by 1, then the new value of x (6) is assigned to y




```
int x = 5, y;
y = x++;
```

post-increment operator

First the value of x (5) is assigned to y then the value of x (5) incremented by 1 (to 6)

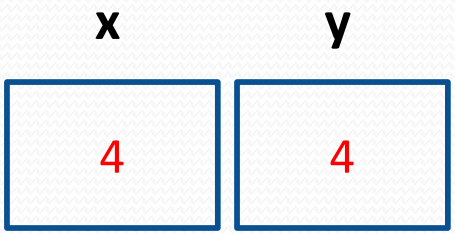


Decrement Operator (--)

```
int x = 5, y;
y = --x;
```

pre-decrement operator

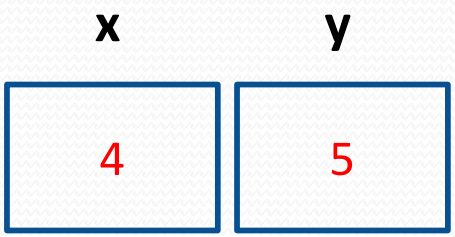
First the value of x is decremented by 1, then the new value of x (4) is assigned to y




```
int x = 5, y;
y = x--;
```

post-decrement operator

First the value of x (5) is assigned to y then the value of x (5) decremented by 1 (to 4)



Increment and Decrement Operators

```
int x = 4, y = 4;
System.out.println("x = " + ++x);
System.out.println("y = " + y++);
System.out.println("x = " + x);
System.out.println("y = " + y);
```

Output:

```
x = 5
y = 4
x = 5
y = 5
```

Some Tips

Note: The minus operator **(-)** can be used to convert a positive number to a negative number and vice versa

```
// declare 3 variables to hold integers
int num1, num2, num3;

// store the value 45 in num1
num1 = 45;

// store the value -45 in num2
num2 = -num1;

// store the value  -(-4) = +4 in num3
num3 = -(3 - 7);
```

CASTING

Difficult

Casting (1)

It is possible to store an integer value in a real variable BUT a real value CANNOT be stored in an integer variable unless it has first been converted to (cast to) an integer for example:

```
// Putting a whole number into an integer variable
int num1 = 7;                                  // OK      ☑

// trying to put a real value into an integer
// variable will cause an error at compilation
int num2 = 5.9;                                // NOT OK  ☒

// putting a whole number into a real
// variable is OK and will leave num3 = 7.0
double num3 = 7;                               // OK      ☑
```

Casting (2)

- We can change a real value into an integer by putting **the (int) command** in front of the value

```
// To convert 5.9 to 5 and store it in num2
// putting a whole number into an integer variable
int num2 = (int) 5.9;
```

- Normally it is not good practice to mix real values and integer values in the same mathematical expressions as **the result will be a real number**

Examples

```
// Declare two integer and two real variables
int num1, num2;
double num3, num4

// Assign some values
num1 = 45;
num3 = 7.0;

// Divide num1 by num3 and assign result to num2
// COMPILATION ERROR as the result is real
// and num2 is an integer
num2 = num1 / num3;                    // NOT OK   ☒

// Try to divide num1 by num3 and assign
// the result to num4 leaving num4 = 6.42587
num4 = num1 / num3;                    // OK       ☑
```

Casting (3)

- It is possible to correct the previous problem by using CASTING to specify what kind of result is required
- This is done by putting the desired data type inside brackets in front of the expression

```
// Divide num1 by num3
// Convert the result to an integer and
// assign the result to num2
// num2 will equal 6

num2 = num1 / num3;                    // NOT OK    ☒
num2 = (int) (num1/num3);              // OK        ☑
```

Casting (4)

Consider:

```
num2 = (int) num1 / num3;                    // NOT OK  ☒
```

int int double

- This is wrong (and will cause an error) as ONLY `num1` is being cast to an integer!
- `num3` **will remain real and the result will be real**
- `num2` **cannot accept a real value**

What is the result of the following?

```
num4 = 45/7;
num4 = (double) 45/7;
num4 = 45/7.0;
num4 = 45.0/7;
```

Example

```
// Declare and initialise cost
double cost = 197.55;

// Declare and initialise taxRate
double taxRate = 0.06;

// Declare a variable to store the amount payable
double taxPayable;

// Calculate tax payable
taxPayable = (int) ((cost * taxRate) * 100) / 100.0;

// Output tax payable
System.out.println("Tax Payable = £" + taxPayable);
```

Tax Rate = 6%

Output:

```
Tax Payable = £11.85
```

Questions

- What are the naming conventions for variables?
- How do you declare a variable?
- How do you initialise a variable?
- What primitive types have you encountered?
- What does the '=' operator do?
- What is the difference between '/' and '%'?
- What is meant by 'operator precedence'?
- What are the two different uses for the '+' operator?
- What do you understand about casting?