FIT ICT Software Development

Lecture 2

Lecturer : Charles Tyner

Review: Our First Program in Java: Printing a Line of Text

- Java application
  - A computer program that executes when you use the java command to launch the Java Virtual Machine (JVM).
- Sample program in Fig. 2.1 displays a line of text.

```java
// Fig. 2.1: Welcome1.java
// Text-printing program.

public class Welcome1
{
   // main method begins execution of Java application
   public static void main( String[] args )
   {
      System.out.println( "Welcome to Java Programming!" );
   } // end method main
} // end class Welcome1
```

```
Welcome to Java Programming!
```

Fig. 2.1 | Text-printing program.

Comments

- Comments

```
// Fig. 2.1: Welcome1.java
```

  - `//` indicates that the line is a comment.
  - Used to document programs and improve their readability.
  - Compiler ignores comments.
  - A comment that begins with `//` is an end-of-line comment—it terminates at the end of the line on which it appears.
- Traditional comment, can be spread over several lines as in

```
/* This is a traditional comment. It
   can be split over multiple lines */
```

  - This type of comment begins with `/*` and ends with `*/`.
  - All text between the delimiters is ignored by the compiler.

White Space

- Blank lines and space characters
  - Make programs easier to read.
  - Blank lines, spaces and tabs are known as white space (or whitespace).
  - White space is ignored by the compiler.

Review: Our First Program in Java: Printing a Line of Text (Cont.)

- Class declaration

```
public class Welcome1
```

  - Every Java program consists of at least one class that you define.
  - `class` keyword introduces a class declaration and is immediately followed by the class name.
  - Keywords (Appendix C) are reserved for use by Java and are always spelled with all lowercase letters.

Review: Our First Program in Java: Printing a Line of Text (Cont.)

- Class names
  - By convention, begin with a capital letter and capitalize the first letter of each word they include (e.g., `SampleClassName`).
  - A class name is an identifier—a series of characters consisting of letters, digits, underscores (_) and dollar signs (`$`) that does not begin with a digit and does not contain spaces.
  - Java is case sensitive—uppercase and lowercase letters are distinct—so `a1` and `A1` are different (but both valid) identifiers.

Review: Our First Program in Java: Printing a Line of Text (Cont.)

- Braces
  - A left brace, `{`, begins the body of every class declaration.
  - A corresponding right brace, `}`, must end each class declaration.
  - Code between braces should be indented.
  - This indentation is one of the spacing conventions mentioned earlier.

Review: Our First Program in Java: Printing a Line of Text (Cont.)

- Declaring the `main` Method

```
public static void main( String[] args )
```

  - Starting point of every Java application.
  - Parentheses after the identifier `main` indicate that it's a program building block called a method.
  - Java class declarations normally contain one or more methods.
  - `main` must be defined as shown; otherwise, the JVM will not execute the application.
  - Methods perform tasks and can return information when they complete their tasks.
  - Keyword `void` indicates that this method will not return any information.

Review: Our First Program in Java: Printing a Line of Text (Cont.)

- Body of the method declaration
  - Enclosed in left and right braces.
- Statement

```
System.out.println("Welcome to Java Programming!");
```

  - Instructs the computer to perform an action
    - Print the string of characters contained between the double quotation marks.
  - A string is sometimes called a character string or a string literal.
  - White-space characters in strings are not ignored by the compiler.
  - Strings cannot span multiple lines of code.

Review: Our First Program in Java: Printing a Line of Text (Cont.)

- `System.out` object
  - Standard output object.
  - Allows Java applications to display strings in the command window from which the Java application executes.
- `System.out.println` method
  - Displays (or prints) a line of text in the command window.
  - The string in the parentheses the argument to the method.
  - Positions the output cursor at the beginning of the next line in the command window.

```java
// Fig. 2.3: Welcome2.java
// Printing a line of text with multiple statements.

public class Welcome2
{
   // main method begins execution of Java application
   public static void main( String[] args )
   {
      System.out.print( "Welcome to " );
      System.out.println( "Java Programming!" );
   } // end method main
} // end class Welcome2
```

Prints `Welcome to` and leaves cursor on same line

Prints `Java Programming!` starting where the cursor was positioned previously, then outputs a newline character

```
Welcome to Java Programming!
```

Fig. 2.3 | Printing a line of text with multiple statements.

Escape Sequences

- Newline characters indicate to `System.out`'s `print` and `println` methods when to position the output cursor at the beginning of the next line in the command window.
- Newline characters are white-space characters.
- The backslash (`\`) is called an escape character.
  - Indicates a "special character"
- Backslash is combined with the next character to form an escape sequence.
- The escape sequence `\n` represents the newline character.

```java
// Fig. 2.4: Welcome3.java
// Printing multiple lines of text with a single statement.

public class Welcome3
{
   // main method begins execution of Java application
   public static void main( String[] args )
   {
      System.out.println( "Welcome\nto\nJava\nProgramming!" );
   } // end method main
} // end class Welcome3
```

Each \n moves the output cursor to the next line, where output continues

```
Welcome
to
Java
Programming!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.

`\"`	Double quote. Used to print a double-quote character. For example,
	`System.out.println( "\"in quotes\"" );`
	displays
	`"in quotes"`

Fig. 2.5 | Some common escape sequences.

Output in Java - Concatenation

Two strings can be joined together with the plus symbol (+), known as the **concatenation operator**.

```
System.out.println("Hello " + "world");
```

Outputting values on the screen

Values and **expressions** can also be printed on the screen using these output commands.

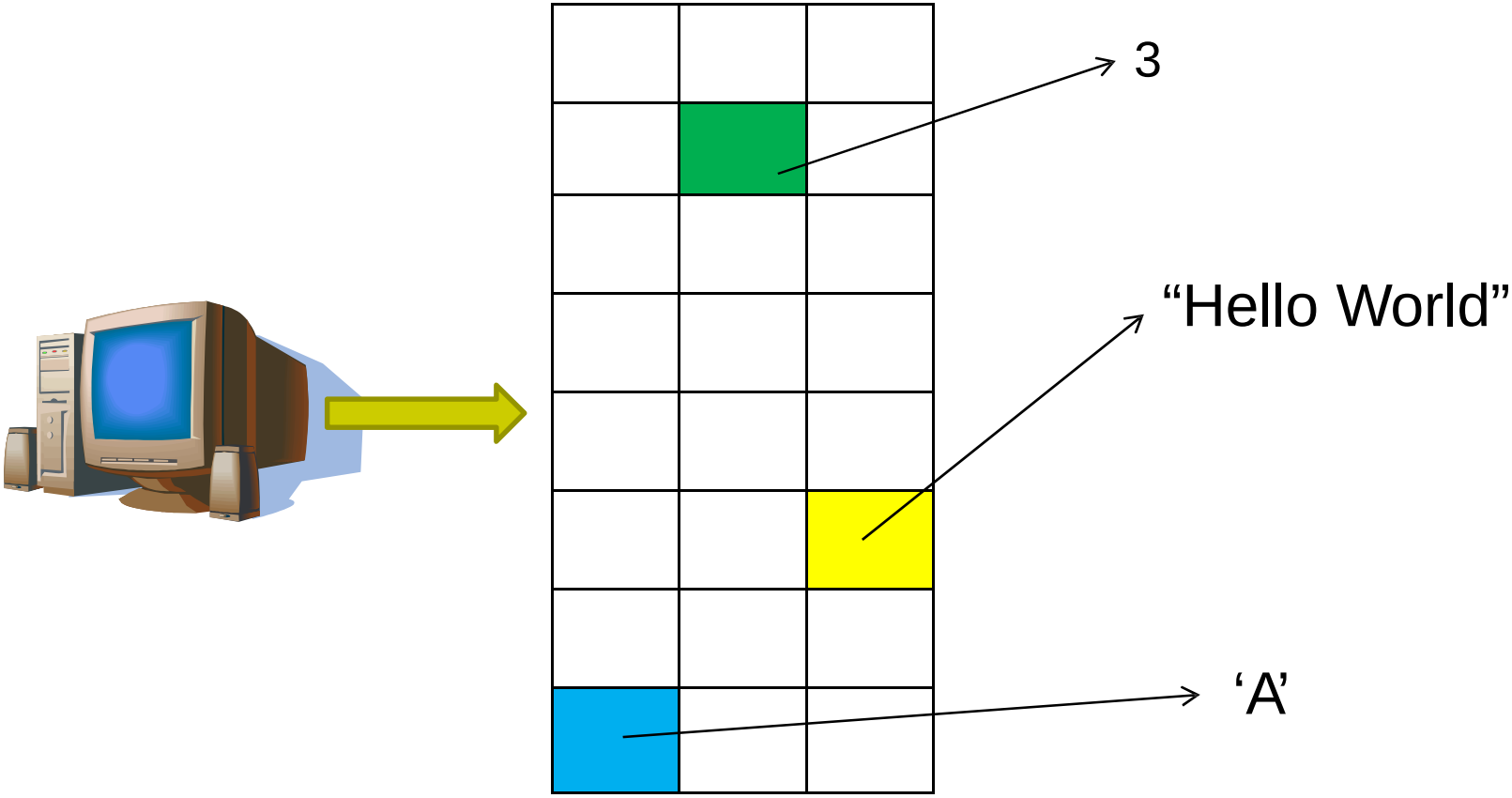
for example

30 people visiting the cinema, each charged an entrance fee of 7.50, the total cost of tickets could be displayed as follows:

```
System.out.println("cost = " + (30*7.5) );
```

Declaring Variables

How memory works



Simple data types in Java

The types of value used within a program are referred to as **data types**.

price of a cinema ticket :	**real number**
how many tickets sold :	**integer**

In Java there are a few simple data types that programmers can use.

Often referred to as the **scalar types**

The scaler types of Java

Java type	Allows for	Range of values
`byte`	very small integers	-128 to 127
`short`	small integers	-32768 to 32767
`int`	big integers	-2147483648 to 2147483647
`long`	very big integers	-9223372036854775808 to 9223372036854775807
`float`	real numbers	+/- $1.4 * 10^{-45}$ to $3.4 * 10^{38}$
`double`	very big real numbers	+/- $4.9 * 10^{-324}$ to $1.8 * 10^{308}$
`char`	characters	Unicode character set
`boolean`	true or false	not applicable

Declaring variables in Java

- The procedure of creating **named locations** in the computer's memory that will contain values while a program is running is known as declaring a variable
- these **named locations** are called **variables** because their values are allowed to *vary* over the life of the program.

To create a variable in your program you must:

- give that variable a **name** (of your choice);
- decide which **data type** in the language best reflects the kind of values you wish to store in the variable.

Naming variables

You can choose any name for variables as long as

- the name is **not already a reserved word** in the Java language (such as **class**, **void**);
- the name has **no spaces** in it;
- the name does not include operators such as + and -;
- the name starts either **with a letter**, an underscore (_), or a dollar sign ($).

The convention in Java programs is to begin the name of a variable with a ***lowercase letter***.

Choosing a suitable data type

Four Java types can be used to hold integers (**byte**, **short**, **int** and **long**).

Two Java types that can be use to hold real numbers (**double** and **float**).

The difference among these types is the range of values that they can keep, however

- the **int** type is often chosen to store integers
- the **double** type often chosen to store real numbers

The data type char is used to hold characters.

Once name and type decided upon, the variable is declared as follows:

```
dataType variableName ;
```

Declaring a variable: an example

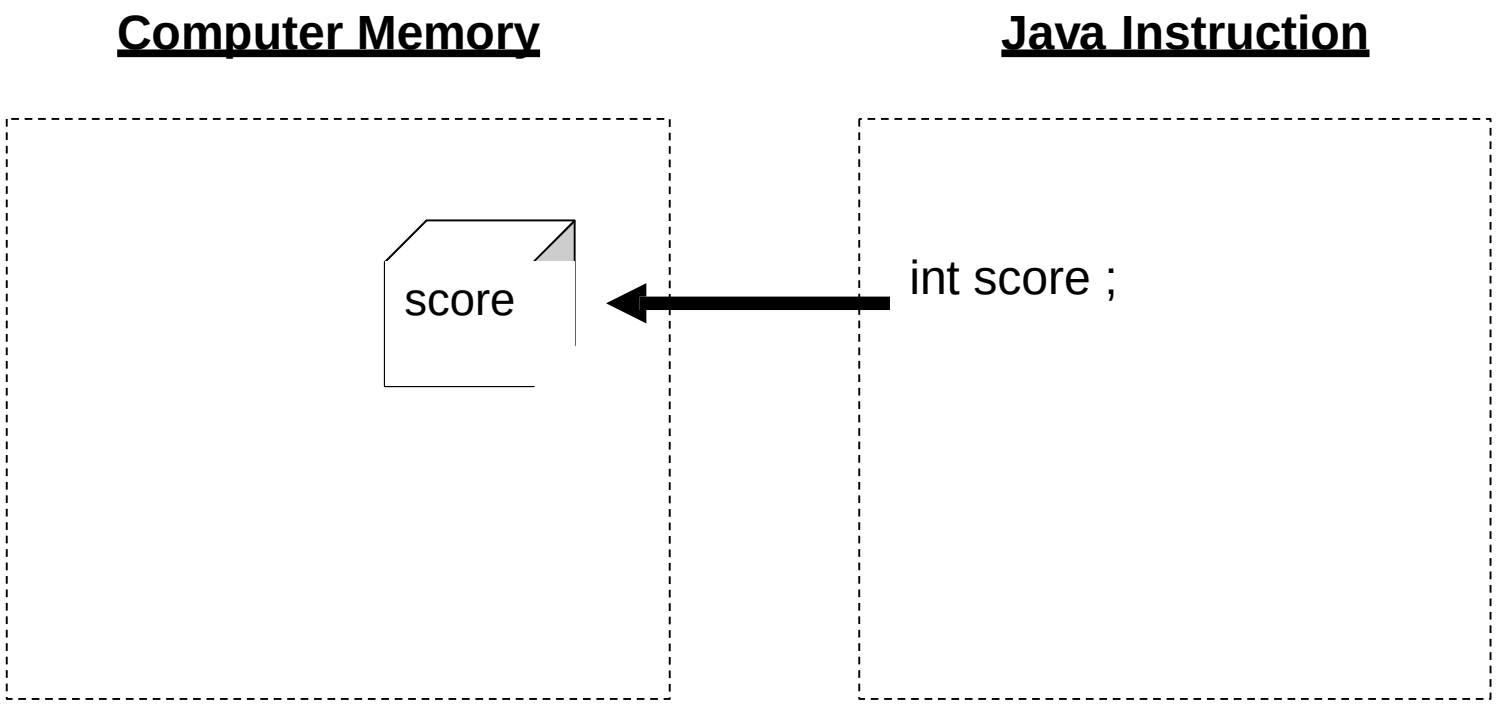
Let's create a variable to keep a player's score in a computer game.

a score will always be a whole number

good meaningful name

int score ;

The effect of declaring a variable in Java



Declaring many variables

Assume that the player of a game can choose a difficulty level (A, B, or C).

```
int score; // to hold score

char level; // to hold difficulty level
```

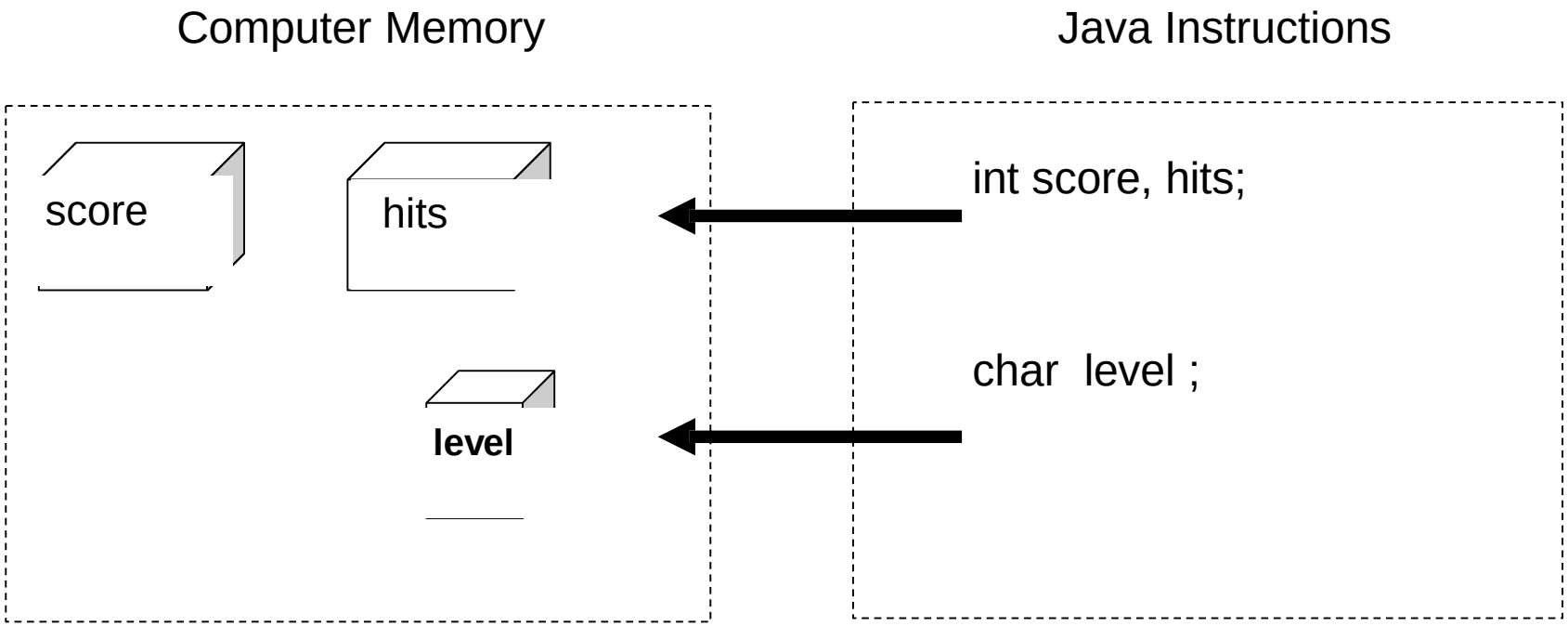
Declaring variables of the same type

Several variables can be declared on a ***single line*** if they are ***all of the same type***.

Assume that there are ghosts in the house that hit out at the player; the number of times a player gets hit by a ghost can also be recorded.

```
int score, hits; // both the same type
char level ;     // different type
```

The effect of declaring many variables in Java



Creating Constants

Constants are data items whose values *do not change*. For example:

- the maximum score in an exam (100);
- the number of hours in a day (24);
- the mathematical value of π (3.1417).

Constants are declared much like variables except

- they are preceded by the keyword **final**
- they are always initialised to their fixed value.

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

Activity: data types & assignment

- Explain which, if any, of the following would result in a compiler error:
  - int x = 75.5; // ERROR 75.5 is a double …not an int!!
  - char grade = A; //ERROR the letter A should be
    //enclosed in single quotes as follows:
    // char grade = ‘A’;

Reserved Words

High Level Languages, including Java, are usually defined in terms of a set of **reserved** words. The Java programming language is defined in terms of around 40 reserved words, which make up most of the basic command vocabulary of the language. See the full list at the link below:

Reserved words in Java

exit
break
void
if
main
case
end
system
double
do
else
(etc…)

Note:
These reserved words cannot be redefined for other uses: i.e. they may not be used as for variables names.

Assignments in Java

Assignments in Java

Assignments allow **values to be put** into **variables**.

Written in Java with the use of the equality symbol (=), known as **the assignment operator**.

Simple assignments take the following form:

variableName = value;

```
score = 0;
```

Initialising variables

You may **combine** the **assignment statement** with a **variable declaration** to put an initial value into a variable as follows:

```
int score = 0;
```

Note, the following declaration will **not** compile in Java:

```
int score = 2.5 ;
```

WHY???? This will not compile because 2.5 is a **double** value

Putting values into character variables

When assigning a value to a character variable, you must **enclose the value in single quotes**.

<u>for example</u>

set the initial difficulty level to A

```
char level = ‘A’;
```

Assigning a Variable

Once the declaration is made with a legal name (identifier), operations can be performed on the variable, e.g.

```
int age; age  =  21;
System.out.print(age);
```

This command displays the Integer age to the screen

Re-assigning variables

Remember: you need to declare a variable only once.

You can then assign to it as many times as you like.

```
char level = ‘A’;
level = ‘B’;
```

Re-assigning variables

Whenever a value is placed in a memory location, this value replaces the previous value in that location.

The previous value is therefore destroyed!

Memory locations showing the name and value of variables:

```
char level = 'A';
 level = 'B';
```

Starts out as:

level | A |

Then changes to:

level | B |

Input from the Keyboard

Input from keyboard and adding integers

- Integers
  - Whole numbers, like –22, 7, 0 and 1024)
- Programs remember numbers and other data in the computer's memory and access that data through program elements called variables.
- The program of Fig. 2.7 demonstrates these concepts.

```java
// Fig. 2.7: Addition.java
// Addition program that displays the sum of two numbers.
import java.util.Scanner; // program uses class Scanner

public class Addition
{
   // main method begins execution of Java application
   public static void main( String[] args )
   {
      // create a Scanner to obtain input from the command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

      System.out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second number from user

      sum = number1 + number2; // add numbers, then store total in sum
```

Imports class **Scanner** for use in this program

Creates **Scanner** for reading data from the user

Variables that are declared but not initialized

Reads an **int** value from the user

Reads another **int** value from the user

Sums the values of **number1** and **number2**

Fig. 2.7 | Addition program that displays the sum of two numbers. (Part 1 of 2.)

```
        System.out.printf( "Sum is %d\n", sum ); // display sum
    } // end method main
} // end class Addition
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Fig. 2.7 | Addition program that displays the sum of two numbers. (Part 2 of 2.)

Input from keyboard and adding integers

- `import` declaration
  - Helps the compiler locate a class that is used in this program.
  - Rich set of predefined classes that you can reuse rather than “reinventing the wheel.”
  - Classes are grouped into packages—named groups of related classes—and are collectively referred to as the Java class library, or the Java Application Programming Interface (Java API).
  - You use `import` declarations to identify the predefined classes used in a Java program.

Input from keyboard and adding integers

- Variable declaration statement

```
Scanner input = new Scanner( System.in );
```

  - Specifies the name (`input`) and type (`Scanner`) of a variable that is used in this program.
- Variable
  - A location in the computer's memory where a value can be stored for use later in a program.
  - Must be declared with a name and a type before they can be used.
  - A variable's name enables the program to access the value of the variable in memory.
  - The name can be any valid identifier.
  - A variable's type specifies what kind of information is stored at that location in memory.

Input from keyboard and adding integers

- `Scanner`
  - Enables a program to read data for use in a program.
  - Data can come from many sources, such as the user at the keyboard or a file on disk.
  - Before using a `Scanner`, you must create it and specify the source of the data.
- The equals sign (=) in a declaration indicates that the variable should be initialised (i.e., prepared for use in the program) with the result of the expression to the right of the equals sign.
- The `new` keyword creates an object.
- Standard input object, `System.in`, enables applications to read bytes of information typed by the user.
- `Scanner` object translates these bytes into types that can be used in a program.

Input from keyboard and adding integers

- Variable declaration statements

```
int number1; // first number to add
int number2; // second number to add
int sum; // sum of number1 and number2
```

  declare that variables `number1`, `number2` and `sum` hold data of type `int`

  - They can hold integer.
  - Range of values for an `int` is –2,147,483,648 to +2,147,483,647.
  - Actual `int` values may not contain commas.

- Several variables of the same type may be declared in one declaration with the variable names separated by commas.

Scanner class Numeric and String methods

Method	***Returns***
`int nextInt()`	Returns the next token as an `int`. If the next token is not an integer, `InputMismatchException` is thrown.
`long nextLong()`	Returns the next token as a `long`. If the next token is not an integer, `InputMismatchException` is thrown.
`float nextFloat()`	Returns the next token as a `float`. If the next token is not a float or is out of range, `InputMismatchException` is thrown.
`double nextDouble()`	Returns the next token as a `long`. If the next token is not a float or is out of range, `InputMismatchException` is thrown.
`String next()`	Finds and returns the next complete token from this scanner and returns it as a string; a token is usually ended by whitespace such as a blank or line break. If not token exists, `NoSuchElementException` is thrown.
`String nextLine()`	Returns the rest of the current line, excluding any line separator at the end.
`void close()`	Closes the scanner.

Input from keyboard and adding integers

- Prompt
  - Output statement that directs the user to take a specific action.
- `System` is a class.
  - Part of package `java.lang`.
  - Class `System` is not imported with an `import` declaration at the beginning of the program.

Input from keyboard and adding integers

- `Scanner` method `nextInt`

```
number1 = input.nextInt(); // read first number from user
```

  - Obtains an integer from the user at the keyboard.
  - Program waits for the user to type the number and press the Enter key to submit the number to the program.
- The result of the call to method `nextInt` is placed in variable `number1` by using the assignment operator, `=`.
  - "`number1` gets the value of `input.nextInt()`."
  - Operator = is called a binary operator—it has two operands.
  - Everything to the right of the assignment operator, =, is always evaluated before the assignment is performed.

Input from keyboard and adding integers

- Arithmetic

```
sum = number1 + number2; // add numbers
```

  - Assignment statement that calculates the sum of the variables `number1` and `number2` then assigns the result to variable `sum` by using the assignment operator, `=`.
  - "`sum` gets the value of `number1` + `number2`."
  - In general, calculations are performed in assignment statements.
  - Portions of statements that contain calculations are called expressions.

Input from keyboard and adding integers

- Integer formatted output

```
System.out.printf( "Sum is %d\n", sum );
```

  - Format specifier `%d` is a placeholder for an `int` value
  - The letter `d` stands for “decimal integer.”

Displaying Text with `printf`

- `System.out.printf` method
  - `f` means "formatted"
  - displays formatted data
- Multiple method arguments are placed in a comma-separated list.
- Java allows large statements to be split over many lines.
  - Cannot split a statement in the middle of an identifier or string.
- Method `printf`'s first argument is a format string
  - May consist of fixed text and format specifiers.
  - Fixed text is output as it would be by `print` or `println`.
  - Each format specifier is a placeholder for a value and specifies the type of data to output.
- Format specifiers begin with a percent sign (%) and are followed by a character that represents the data type.
- Format specifier `%s` is a placeholder for a string.

```java
// Fig. 2.6: Welcome4.java
// Displaying multiple lines with method System.out.printf.

public class Welcome4
{
   // main method begins execution of Java application
   public static void main( String[] args )
   {
      System.out.printf( "%s\n%s\n",
         "Welcome to", "Java Programming!" );
   } // end method main
} // end class Welcome4
```

Each %s is a placeholder for a String that comes later in the argument list

Statements can be split over multiple lines.

```
Welcome to
Java Programming!
```

Fig. 2.6 | Displaying multiple lines with method `System.out.printf`.

Memory Concepts

number1	45

Fig. 2.8 | Memory location showing the name and value of variable `number1`.

number1	45
number2	72

Fig. 2.9 | Memory locations after storing values for `number1` and `number2`.

number1	45
number2	72
sum	117

Fig. 2.10 | Memory locations after storing the sum of `number1` and `number2`.

Activity: reading from the keyboard

- Consider the following declaration of a variable to store students examination marks:

```
double mark;
```

- Which one of the following statements allows the user to enter a mark of 70 into this variable while the program is running?
  a) mark = 70;
  b) mark = System.out.print("Please enter a mark");
  c) mark = input.nextDouble();
  d) mark = input.nextInt();

Arithmetic Operators

Arithmetic Operators

Java has the **four familiar arithmetic operators**, plus a remainder operator for this purpose.

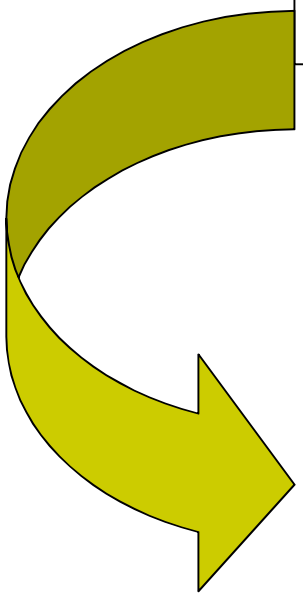
The arithmetic operators of Java	
Operation	Java operator
addition	+
subtraction	-
multiplication	*
division	/
remainder	%

Arithmetic Operators - Example

You can use the arithmetic operators in assignment statements much like you might use a calculator.

For Example

```
int x;
x = 10 + 25;
```



Computer Memory

X

35

The Modulus operator

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

Examples of the modulus operator in Java	
Expression	Value
29 % 9	2
36 % 5	1
6 % 8	6
40 % 40	0
10 % 2	0

Modulus Operator: an example

A large party of 30 people go to visit the roller coaster rides at a local theme park.

When they reach the end of the queue, they are told that only groups of four can get on!

To calculate how many groups in the party can get on the ride and how many people in the party will have to wait their turn, the **division** and **modulus operators** can be used as follows:

```
int takeTurn, missTurn;
takeTurn = 30/4; //calculates the number of groups
missTurn = 30%4;  //calculates the people that wait
```

Modulus operator: an example

After the previous instructions, the value of **takeTurn** will be **7** (the result of dividing 30 by 4).

The value of **missTurn** will be **2**.
(the remainder after dividing 30 by 4).

Arithmetic Expressions

They must be written in a straight -line format to facilitate entering programs into the computer.

- This means that expressions such as "a divided by b" must be written as a / b so that all constants, variables, and operators appear in a straight line.
- Parentheses are used in Java expressions in the same manner as in algebraic expressions.
- For example , to multiply a times b + c we write:

```
a * (b + c)
```

Rules of Operator Precedence

Rules of Operator Precedence

Java applies the operators in arithmetic expressions in a precise sequence determined by the rules of operator precedence.

- Operators in expressions contained within pairs of parentheses are evaluated first.
  This can therefore be used as a mechanism to force an order or evaluation in your program.

- In cases of nested or embedded parentheses, the operators in the innermost pair are applied first.

- Multiplication, division and modulus operations are applied next, from left to right.

- Addition and subtraction operations are applied last from left to right.

Precedence of Arithmetic Operators

Operator(s)	Operation(s)	Order of evaluation (precedence)
`( )`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*, /, or %`	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
`+ or -`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Worked Example

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;`

`2 * 5 is 10` (Leftmost multiplication)

Step 2. `y = 10 * 5 + 3 * 5 + 7;`

`10 * 5 is 50` (Leftmost multiplication)

Step 3. `y = 50 + 3 * 5 + 7;`

`3 * 5 is 15` (Multiplication before addition)

Step 4. `y = 50 + 15 + 7;`

`50 + 15 is 65` (Leftmost addition)

Step 5. `y = 65 + 7;`

`65 + 7 is 72` (Last addition)

Step 6. `y = 72;` (Last operation—assignment)

Operator Precedence

We stated that:

- Multiplication, division and modulus operations are applied, from left to right.
- Addition and subtraction operations are applied last, from left to right

This often creates confusion :

Multiplication, division and modulus are applied before addition and subtraction.

This is not stating that multiplication happens before division. The sequence is applied from left to right when determining whether to apply items at the same level.

Example a): 8 / 2 * 3 First 8 / 2 which results in 4
Then 4 * 3 which results in 12

Similarly , Example b): 3 – 1 + 2 First subtract 1 from 3 and then add 2

Summary of Operator Precedence

First

Parentheses

followed by

Multiplication, Division and Modulus *Left to Right*

Then

Addition and Subtraction *Left to Right*

Expressions

Expressions in Java

Right-hand side of an assignment statement can itself contain variable names.

```
double price, tax, cost;
price = 500;
tax = 17.5;
cost = price * (1 + tax/100) ;
```

Value is 500

Value is 17.5

Value is 587.5

More expressions

Nothing to stop you using the name of the variable you are assigning to in the expression itself.

```
price = price * (1 + tax/100) ;
```

The new value of ‘price’

the old value of ‘price’

Documentation

Documentation is a set of text notes that appear within a program.

These notes include:

- A description of how to use the program (User's Guide).
- The purpose of the program….how it is installed, tested and expected results.
- Other important information to be included is the name of the person(s) who wrote the program and the date on which it was written and any special or user defined methods used

Importance of Documentation

NOTE: The importance of **adequate** documentation cannot be over emphasised.

It is also important not to OVER DOCUMENT ! !

Guide to Requirements

```
/* ***********************************************/
/*   Java Program to add two numbers.            */
/*   Written by:        ...................      */
/*  Course             :...................      */
/*  Language           :...................      */
/*  Date Started       :...................      */
/*  Last Update        :...................      */
/*  Program Description        :...............  */
/*  ...................................          */
/*  ...................................*/
/***********************************************/
Class MyProg{
                    Etc….
```

Indentation

Programs should be indented consistently;

ie: the number of spacings to the right for each command must be appropriate and the same for each similar section of code.

Note: Algorithms

- Any computing problem can be solved by executing a series of actions in a specific order.
- An algorithm is a procedure for solving a problem in terms of
  - the actions to execute and
  - the order in which these actions execute
- The "rise-and-shine algorithm" followed by one executive for getting out of bed and going to work:
  - (1) Get out of bed; (2) take off pajamas; (3) take a shower; (4) get dressed; (5) eat breakfast; (6) carpool to work.
- Suppose that the same steps are performed in a slightly different order:
  - (1) Get out of bed; (2) take off pajamas; (3) get dressed; (4) take a shower; (5) eat breakfast; (6) carpool to work.
- Specifying the order in which statements (actions) execute in a program is called program control.