

# Chapter 1

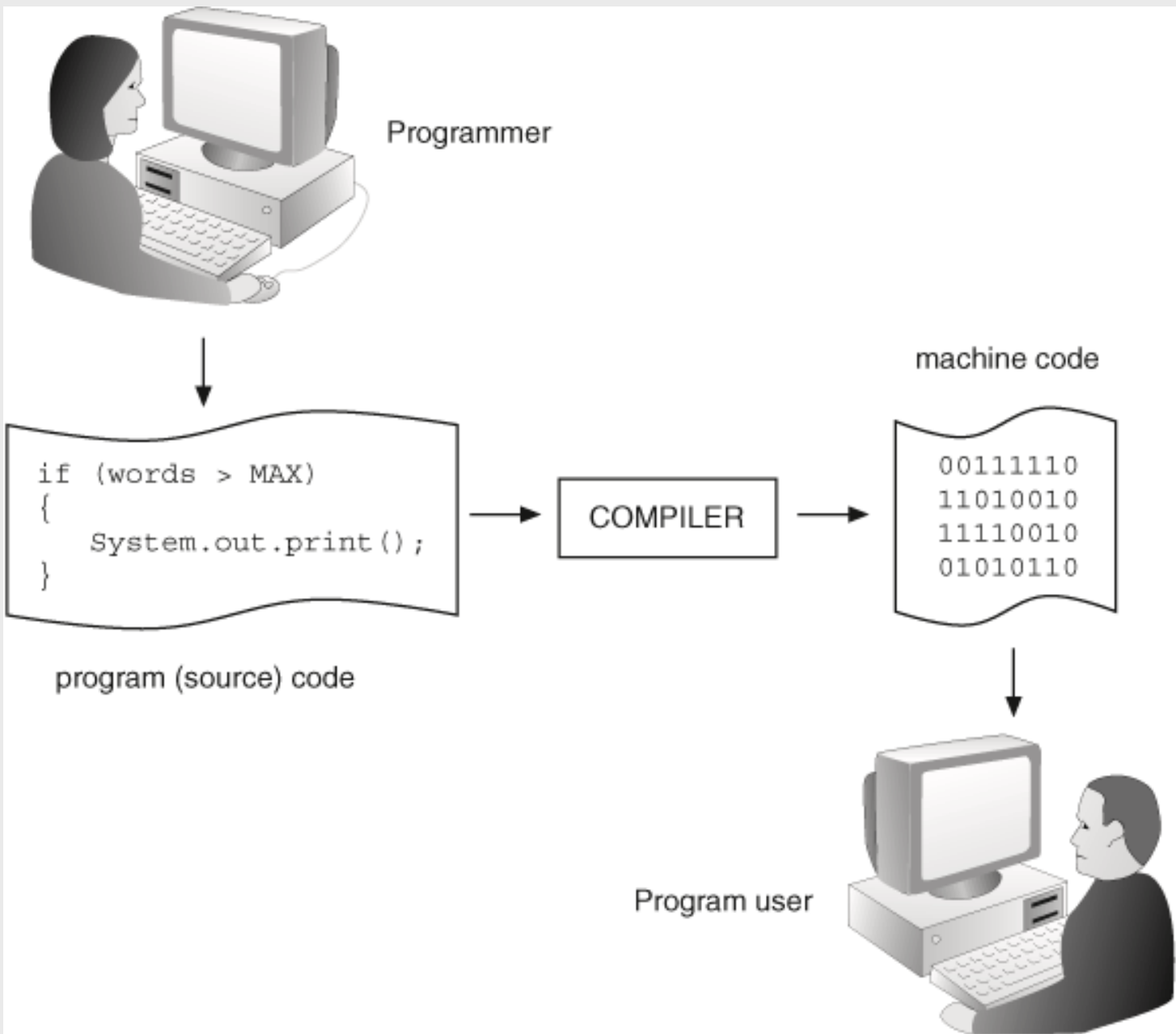
## The first step

# Software

- the set of instructions that tells a computer what to do is called a **program**;
- **software** is the name given to a single program or a set of programs.
- **application software** is the name given to useful programs that a user might need (such as word-processors and games).
- **system software** is the name given to special programs that help the computer to do its job (such as operating systems and network software)
- **programming** is the task of writing instructions for the computer;
- these instructions have to be written in a special programming language.

# Compiling programs

- modern programming languages like Java consist of instructions that look a bit like English;
- the set of instructions written in a programming language is called the **program code** or **source code**.
- these instructions have to be translated into **binary** instructions (i.e. 0's and 1's);
- the language of the computer is often referred to as **machine code**;
- a special piece of system software called a **compiler** translates source code to machine code.

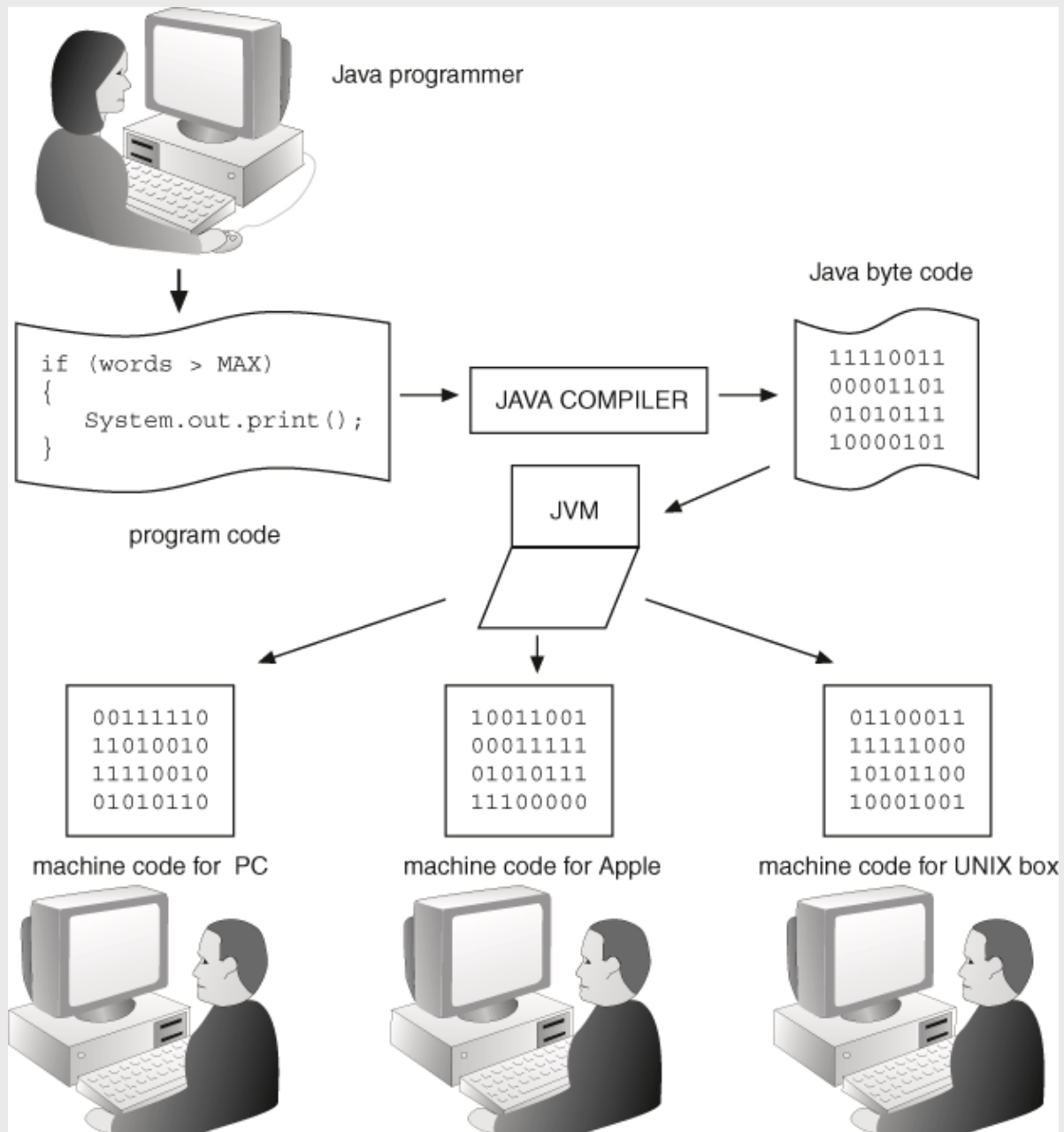


# Programming in Java

- Java is **platform-independent**;
- Java can work within the World Wide Web of computers via **browsers**, such as *Netscape* and *Internet Explorer*;
- Java programs that run on the Web are called **applets**;
- inside a modern browser is a special program, called a **Java Virtual Machine (JVM)**;
- the JVM is able to run a Java program for the particular computer on which it is running.

# How can it do this?

- Java compilers do not translate the program into machine code, but into special instructions called **Java byte code**;
- Java Byte Code (which still consists of 0's and 1's), contains instructions that are exactly the same irrespective of any computer;
- Java Byte Code is *universal*, whereas machine code is specific to a particular type of computer;
- the job of the JVM is to translate each instruction for the computer it is running on, before the instruction is performed.



# Your first program

## Program 1.1

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello world");
    }
}
```

when your program runs you will see the words "Hello world" displayed.



# Java program structure

- in the Java programming language:
  - A program is made up of one or more **classes**
  - A class contains one or more **methods**
  - A method contains program **statements**
- a Java application always contains a method called **main**

# Java program structure

```
// comments about the class
```

```
public class MyProgram
```

```
{
```

class header



class body




Comments can be placed almost anywhere

```
}
```

# Java program structure

```
// comments about the class
public class MyProgram
{
    // comments about the method
    public static void main (String[] args)
    {
    }
}

method body
method header
```



# Adding comments to a program

## Program 1.1 - with added comments

```
// this is a short' comment, so we use the first method

public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello world");
    }

    /* this is the second method of including comments - it is more
       convenient to use this method here, because the comment is
       longer and goes over more than one line */
}
```

# Simple data types in Java

- What types of data does Java allow us to store in our programs?

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

# Declaring variables in Java

- the above data types are used in programs to create named locations in the computer's memory;
- these will contain values while a program is running;
- this process is known as **declaring**;
- these named locations are called **variables**.
- a variable name must be a valid **identifier**.

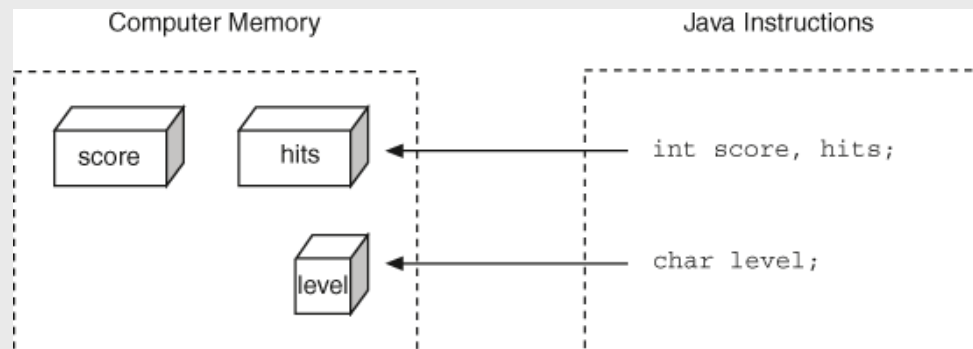
# A variable is declared as follows:

```
dataType variableName ;
```

```
int score;  
char level;
```

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

```
int score, hits; // two variables declared at once  
char level ;    // this has to be declared separately
```



# Identifiers

- **identifiers** are the words a programmer uses in a program
- an identifier can be made up of letters, digits, the underscore character ( `_` ), and the dollar sign ( `$` )
- identifiers cannot begin with a digit
- Java is *case sensitive* - **Total**, **total**, and **TOTAL** are different identifiers
- by convention, Java programmers use different case styles for different types of identifiers, such as
  - lower case for variables – **level**
  - title case for class names – **Hello**
  - upper case for constants – **MAXIMUM**



# Identifiers

- sometimes we choose identifiers ourselves when writing a program (such as `score`, `hits`, `level`)
- sometimes we are using another programmer's code, so we use the identifiers that they chose (such as `println`)
- often we use special identifiers called **reserved words** (such as `class`) that already have a predefined meaning
- a reserved word cannot be used in any other way

# Reserved words

- Java reserved words:

abstract	else	interface	super
boolean	extends	long	switch
break	false	native	synchronized
byte	final	new	this
case	finally	null	throw
catch	float	package	throws
char	for	private	transient
class	goto	protected	true
const	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double	int	strictfp	

# Assignments in Java

- assignments allow values to be put into variables;
- they are written in Java with the use of the equality symbol (=);
- this symbol is known as **the assignment operator**;
- simple assignments take the following form:  
variableName = value;

*For example*

```
score = 0;
```

- you can combine the assignment statement with a variable declaration to put an initial value into a variable:

```
int score = 0;
```

- this is equivalent to:

```
int score;  
score = 0;
```

- when assigning a value to a character variable, you must enclose the value in single quotes:

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

- Each variable should be declared only once. You can then assign values to it as many times as you like

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

- The following declaration will not compile

```
int score = 2.5;
```

- On the other hand, the following is legal

```
double someNumber = 1000;
```

# Creating constants

- there are occasions where data items in a program have values *that do not change*.
- values that remain constant throughout a program (as opposed to variable) should be named and declared as **constants**;
- constants are declared much like variables in Java except that they are preceded by the keyword **final**, and are always initialized to their fixed value.

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

# Arithmetic operators

## The arithmetic operators of Java

Operation	Java operator
addition	+
subtraction	-
multiplication	*
division	/
remainder	%

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

terms on the right-hand side of assignment operators (like 10 + 25) are referred to as **expressions**.

```
int R1, R2;  
double R3, R4;
```

```
R1 = 30 / 4;           // yields 7  
R2 = 30 / 4.0;       // compilation error  
R3 = 30 / 4;         // yields 7.0  
R4 = 30 / 4.0;      // yields 7.5
```



# Order of evaluation

- brackets
- division / multiplication
- addition / subtraction

# Expressions in Java

- the expression on the right-hand side of an assignment statement can itself contain variable names;
- if this is the case then the name does not refer to *the location*, but to *the contents of the location*.

```
double price, tax, cost;           // declare three variables
price = 500;                       // set price
tax = 17.5;                        // set tax rate
cost = price * (1 + tax/100);      // calculate cost
```

# You can use the name of the variable you are assigning to in the expression itself

this means that the old value of the variable is being used to calculate its new value.

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

the new value of  
price

the old value of  
price

# A special shorthand

- if a variable  $x$  has been declared as an **int**, then the instruction for incrementing  $x$  would be:

$$x = x + 1;$$

- this is so common that there is a special shorthand for this instruction:

$$x++;$$

- the '++' is therefore known as the increment operator;
- similarly there exists a decrement operator, '--':

$$x--;$$

is shorthand for:

$$x = x - 1;$$

```
int X = 2, Y;  
Y = X++;           // after: X = 3 and Y = 2
```

```
int X = 2, Y;  
Y = ++X;          // after: X = 3 and Y = 3
```

```
int X = 2, Y = 5, Z = 4;  
Y += X;           // after: X = 2 and Y = 7  
Z *= X;           // after: X = 2 and Z = 8
```

```
Y += X; is shorthand for Y = Y + X;  
Z *= X; is shorthand for Z = Z * X;
```

# Output in Java

To display a message in Java:

```
System.out.println(message to be printed  
on screen);
```

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

- println is short for print line and the effect of this statement is to start a new line after displaying whatever is in the brackets.
- there is an alternative form of the System.out.print statement, which uses System.out.print.

### Program 1.1 - with an additional line

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello world"); // notice the use of println
        System.out.println("Hello world again!");
    }
}
```

- When we run this program, the output will be:

Hello world

Hello world again!

# Changing println to print:

**Program 1.1 - to show the effect of using *print* instead of *println***

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.print("Hello world"); // notice the use of println
        System.out.println("Hello world again!");
    }
}
```

- When we run this program, the output will be:

Hello worldHello world again!

- To have a blank line, use println with empty brackets



# Introducing *strings*

- collections of characters such as "Hello world" are called **strings**;
- in Java, literal strings like this are always enclosed in speech marks;
- two strings can be joined together with the plus symbol (+);
- when using this symbol for this purpose it is known as the **concatenation operator**.

```
System.out.print("Hello " + "World");
```

Java also allows any values or expressions of the simple types to be displayed using these output commands.

**For example**

```
System.out.print(10*10);
```

This instruction prints the number 100 on the screen

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

This displays:

```
cost = 225.0
```

### Program 1.3

```
/* a program to calculate and display the cost of a product  
after sales tax has been added */
```

```
public class FindCost2  
{  
    public static void main(String[] args)  
    {  
        double price, tax;  
        price = 500;  
        tax = 17.5;  
        price = price * (1 + tax/100); // calculate cost  
  
        // display results and pause before closing  
        System.out.println("*** Product Price Check ***");  
        System.out.println("Cost after tax = " + price);  
    }  
}
```

This program produces the following output:

```
*** Product Price Check ***
```

```
Cost after tax = 587.5
```

# Input in Java: the *Scanner* class

- in order to use the Scanner class, we have to import the util package
  - package is a collection of pre-compiled classes
  - for that, we have to place the following line at the beginning of our program:

```
import java.util.*;
```

- you can use all the input methods that have been defined in this class.

# Input in Java: the *Scanner* class

- the following instruction declares an **object** of the **Scanner class** (more about it in chapters 6 and 7)
  - in Java, `System.in` represent the keyboard
  - by associating our `Scanner` object with `System.in`, we are telling it to get the input from the keyboard

```
Scanner sc = new Scanner(System.in)
```

# Input methods of the Scanner class

- if we want a user to type in an integer at the keyboard:

```
x = sc.nextInt();
```

- in the case of a **double**, y:

```
y = sc.nextDouble();
```

- in the case of a char, c:

```
c = sc.next().charAt(0);
```

# Example

- Write a program that displays the price of a product after its tax has been added. The initial price of a product as well as the tax rate will be specified by the user.

```
import java.util.*;

public class computePrice{

    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        double price, tax;

        System.out.print("Enter initial price: ");
        price = sc.nextDouble();
        System.out.print("Enter tax rate: ");
        tax = sc.nextDouble();
        price *= 1 + tax / 100;
        System.out.println("After tax: " + price);
    }
}
```



# Sample runs

- Enter initial price: 1000  
Enter tax rate: 12.5  
After tax: 1125.0
  
- Enter initial price: 1000  
Enter tax rate: cs111  
Exception in thread "main" java.util.InputMismatchException  
at java.util.Scanner.throwFor(Scanner.java:819)  
at java.util.Scanner.next(Scanner.java:1431)  
at java.util.Scanner.nextDouble(Scanner.java:2335)  
at computePrice.main(computePrice.java:12)

# More about strings

- a String is not a simple data type like an int or a char;
- a String is a class - you learn about this in the second semester
- however, you can declare a String in a similar way to the way you do it with variables such as **int** or **char**;
- notice that String "type" has to start with a capital "S".

```
String name;
```

- Java allows you to use the normal assignment operator ( = ) with strings:

```
name = "Quentin";
```

- to obtain a string from the keyboard you can use the next method of Scanner.
  - **note:** your string should not contain spaces; you will see a way around this in chapter 6.

# Example

- Write a program that displays a greeting message to a user at the time of the execution starts.

```
import java.util.*;

public class Hello{

    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String name;

        System.out.print("What is your name? ");
        name = sc.next();
        System.out.println("Hello " + name);
    }
}
```

# Sample runs

- What is your name? Pinar  
Hello Pinar
- What is your name? Pinar Duygulu  
Hello Pinar
- What is your name? 1234.67  
Hello 1234.67