Computer Science 101 S1

Lecture 4

Contents

Java has eight primitive types
Storing information - variables
The assignment statement
Expressions
Modulus
Increment, decrement operators

Coursebook: §5, §6

Review

What is the output of the following programs?

```
public class MyProgram {
 public void start() {
      System.out.print("abc\ndef");
      System.out.println("\"");
 }
}
```

```
abc
def"
```

```
public class MyProgram {
 public void start() {
      System.out.println("\\n\n\n\\n");
 }
}
```

```
\n

\n
```


Variables

A *variable* is a section of memory that we can use to store some piece of information:

every variable has a name (identifier)
each variable only capable of holding one *type* of thing
each variable holds a single value at a time

Variables

Variable Analogy

A variable is like a box with a label.
The box is shaped so that only one kind of thing fits in it.
The box is able to hold one thing at a time.

Declaring a variable

We must *declare* a variable before we can use it to store a value:

- the declaration specifies what **type** of data the variable will store
- it also specifies the **name** of the variable

For example:

```
int wholeNum;
int number;
double costPrice, rate, sellPrice;
```

7

Type of a primitive variable

All variables have a *type*. In Java there are eight primitive types : **`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**, **`char`**, **`boolean`**

Integers (whole valued numbers): **`byte`**, **`short`**, **`int`**, **`long`**

Floating-point numbers (numbers with fractions): **`float`**, **`double`**

Characters (symbol in character set (text)): **`char`**

Boolean (true or false values): **`boolean`**

8

Range of values allowed

Values stored by integer and floating point types:

Type	*Size*	*Minimum Value*	*Maximum Value*
byte	8 bits	-128	127
short	16 bits	-32 768	32 767
int	32 bits	-2 147 483 648	2 147 483 647
long	64 bits	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
float	32 bits	+/- 1.4 E-45	+/- 3.4 E+38
double	64 bits	+/-4.9 E-324	+/- 1.8 E+308

Note: float and double use scientific notation

9

Variable identifiers

Use the following style guidelines for variable identifiers:

- all variable identifiers should begin with a lower case letter
- any subsequent words should start with a capital letter
- all other letters should be lower case
- use sensible names that inform the programmer what the variable is used for

Examples

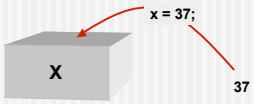
```
int numberOfStudents;
long dollarsOwnedByBillGates;
double percentagePassing101,
                  percentagePassing105;
boolean isTheLectureOverYet;
```

10

Assignment statements

The assignment operator (**=**) is used to put a value into a variable e.g.

```
int x;
x = 37;
```



Assigns a value to the variable i.e. puts a new value into the box identified by the variable name

Both the variable and the value must be compatible types

The variable can *only hold one value* at a time – assigning a new value to it will *replace* any previous value

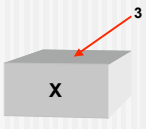
11

Dynamic initialisation

We can assign a value to a variable at the same time that we declare it.

So instead of:

```
int x;
x = 3;
```



we can simply have:

```
int x = 3;
```

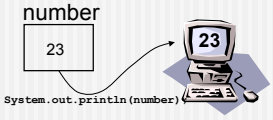
12

Printing variables

A variable can be used anywhere that a literal can be used. We can print out the value stored in a variable using the `System.out.println()` statement. For example

```
int number = 23;
System.out.println(number);
```

will print out the value 23.



13

Literal values

```
/*
  * Cost of call
*/
public class Q1Program {
 public void start() {
      ...
      ...   1.5    ...
      ...
      ...   ...   1.5
      ...
      1.5    ...   ...
 }
}
```

Avoid repeatedly using literal values if possible

14

Use a variable - not ideal

```
/*
  * Cost of call
*/
public class Q1Program {
 public void start() {
      double costPerTwentyMinutes = 1.5;
      ... costPerTwentyMinutes ...
      ...
      ...   ... costPerTwentyMinutes
      ...
      costPerTwentyMinutes...  ...
 }
}
```

15

Symbolic constants

```
/*
  * Cost of call
*/
public class Q1Program {
 public void start() {
      final double PER_TWENTY_MINUTES = 1.5;
      ...  PER_TWENTY_MINUTES ...
      ...
      ...   ... PER_TWENTY_MINUTES
      ...
      PER_TWENTY_MINUTES ...  ...
 }
}
```

16

Symbolic constants

A *symbolic constant* is like a variable, except that the value it stores cannot change once it is initialised.

For example:

```
final int DAYS_IN_YEAR = 365;
final double GST = 0.125;
```

Use the following style guidelines for naming symbolic constants:

- use all upper case letters
- separate multiple words with an underscore

17

Assignment – symbolic constants

Cost of the call

```
final double PER_TWENTY_MINUTES = 1.5;
final double PER_FIVE_MINUTES = 0.5;
final double PER_SINGLE_MINUTE = 0.12;
final int TWENTY_MINUTES = 20;
final int FIVE_MINUTES = 5;
```

18

Expressions

An *expression* is code that can be evaluated to give a single value:

- a literal value is an expression
- a variable is an expression
- an arithmetic formula is an expression

Examples of Expressions

```
35;
age;
3 * number + 56;
age / value – (value * (number - age));
Math.random();
```

19

Expressions

Whenever we perform an assignment using the assignment operator (=)

- the name of the variable is on the left of the = sign
- an *expression* is on the right of the = sign

Example

```
int num1, num2;

num1 = 130;
num2 = num1;
```

num1 | 130

num2 | 130

20

Arithmetic operators

Normal arithmetic: `+` `-` `*` `/`

Modulus - Remainder after a division: `%`

Increment, Decrement - Increase or decrease by one: `--` `++`

21

Operators and type

Operators apply to variables of the same type. The result is the same type as the variables (operands)

Example

```
2 + 3        <-- Result is an integer
2.0 + 3.0    <-- Result is a floating point number
```

22

Operators and type

Division

Result of dividing two integers must be an integer

Truncation is used i.e. fractional part is lost

Example

```
10 / 4    <-- Result is 2
10 / 3    <-- Result is 3
```

23

Modulus

Modulus means the remainder after a division (Integer value)

Examples

10 divided by 5 equals 2 with no remainder

10 divided by 4 equals 2 with 2 remainder

11 divided by 4 equals 2 with 3 remainder

```
Result is 2 <-- 10 / 5      10 % 5 --> Result is 0
Result is 2 <-- 10 / 4      10 % 4 --> Result is 2
Result is 2 <-- 11 / 4      11 % 4 --> Result is 3
```

Example

```
3207
3.5
9.0
```

```
public class Operations {
 public void start() {
      int a = 23/7;
      int b = 23%7;
      int c = 7/23;
      int d = 7%23;
      double x = 7.0/2.0;
      double y = 7.0+2.0;
      System.out.print(a);
      System.out.print(b);
      System.out.print(c);
      System.out.println(d);
      System.out.println(x);
      System.out.println(y);
 }
}
```

25

Expressions with mixed types

The result of an arithmetic expression is the same type as the operands;

integer + integer → integer

e.g. 4 + 6 → 10

floating point + floating point → floating point

e.g. 4.0 + 6.1 → 10.1

If one operand is an integer, and the other is a floating point, then the result will be a floating point

integer + floating point → floating point

e.g. 3 + 2.1 → 5.1

26

Increment and decrement

Increases and decreases a variable by one

i.e. adds or subtracts one from the variable

Example

```
int a = 0;
a++;
System.out.println(a);

int b = 4;
b--;
System.out.println(b);
```

```
1
3
```

27

What you need to know

Java has 8 primitive types.

How to declare a variable.

Rules for variable identifiers.

Dynamic initialisation of a variable.

Printing variables using System.out.println();

Symbolic constants.

Expressions.

28

What you need to know

The assignment statement:

variable is on the left of =

an expression is on the right of =

Arithmetic operators

same types - result is of the same type

mixed types (int and double) - result is a double

Modulus

Increment/Decrement

Ex01

```
7
4
5
```

```
public class Ex01 {
 public void start() {
      int a = 4;
      int b = 5;
      int c = b;
      b = a;
      a = 7;

      System.out.println(a);
      System.out.println(b);
      System.out.println(c);
 }
}
```

30

Ex02

Total pay: $101.25

```
public class Ex02 {
 public void start() {
      final double PAY_PER_MINUTE = 0.25;
      final int MINUTES_PER_HOUR = 60;
      double totalPay;
      int minutes = 45;
      int hours = 6;
      int totalMinutes;

      totalMinutes = hours * MINUTES_PER_HOUR +
                                          minutes;
      totalPay = PAY_PER_MINUTE * totalMinutes;


      System.out.print("Total pay: $");
      System.out.println(totalPay);
 }
}
```

31

Ex03

I have $708690 and I wish to divide this evenly among 108 people so that each person get a whole number of dollars. Complete the program on the next slide which displays the number of dollars per person and the number of dollars which are left over.

```
> java Ex03
Dollars per person: $6561
Remaining dollars: $102
```

32

Ex03

```
public class Ex03 {
 public void start() {
      final int TOTAL_DOLLARS = 708690;
      int people = 108;
      int dollarsEach;
      int dollarsRemaining;



      System.out.print("Dollars per person: $");
      System.out.println(                    );
      System.out.print("Remaining dollars: $");
      System.out.println(                    );
 }
}
```

33

Ex02 - Memory picture

PAY_PER_MINUTE	0.25
MINUTES_PER_HOUR	60
totalPay	
minutes	45
hours	6
totalMinutes	