


# 1B1a


## Imperative Programming Part II

1 Copyright © 2003, Graham Roberts Department of Computer Science 

## Agenda

- We've seen the basic ideas of sequence, iteration and selection.
- Now let's look at what else we need to start writing useful programs.

Details now start to be really important.


2 Copyright © 2003, Graham Roberts Department of Computer Science 

## Reading

Finish reading Part I, chapters 1 and 2.

Start looking at Part IV, the Java Language Reference.


Note, there is a lot of detail in these chapters. Don't expect to read them once and be done.

3 Copyright © 2003, Graham Roberts Department of Computer Science 

## Displaying Messages

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


- Been using this to display text on the computer screen. How does it work?

4 Copyright © 2003, Graham Roberts Department of Computer Science 

## Objects, of course!

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

- *out* is another kind of *object* – a `PrintStream` object.
- A stream is a sequence of characters, with a *source* and a *sink*.
- *System.out* is a sink object connected to the computer display.

5 Copyright © 2003, Graham Roberts Department of Computer Science 


## `println` and `print`

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

- Display message, followed by a newline.
  - next message appears on a next line.

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

- Just display message.
  - next message appears on the same line as the last.

6 Copyright © 2003, Graham Roberts Department of Computer Science 

## \n

- \n is the character representation of newline.  
System.out.print("Hello World\n");  
has the same result as:  
System.out.println("Hello World");  
  
System.out.println("Hello World\n");  
will result in *two* newlines.

7

Copyright © 2003, Graham Roberts

Department of Computer Science



## Displaying messages using a loop

- ```
// This is real Java syntax
while (true)
{
    System.out.println("Hello");
}
```
- Hello for ever...

8

Copyright © 2003, Graham Roberts

Department of Computer Science



## Counting

- How do we display our message just 10 times?
- We obviously need to count 1 to 10, then stop.
- We need a counter! How?

9

Copyright © 2003, Graham Roberts

Department of Computer Science



## A counter

- We need a *container* to hold a counter, which can be incremented (add 1).
- We will call the container a *Variable*.
- The variable can hold an integer value we can count with.

10

Copyright © 2003, Graham Roberts

Department of Computer Science



## Let's think variable

- A variable is a container:



- It can hold a *value*,
- and needs a *name or identifier*.

11

Copyright © 2003, Graham Roberts

Department of Computer Science



## What is a value?

- Values are things like an integer or floating point number, or a character, or text...
- Values themselves are *abstract, intangible*.
- So we use *representations* of values in order to work with them.

12

Copyright © 2003, Graham Roberts

Department of Computer Science



## Representations

- 1, I, One, one, one, ONE
  - all representations of one.
- In the computer, integers are represented by binary numbers (e.g., 32-bit 2s complement binary numbers).
  - 11001010010111010000110001111001

13 Copyright © 2003, Graham Roberts

Department of Computer Science



## Other representations

- Floating point numbers are represented using IEEE 754 format.
- Characters are represented by *Unicode* binary character codes.
- Text by a sequence of characters.
- Boolean by binary zero or one.

14 Copyright © 2003, Graham Roberts

Department of Computer Science



## Why talk about representations?

- Representations have *finite ranges*.
  - 32-bit integer ranges from -2147483648 to 2147483647
- A variable holding an integer representation cannot *have a value* outside the range.

Note the use of this phrase.

15 Copyright © 2003, Graham Roberts

Department of Computer Science



## Questions?

16 Copyright © 2003, Graham Roberts

Department of Computer Science



## Type

- A variable container is very specific about the kind of values it can hold.
- A *type* defines what kind of value.
- To use a variable you *have* to state what type of value it can hold.
- We typically say “a variable has a type”.

17 Copyright © 2003, Graham Roberts

Department of Computer Science



## Common types

- boolean – true or false
- int – 32-bit 2's complement integer
- long – 64-bit 2's complement integer
- char – 16-bit unsigned Unicode character code
- float – 32-bit floating point
- double – 64-bit floating point  
(Check book for more detail)

18 Copyright © 2003, Graham Roberts

Department of Computer Science



## Just checking - What has type?

- A value has a type that defines what kind of value it is.
- A variable has a type that determines what kind of values it can hold.
- To store a value (or really its representation) in a variable, the types must match.

19 Copyright © 2003, Graham Roberts

Department of Computer Science



## Shape?

- It may help to think of a type denoting a shape.
- Only values of the right shape can fit in a variable of a given shape.

20 Copyright © 2003, Graham Roberts

Department of Computer Science



## Type int

- *int* is the name of the integer type (32-bit 2's complement).
- A variable named *size* of type *int* can be *declared* like this:  

```
int size;
```
- You have to *declare* a variable before you can use it.

21 Copyright © 2003, Graham Roberts

Department of Computer Science



## Declaration?

- Anything you name (such as a variable) must be introduced first, in order to know what is being named.
- The introduction, or *declaration*, gives the name and type of the thing being named.
- Once declared a name can be used but not before.

22 Copyright © 2003, Graham Roberts

Department of Computer Science



## Missing declaration...

- If you use a name that has not been declared the Java compiler will complain!

```
compiling: T1.java
T1.java:5: Undefined variable: counter
  counter = 10 ;
  ^
1 error
```

23 Copyright © 2003, Graham Roberts

Department of Computer Science



## Primitive types

- The types listed earlier (and a few others) are *primitive* types.
- Why? They are directly represented by typical processors (and, hence, the JVM).
- They are the most efficient.

24 Copyright © 2003, Graham Roberts

Department of Computer Science



## Non-primitive types?

- Yes, they not only exist but will be very important.
- Every kind of value we use must have a type: Address, BankAccount, Date, Book,...
- Non-primitive types are abstractions, constructed from primitive types.
- They are classes.

25 Copyright © 2003, Graham Roberts

Department of Computer Science



## String

- String is the type of a line of text:
  - “This is a String”
- It is a non-primitive type that is widely used.
- A String is actually an *object*.
- There is a class String.

26 Copyright © 2003, Graham Roberts

Department of Computer Science



## Questions?

27 Copyright © 2003, Graham Roberts

Department of Computer Science



## Integer variables

- What can you do with them?
- First *declare* your variable:
  - `int myInteger;`
- What is the value of this variable?
- It hasn't got one – you *must* give it one before you can use it.

28 Copyright © 2003, Graham Roberts

Department of Computer Science



## Initialising

```
int myInteger = 10;
```

- Declare `myInteger` *and* give it an initial value.
- Always, ALWAYS, *initialise* a variable.
- Actually you have no choice! The Java compiler will make sure you do.

29 Copyright © 2003, Graham Roberts

Department of Computer Science



## 10?

- 10 is a *literal* value of type `int`.
- All primitive types have literal values that can be used directly in a program.
- 3.141 is a floating point literal of type `double`.
- `true` and `false` are the boolean literals.

30 Copyright © 2003, Graham Roberts

Department of Computer Science



## More initialising

- `double d = 1.23456789;`
- `boolean b = false;`
- `char c = 'a';`
- `float f = 1.234F;`
- `int x = 0xff;`
- `String s = "Hello";`  
(Many more examples in book.)

31 Copyright © 2003, Graham Roberts

Department of Computer Science



## Changing a variable's value

- A variable is changed by an *assignment expression*.  
`age = 21;`
- The value 20 (or really its representation) is stored into the variable container.
- The old value is overwritten and lost.

32 Copyright © 2003, Graham Roberts

Department of Computer Science



## = or =

- Note that we have now used `=` for *two* things.  
`int length = 5;`  
`length = 20;`
- Initialisation v. assignment.
- Subtle but different. Why?

33 Copyright © 2003, Graham Roberts

Department of Computer Science



## State

- The state of a program is given by:
  - the Java Virtual Machine
  - the value of the variables
- The basic idea of computation is to transform the initial state to the final state.
- Each program instruction clicks the state forward one step.

34 Copyright © 2003, Graham Roberts

Department of Computer Science



## Wrong state?

- A computation can fail if any invalid state is reached (e.g., a variable has the wrong value).
- A typical computation may proceed through billions of states...

35 Copyright © 2003, Graham Roberts

Department of Computer Science



## Operators and expressions

- An operator applies an operation to values!
- `+, -, /, *`  
`x = 2 + 3;`  
`y = 3.2 * 2.4;`
- We can combine variables, operators and literals to write *expressions*.

36 Copyright © 2003, Graham Roberts

Department of Computer Science



## Comparison

- There are also *boolean operators* to compare values:

`<, >, <=, >=, ==, !=`

```
boolean b1 = (x < 5);
boolean b2 = (y == 6);
```

37 Copyright © 2003, Graham Roberts

Department of Computer Science



## Precedence

- How do you know the meaning of:  
 $2 + 3 * 5 / 8$
- You use *precedence rules* – these determine which operators are applied first.
- *High precedence* operators are applied before *low precedence*.

38 Copyright © 2003, Graham Roberts

Department of Computer Science



## Use brackets

- We can bracket the sub-expressions to make the evaluation order:

$2 + ((3 * 5) / 8)$

39 Copyright © 2003, Graham Roberts

Department of Computer Science



## Lots of operators

- See the book for the full list!!
- Understand the difference between unary and binary operators.
- Check the precedence table.

40 Copyright © 2003, Graham Roberts

Department of Computer Science



## Types and operators

- A type determines exactly which operators can be applied to a value.
- No other operators can be applied.

`x = 2 ! 3; // Error!`

- Meaningless as `!` is not a binary operator taking integer arguments.
- Won't compile.

41 Copyright © 2003, Graham Roberts

Department of Computer Science



## So a type is?

- A type defines:
  - the set of values belonging to the type.
  - the set of operations that can be applied to the values.
- In our programs, values of a type are given concrete (and finite) representations.

42 Copyright © 2003, Graham Roberts

Department of Computer Science



## Questions?

43 Copyright © 2003, Graham Roberts

Department of Computer Science



## Interesting...

- Given assignment and operators we can write:

$$x = x + 1;$$

- Mathematicians panic now...
- But, of course, we are not writing a mathematical formula.
- This is a program *statement*.

44 Copyright © 2003, Graham Roberts

Department of Computer Science



## Statement v. Expression

- A statement is a complete instruction.
  - $x = y + z;$
  - `while (a < 10) { ... }`
  - `if (c != d) { ... }`
- An expression is part of a statement.
  - $x + y$
  - $a < b$

45 Copyright © 2003, Graham Roberts

Department of Computer Science



## ++ (and --)

$$x = x + 1;$$

$$x = x - 1;$$

- Increment or decrement a variable.
- Can use the ++ or -- operators:
  - Or `x++;` `x--;`
  - Or `++x;` `--x;`
  - Or `x += 1;` `x -= 1;`

46 Copyright © 2003, Graham Roberts

Department of Computer Science



## Oh, yes...

- This all started as we wanted a counter for our program.
- We now have the bits, let's put them together.

47 Copyright © 2003, Graham Roberts

Department of Computer Science



## Counting

```
int counter = 0; // We count from zero
while (counter < 10)
{
    System.out.println("Hello");
    counter++;
}

Done!!
```

48 Copyright © 2003, Graham Roberts

Department of Computer Science



## A bit more...

```
int counter = 0; // We count from zero
while (counter < 10)
{
    System.out.print("Hello ");
    System.out.println(counter);
    counter++;
}
```

49 Copyright © 2003, Graham Roberts

Department of Computer Science



## Result...

```
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
Hello 6
Hello 7
Hello 8
Hello 9
Hello 10
```

50 Copyright © 2003, Graham Roberts

Department of Computer Science



## A bit more compact...

```
int counter = 0; // We count from zero
while (counter ++ < 10)
{
    System.out.println("Hello " + counter);
}
```

Note the way this code is laid out.  
Use indentation and blank space  
to the best effect.

51 Copyright © 2003, Graham Roberts

Department of Computer Science



## Are there other kinds of loop?

- Yes!
- We have:
  - while loops
  - do loops
  - for loops

52 Copyright © 2003, Graham Roberts

Department of Computer Science



## While loops

- Seen them already:
 

```
while (boolean-expression)
{
    // Statements in loop body
}
```
- The loop body will be executed *zero* or more times.

53 Copyright © 2003, Graham Roberts

Department of Computer Science



## Do loops

- do
 

```
{
    // Statements in loop body
}
```

```
while (boolean-expression);
```
- The loop body will be executed *one* or more times.

54 Copyright © 2003, Graham Roberts

Department of Computer Science



## For loops

- Often used for counting:  
for (start ; limit ; increment/decrement)  
{  
    // Statements in loop body  
}
- Count from start to limit by increment/decrement size.

55 Copyright © 2003, Graham Roberts

Department of Computer Science



## For Loop Example

- ```
for (int counter = 0 ; counter < 10 ; counter++)  
{  
    System.out.println("Hello " + counter);  
}
```
- Start at zero, then count up by one, while less than 10.

56 Copyright © 2003, Graham Roberts

Department of Computer Science



## Evens

- ```
for (int counter = 0 ; counter < 10 ; counter = counter + 2)  
{  
    System.out.println("Hello " + counter);  
}
```
- Count up 0,2,4,6,8

57 Copyright © 2003, Graham Roberts

Department of Computer Science



## While v. For

- Q. Is a for loop a while loop in fancy dress?
- A. Yes!
- A for loop can be seen as *syntactic sugar*.
- But it often gives a neater solution than a while loop, especially if counting.

58 Copyright © 2003, Graham Roberts

Department of Computer Science



## While, do, for – which to use?

- Many problems can be solved using any kind of loop.
- However, often one kind of loop gives a better (more elegant) solution.

59 Copyright © 2003, Graham Roberts

Department of Computer Science



## Loops – want to know more?

See the text book and do the exercises!

Questions?

60 Copyright © 2003, Graham Roberts

Department of Computer Science



## Remember selection?

- The if statement
 

```
if (boolean-expression) // Must have the brackets
{
    // Statement sequence
}
else
{
    // Statement sequence
}
```

61 Copyright © 2003, Graham Roberts

Department of Computer Science



## Short-cut?

- You can write:
 

```
if (boolean-expression)
    statement; // No braces
next-statement;
```
- In fact, you can do the same with loops.

62 Copyright © 2003, Graham Roberts

Department of Computer Science



## But...

Originally write:

```
if (x > 10)
    x = 10; // Limit x
z = x * y; // Use x
```

But then change:

```
if (x > 10)
    x = 10; // Limit x
    y = 1; // and update y
z = x * y; // Use x
```

Uh oh, this was meant to be executed only if  $x > 10$ ...

63 Copyright © 2003, Graham Roberts

Department of Computer Science



## Moral

*Always* put the braces in, even when the if statement (or loop) body contains only a single statement.

64 Copyright © 2003, Graham Roberts

Department of Computer Science



## Defensive programming

- Anticipate the kinds of programming errors you might make.
- Write the code in a style that prevents mistakes happening or, at least, makes them stand out.
- Code layout, indentation, use of blank space, use of braces all help.

65 Copyright © 2003, Graham Roberts

Department of Computer Science



## More selection?

- Yes.
- Check out the *switch* statement.
- Look at the conditional operator (a ternary operator).

All in the book!

66 Copyright © 2003, Graham Roberts

Department of Computer Science



## Statements (repeat)

- We've been using this bit of jargon – let's just be clear what it means.
- A statement is a complete command terminated by a semi-colon.

```
a = b * c * d ; // A statement
```

In fact, an assignment statement.

67

Copyright © 2003, Graham Roberts

Department of Computer Science



## Expression?

- An expression is a sub-part of a statement:

$$1 * 2$$

$$a + b / c$$

- A full statement can be constructed from a number of expressions.

```
int a = y * (p + q) - (r / s) ;
```

68

Copyright © 2003, Graham Roberts

Department of Computer Science



## Compound Statement

- A sequence of statements bracketed by braces.

```
{
  a = 1 * 2;
  d = b / c;
}
```

- Loop and if statement bodies.

69

Copyright © 2003, Graham Roberts

Department of Computer Science



## Summary

- Programs need to work with values.
- We use variables, assignment and operators.
- Variables have types.
- We can write expressions and statements.
- We can do selection and iteration.

70

Copyright © 2003, Graham Roberts

Department of Computer Science

