Data Structures & Algorithms 1

Programming Revision

1

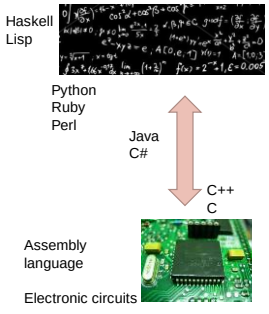
Programming language

- We will need to use some programming language to represent data structures and algorithms
- We will use the Java language
- However, you could use any other programming language to encode the same ideas - another popular language is C++

2

Programming Languages

- Languages are on a continuum from low-level electronics to high-level
- At the lowest level the programming language provides no abstraction from the physical device
- At the highest level the language is so abstract it is purely mathematical
- Java is in the middle



3

Java programming

- Java is a programming language first released in 1995 originally developed by James Gosling at Sun Microsystems
- One reason Java is popular is because it is platform independent
- Programs written in Java can run on any hardware or operating-system
- Compiled code is run on a Java Virtual Machine (JVM) which converts it to the native language

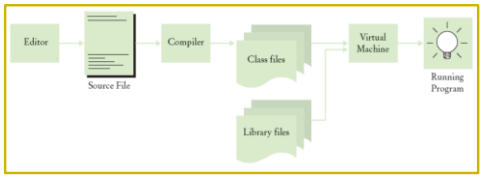
4

Platform independence

- Turing showed that machine, software and input can all be represented in terms of patterns of information
- The compiler translates the Java code into machine code that the JVM can run
- The JVM is a machine simulated by the actual physical machine it is running on

5

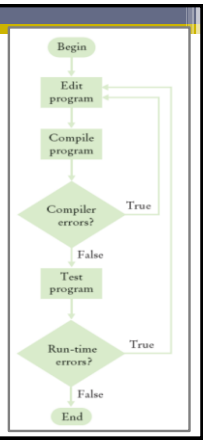
The compilation process



6

Edit, compile, run

- Compiling turns the code you wrote in Java (.java file) into a format that the computer can run on the JVM (.class file)
- You can't run your code without compiling it
- Every time you change your code you need to recompile



7

Revision

- We will now revise the following:
  - Variables & Data Types: (ints, doubles)
  - Variable Operators: (addition, subtraction)
  - Selection: (if, else)
  - Iteration: (for, while, do)

8

Variables

- Variable is a name for a location in memory
- 2 types of variables
  - Primitive (e.g. int and double – usually smaller case letters)
  - Reference (e.g. objects – usually starts with capital letter)
- Must have a type and a name
  - Cannot be a reserved word (public, void, static, int, …)

data type → int, variable name → total

```
int total;
```

9

Variables

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is not initialized, the value of that variable is undefined

10

Scope & garbage collection

- Variables defined within a member function are local to that function (this is referred to as the scope of a variable)

```
for (int i = 0; i < 50; i++){…}
```

- Local variables are destroyed (garbage collected) when function exits (or goes out of scope.)
- Programmer need not worry about de-allocating memory for out of scope objects/variables.
  - Unlike in C or C++

11

Assignment

- An *assignment statement* changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```

- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in `total` is overwritten
- You can assign only a value to a variable that is consistent with the variable's declared type

12

Primitive types

- There are exactly eight primitive data types in Java
- Four of them represent integers:
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers:
  - `float`, `double`
- One of them represents characters:
  - `char`
- And one of them represents true/false boolean values:
  - `boolean`

13

Bits and bytes

- A single bit is a one or a zero, a true or a false, a "flag" which is on or off
- A byte is made up of 8 bits like this : 10110001
- 1 Kilobyte = about 1,000 bytes (1,024 to be precise)
- 1 Megabyte = about 1,000,000 bytes (1,024 * 1,024)
- 1 Gigabyte = about 1,000,000,000 bytes

14

Primitive types

Type	Description	Size
`int`	The integer type, with range –2,147,483,648 . . . 2,147,483,647	4 bytes
`byte`	The type describing a single byte, with range –128 . . . 127	1 byte
`short`	The short integer type, with range –32768 . . . 32767	2 bytes
`long`	The long integer type, with range –9,223,372,036,854,775,808 . . . –9,223,372,036,854,775,807	8 bytes

15

Primitive types

Type	Description	Size
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme	2 bytes
`boolean`	The type with the two truth values `false` and `true`	1 bit

16

Number types

- Illegal to assign a floating-point expression to an integer variable

```
double balance = 13.75;
int dollars = balance; // Error
```

- `Casts`: used to convert a value to a different type

```
int dollars = (int) balance; // OK
```

- `Math.round` converts a floating-point number to nearest integer

```
long rounded = Math.round(balance);
// if balance is 13.75, then
// rounded is set to 14
```

17

Arithmetic expressions

- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- If either or both operands associated with an arithmetic operator are floating point, the result is a floating point

18

Modulus operator %

- The % symbol is the modulus operator
- This divides the first number by the second number and gives you the remainder
  - 55 % 10 = 5
  - 42 % 4 = 2

19

Answer

- Both of these work
- How can we figure out how many times 7 divides into a variable called ***number***?
  - (number - (number % 7))/ 7
  - number / 7 – ((number / 7) % 1)

20

Operator precedence

- Operators can be combined into complex expressions

  result = total + count / max - offset;

- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation (BOMDAS rule)
- Arithmetic operators with the same precedence are evaluated from left to right
- Parentheses can be used to force the evaluation order

21

Increment and decrement

- The increment and decrement operators are arithmetic and operate on one operand
- The *increment operator* (`++`) adds one to its operand
- The *decrement operator* (`--`) subtracts one from its operand
- The statement `count++;`

  is functionally equivalent to `count = count + 1;`

22

Assignment operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

  num += count;

  is equivalent to

  num = num + count;

23

Relational operators

- `>` greater than
- `>=` greater than or equal to
- `<` less than
- `<=` less than or equal to
- `==` equal to
- `!=` not equal to

24

Frequent mistake!!

- If we want to put the variable "number" equal to ten we use one equals sign
  - number = 10;
- However, if we want to check *if* number is equal to ten then we use a double equals
  - if (number == 10)

25

The `Math` class

- Math class: contains methods like `sqrt` and `pow`
- To compute x^n, you write `Math.pow(x, n)`
- However, to compute x^2 it is significantly more efficient simply to compute `x * x`
- To take the square root of a number, use the `Math.sqrt`; for example, `Math.sqrt(x)`

26

The Math class

- In Java,

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

can be represented as

```
(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
```

27

Mathematical methods in Java

`Math.sqrt(x)`	square root
`Math.pow(x, y)`	power x^y
`Math.exp(x)`	e^x
`Math.log(x)`	natural log
`Math.sin(x)`, `Math.cos(x)`, `Math.tan(x)`	sine, cosine, tangent (x in radian)
`Math.round(x)`	closest integer to x
`Math.min(x, y)` `Math.max(x, y)`	minimum, maximum

28

Questions

- What is the value of `643 / 100`?
  - Depends on whether *double* or *int*
- What is the value of `643 % 100`?
  - 43
- Why doesn't the following statement compute the average of `s1`, `s2`, and `s3`?
  - Missing brackets

```
double average = s1 + s2 + s3 / 3; // Error
```

29

Strings

- A string is a sequence of characters
- Strings are objects of the `String` class
- String variables:

```
String message = "Hello, World!";
```

- String length:

```
int n = message.length();
```

- Empty string:

```
""
```

30

Concatenation

- Use the + operator:

```
String name = "Dave";
String message = "Hello, " + name;
   // message is "Hello, Dave"
```

- If one of the arguments of the + operator is a string, the other is converted to a string

```
String a = "Agent";
int n = 7;
String bond = a + n; // bond is Agent7
```

31

Concatenation when printing

- Useful to reduce the number of `System.out.print` instructions

```
System.out.print("The total is ");
System.out.println(total);
```

versus

```
System.out.println("The total is " + total);
```

32

Converting between Strings and numbers

- Convert to number:

```
int n = Integer.parseInt(str);
double x = Double.parseDouble(str);
```

- Convert to string:

```
String str = "" + n;
str = Integer.toString(n);
```

33