Building Blocks

The Big Picture

Chapter 1 provides background –

- how computers run programs
- the basic vocabulary and building blocks of programming
  - many programming languages share certain fundamental concepts and structures

Chapter 2 begins the study of Java –

- basic structure of a Java program
- instructions for storing and manipulating values
- input and output

The Programming Environment

A three step process –

- **edit** – create a plain text file containing the program
- **compile** – translate the program into the low-level machine language that the computer can actually run
  - computers are essentially a collection of little switches wired together
  - machine language instructions deal with turning the switches on and off
- **run** – the computer carries out the machine language instructions

The Programming Environment

- edit
- compile
- run

Java splits these steps up a little differently than most languages –

- `javac` translates the Java program into *bytecode*, a low-level machine independent language
- `java` translates the bytecode into machine language and then carries out the instructions

The Programming Environment – VS Code

- edit file
  - don't forget to save!
- open a terminal
  - Terminal → New Terminal
- change the working directory to the directory containing the program files
  - `cd` *`dirname`* to go into *dirname*
  - `cd ..` to go up one level
- compile

  `javac HelloWorld.java`
  - if successful, creates/updates `HelloWorld.class`
- run

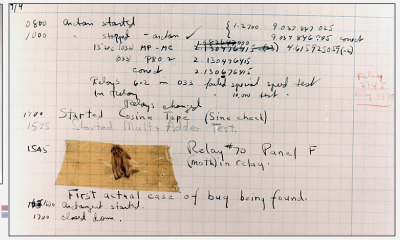
  `java HelloWorld`

Key Points

this process has implications for the kinds of errors you see

- *compiler errors* are errors in the translation process
  - syntax errors
- *runtime errors* involve bad values or incorrect instructions
  - logic errors (bugs)

Rear Admiral
Grace Murray Hopper



https://en.wikipedia.org/wiki/Grace_Hopper
https://en.m.wikipedia.org/wiki/File:First_Computer_Bug,_1945.jpg

Key Points

carrying out an instruction is a mechanical process

- a given instruction results in setting switches a certain way

this means the computer can only do exactly what you tell it

- every detail of the program must be correct at the same time in order for the program to work

consequences

- attention to detail is important
- trial-and-error is ineffective for getting a program to work
  - need to learn what is right so you can identify what is wrong

Program Structure

```
public class Progname {
  public static void main ( String[] args ) {
    statements
  }
}
```

- syntax
  - the parts in *italics* are not typed literally – replace with the appropriate thing for your specific program
  - the rest should appear as shown
    - case of keywords must be exactly as shown; convention is that *`Progname`* starts with a capital letter
    - there must be whitespace in the places shown but the type of whitespace (line breaks vs spaces, level of indentation) is a convention for easier reading
  - file must be named *`Progname.java`*
- semantics
  - program execution starts with the first statement in `main` and then moves through the rest in order from top to bottom

Values

a program is a list of instructions (*statements*) that manipulate *values*

- values can be numbers, text, true/false, …

Java is a *strongly-typed* language

- every value has an associated *type* indicating what kind of value it is
  - e.g. 5 is an `int`, 5.8 is a `double`
    - 5.0 is also a `double`
  - e.g. “hello” is a `String`
  - e.g. ‘h’ is a `char` (character)
  - e.g. `true` is a `boolean`
- the type determines how you can manipulate values
  - e.g. you can divide two `doubles` but you can't divide two `Strings`

`int`
`double`
`char`
`String`
`boolean`

For each of the following, choose the most appropriate type for a variable holding that value.

someone's age	[Choose]
someone's name	[Choose] ★
whether or not it is currently raining	[Choose] ★
your bank account balance	[Choose] ★
the answer to a multiple choice question (choices are labeled a-d)	[Choose]
the number of days in the current month	[Choose]
the length of a car trip	[Choose] ★

- `int` vs `short`, `long`
  - all three are integer types – the difference is how much space is used to hold a value, and thus the range of values that can be stored
  - we typically use `int` because it has a large enough range for most applications (-2147483648 to 2147483647) without taking up unnecessary space
    - `long` takes twice as much space as `int`
    - `short` takes half the space of `int` but has only a small range (-32768 to 32767)

- `double` vs `float`
  - both are *floating point* types – the difference is how much space is used to hold a value, and thus the precision of values that can be stored
  - double takes twice as much space as float, but typically the greater precision is desired to reduce numerical errors

Literals

a *literal* is a value written directly

- for numbers, just write the number: `5` or `5.8`
  - it is a `double` value if there is a decimal point and an `int` value if not
- for `String`s, use double quotes: “`abc`”
- for `char`s, use single quotes: `'a'`
- for `boolean`s, it is just the words `true` and `false`

be careful not to be fooled by what the value might look like

- `5` is an `int`, `5.0` is a `double`
- `'a'` is a `char`, “`a`” is a `String`
- `5`, `'5'`, ”`5`” are all different – `int`, `char`, and `String`
- `true` is a `boolean`, “`true`” is a `String`