Inf1B

Getting Started

Fiona Mcneill
adapting earlier versions by Perdita Stevens, Ewan Klein, Volker Seeker, et al.

School of Informatics

Where have you left off last semester?

Haskell

functional

Java™

imperative
object oriented

Imperative Programming

Pancake Recipe

- Take a bowl
- Add flour
- Add eggs
- Add milk
- While not yet smooth
  - Whisk the batter
- Fry in a pan

- statements are used which are processed step by step
- programs carry state which in OO is expressed in objects

What is object orientation?

It means: your program is structured like the domain (real world).
Objects (organised into classes of similar objects) typically represent things (organised into types of similar things).
Objects have

- state: they can store data
- behaviour: they can do things, in response to messages
- identity: two objects with the same state can still be different objects.

Any of state, behaviour, identity can be trivial for a particular object, though.
In Java, all behaviour is associated with a class. However, it can be static – that is, not associated with any particular object of the class.

A First Example

HelloWorld.java

```
/*************************
 * Prints "Hello, World!"
 *************************/

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

Creating a New Class

1. All Java code sits inside a class.
2. By important convention, class names are capitalised and in ‘CamelCase’.
3. Each class goes into a file of its own (usually; and always in this course).
4. The name of the file has to be the same as the name of the class, and suffixed with `.java`.

Compiling Classes

1. Java is a **compiled language**.
2. This means that it has to be converted into machine code.
3. Traditionally, you would do this at the command line.
   3.1 Write your class - e.g., `MyClass.java`.
   3.2 Compile it using `javac MyClass.java`
   3.3 This will create a machine readable class file called MyClass.class
   3.4 Now you can run your class using `java MyClass`
4. But ... nowadays mostly people use an IDE (Integrated Design Environment) so you don't really need to worry about this!

A First Example

Declare a class

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

- Basic form of a class definition.
- Class definition enclosed by curly braces.

A First Example

Declare the `main()` method

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

- We need a `main()` method to actually get our program started.
- All our other code is invoked from inside `main()`.
- `void` means the method doesn't return a value.
- The argument of the method is an array of `String`s; this array is called `args`.
- Definition of a method enclosed by curly braces.

A First Example

Print a string to standard output

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

- `System.out` is an object (a rather special one).
- `println("Hello World!")` is a message being sent to that object: `println` is the method name, `"Hello World!"` is the argument.
- The whole line is a statement: must be terminated with a semi-colon (`;`).
- Strings must be demarcated by double quotes.
- Strings cannot be broken across a line in the file.

Compiling

- The program needs to be compiled before it can be executed.
- Use the `javac` command in a terminal.

At the terminal

```
javac HelloWorld.java
```

- If there's a problem, the compiler will complain.
- If not, compiler creates a Java bytecode file called `HelloWorld.class`.

Running the Program

- Now that we have compiled code, we can run it.
- Use the `java` command in a terminal.

At the terminal

```
java HelloWorld
Hello World!
```

Running the Program

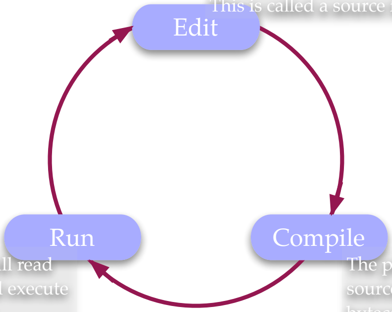
- Now that we have compiled code, we can run it.
- Use the `java` command in a terminal.

At the terminal

```
java HelloWorld
Hello World!
```

- Note that we omit the `.class` suffix in the run command. The `java` command wants a classname as argument, not a filename.

Edit-Compile-Run Cycle

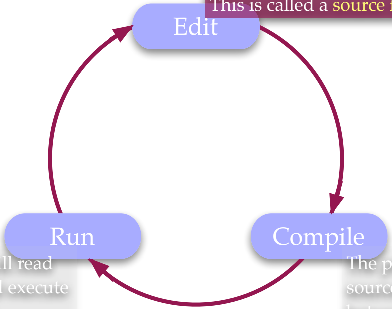


Type in the program using an editor and save the program to a file.
Use the name of the main class and the suffix .java for the file.
This is called a source file.

The process of compiling a source file generates the bytecode file.
The byte code will have a .class suffix; the prefix will be the same.

A java interpreter will read the bytecode file and execute the instructions in it.
If an error occurs while running, the interpreter will stop its execution.

Edit-Compile-Run Cycle

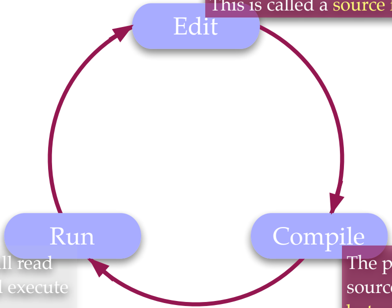


Type in the program using an editor and save the program to a file.
Use the name of the main class and the suffix .java for the file.
This is called a source file.

The process of compiling a source file generates the bytecode file.
The byte code will have a .class suffix; the prefix will be the same.

A java interpreter will read the bytecode file and execute the instructions in it.
If an error occurs while running, the interpreter will stop its execution.

Edit-Compile-Run Cycle

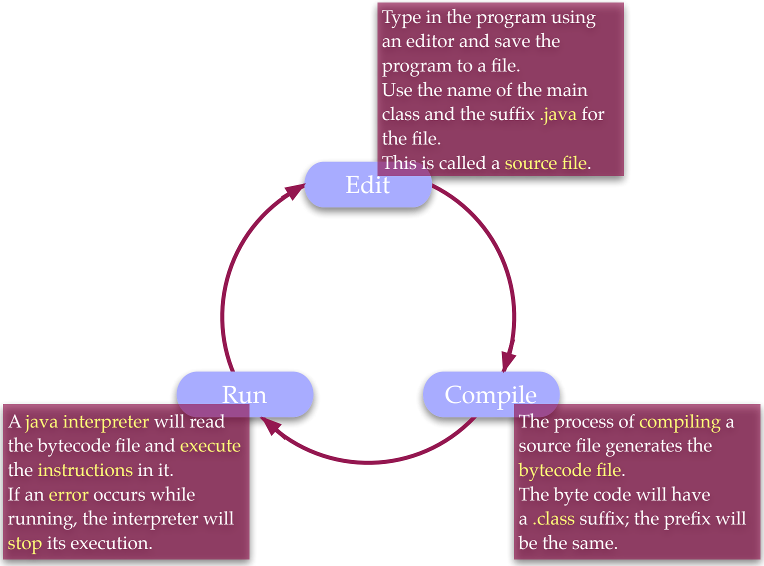


Type in the program using an editor and save the program to a file.
Use the name of the main class and the suffix .java for the file.
This is called a source file.

The process of compiling a source file generates the bytecode file.
The byte code will have a .class suffix; the prefix will be the same.

A java interpreter will read the bytecode file and execute the instructions in it.
If an error occurs while running, the interpreter will stop its execution.

Edit-Compile-Run Cycle



Type in the program using an editor and save the program to a file.
Use the name of the main class and the suffix .java for the file.
This is called a source file.

The process of compiling a source file generates the bytecode file.
The byte code will have a .class suffix; the prefix will be the same.

A java interpreter will read the bytecode file and execute the instructions in it.
If an error occurs while running, the interpreter will stop its execution.

Edit-Compile-Run Cycle

- The program needs to be compiled before it can be executed.
- If you edit a program, you need to compile it again before running the new version.
- However, if you use an integrated development environment, this may compile your code automatically.

Golden Rules of Programming

1. Compile often
2. Save regularly

Development Best Practices

Golden Rules of Programming

1. Compile often
2. Save regularly

Why? Detect errors early!

- Compiler checks syntactical correctness
- Running checks (some) semantic correctness
- Unit tests check (more) semantic correctness

Basic Functionality

Arithmetic

Addition and Division

```
public class Calc {

   public static void main(String[] args) {
      System.out.print("The sum of 6 and 2 is ");
      System.out.println(6 + 2);

      System.out.print("The quotient of 6 and 2 is ");
      System.out.println(6 / 2);
   }
}
```

Output

Arithmetic

Addition and Division

```
public class Calc {

   public static void main(String[] args) {
      System.out.print("The sum of 6 and 2 is ");
      System.out.println(6 + 2);

      System.out.print("The quotient of 6 and 2 is ");
      System.out.println(6 / 2);
   }
}
```

Output

```
The sum of 6 and 2 is 8
The quotient of 6 and 2 is 3
```

String Concatenation, 1

String Concatenation

```
public class Concat {

   public static void main(String[] args) {
   System.out.println("The name is " + "Bond, "
                         + "James Bond");
   }
}
```

Output

```
The name is Bond, James Bond
```

String Concatenation, 2

String Concatenation

```
public class Concat {

   public static void main(String[] args) {
   System.out.println("Is that you, 00" + 7 + "?");
   }
}
```

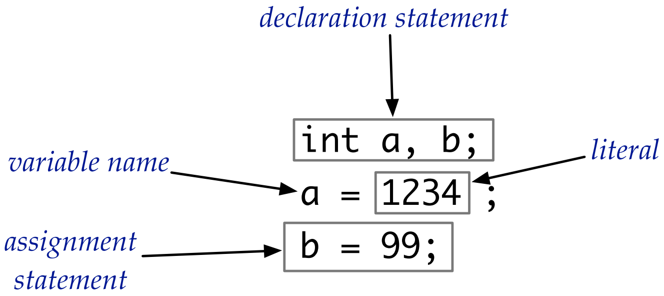
Output

```
Is that you, 007?
```

Assignment: Basic Definitions

Variable: A name that refers to a value

Assignment Statement: Associates a value with a variable



```
int a, b;
a = 1234 ;
b = 99;
```

Important: = is the operator in an imperative statement, not a logical assertion.

Assignment: Combining Declaration and Initialisation

Variables that have been declared, but not assigned to, are a potential source of error. (Exercise for the keen: understand what happens to them in Java.)
It's often best to declare a variable and *initialise* it at the same time.

```
int a, b;
a = 1234;
b = 99;
int c = a + b;
```

combined declaration and assignment statement

Hello World with Added Variables

Storing a String in a variable

```
public class HelloWorld {

   public static void main ( String [] args ) {
        String msg = "Hello World!";
        System.out.println( msg );
      }
}
```

Built-in Data Types

type	value set	literal values	operations
`char`	characters	`'A'`, `'$'`	compare
`String`	sequences of characters	`"Hello World!"`, `"Java is fun"`	concatenate
`int`	integers	17, 1234	add, subtract, multiply, divide
`double`	floating-point numbers	`3.1415`, `6.022e23`	add, subtract, multiply, divide
`boolean`	truth values	`true`, `false`	and, or, not

Integer operations

expression	*value*	*comment*
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	

Integer operations

expression	*value*	*comment*
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	
`5 / 2`	2	no fractional part

Integer operations

expression	*value*	*comment*
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	
`5 / 2`	2	no fractional part
`5 % 2`	1	remainder

Integer operations

expression	*value*	*comment*
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	
`5 / 2`	2	no fractional part
`5 % 2`	1	remainder
`1 / 0`		run-time error

Integer operations

expression	*value*	*comment*
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	
`5 / 2`	2	no fractional part
`5 % 2`	1	remainder
`1 / 0`		run-time error
`3 * 5 - 2`	13	`*` has precedence

Integer operations

expression	*value*	*comment*
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	
`5 / 2`	2	no fractional part
`5 % 2`	1	remainder
`1 / 0`		run-time error
`3 * 5 - 2`	13	`*` has precedence
`3 + 5 / 2`	5	`/` has precedence

Integer operations

expression	*value*	*comment*
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	
`5 / 2`	2	no fractional part
`5 % 2`	1	remainder
`1 / 0`		run-time error
`3 * 5 - 2`	13	`*` has precedence
`3 + 5 / 2`	5	`/` has precedence
`3 - 5 - 2`	-4	left associative
`( 3 - 5 ) - 2`	-4	better style
`3 - ( 5 - 2 )`	0	unambiguous

Floating-Point Numbers

The default floating-point type in Java is `double`.

Floating-Point Operations

expression	*value*
`3.141 + .03`	`3.171`
`3.141 - .03`	`3.111`
`6.02e23 / 2.0`	`3.01e23`
`5.0 / 3.0`	`1.6666666666666667`
`10.0 % 3.141`	`0.577`
`1.0 / 0.0`	`Infinity`
`Math.sqrt(2.0)`	`1.4142135623730951`
`Math.sqrt(-1.0)`	`NaN`

Type Conversion

Sometimes we can convert one type to another.

- Automatic: OK if no loss of precision, or converts to string
- Explicit: use a cast or method like `parseInt()`

expression	*result type*	*value*
`"1234" + 99`	`String`	"123499"
`Integer.parseInt("123")`	`int`	123
`(int) 2.71828`	`int`	2
`Math.round(2.71828)`	`long`	3
`(int) Math.round(2.71828)`	`int`	3
`(int) Math.round(3.14159)`	`int`	3
`11 * 0.3`	`double`	3.3
`(int) 11 * 0.3`	`double`	3.3
`11 * (int) 0.3`	`int`	0
`(int) (11 * 0.3)`	`int`	3

Type Conversion

Moral:

If you want a floating-point result from division, make at least one of the operands a `double`

Summary

- Java is an object oriented, imperative programming language
  - statements are executed step by step
  - objects carry state and have behaviour
- Java is a compiled language (Edit-Compile-Run)
- The entry point into every Java program is the `main` function
- Variables carry values of different types (`int`, `char`, `float`, `boolean`, `String`, ...)
- A range of arithmetic operations can be used
- **casting** is one way to convert between types
- Programs can receive user input at start time using **command line arguments**

Reading

Java Tutorial

pp1-68, i.e. Chapters 1 *Getting Started*, 2 *Object-Oriented Programming Concepts*, and Chapter 3 *Language Basics*, up to *Expressions, Statements and Blocks*

– except note:

- We use IntelliJ, not NetBeans as our IDE.
- We'll come to the Chapter 2 material later.
- We'll talk about Arrays later.

I suggest skimming Ch 2 and the Arrays section, and rereading them later.

Objects First

Appendix *B.1 - B.2*, Appendix *C.1*, Appendix *E.1* and *E.3*

This book has a different order of topics but is generally great for beginners and has some excellent summaries of basics.