Inf1-OP

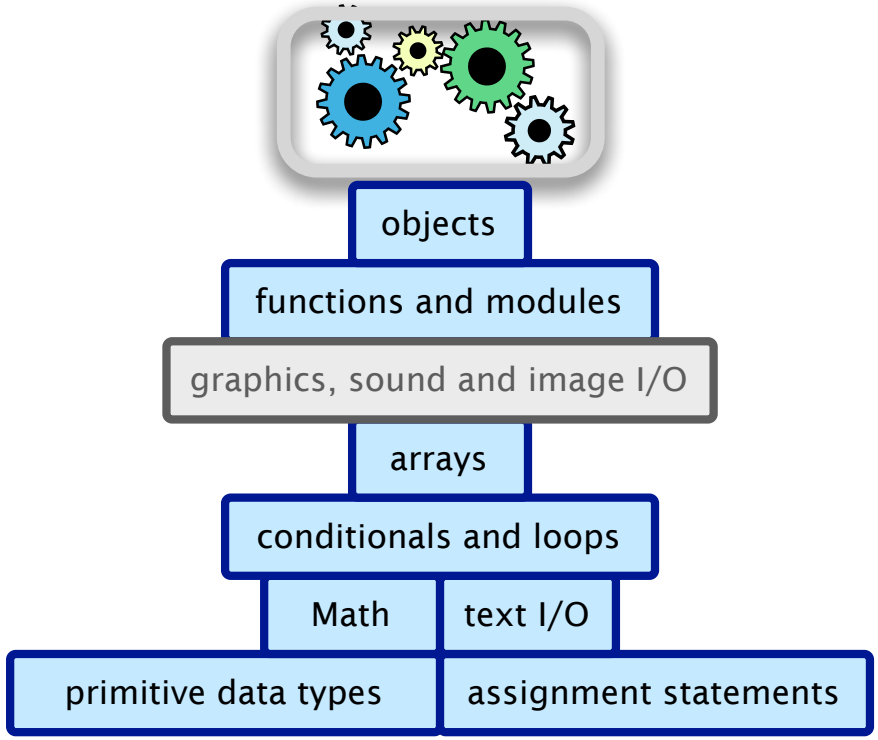
Getting Started

Volker Seeker, adapting earlier version by Perdita Stevens and Ewan Klein

School of Informatics

January 11, 2019

A Foundation for Programming



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. Our first objects will be just little bits of wrapped up behaviour.

Development Environment

Where have I left you last semester?

- nice graphical interface
- built-in game library
- auto-compilation
- a game framework to get you started

Development Environment

What we use in this course

- Integrated Development Environment (IDE)
- has everything Greenfoot has (except for built-in game library) and more

Development Environment

What we use in this course

- Integrated Development Environment (IDE)
- has everything Greenfoot has (except for built-in game library) and more

But...

Development Environment

... consider the following

Development Environment

This week we use the command line!

- move and modify files with the command line
- command line interface to compile and run
- simple text editor to edit source (e.g. gedit)

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 capitalized and in 'CamelCase'.
3. Each class goes into a file of its own (usually; and always in this course).
4. So, use a text editor (e.g., gedit) to create a file called `HelloWorld.java`.
5. The name of the file has to be the same as the name of the class, and suffixed with `.java`.

At the terminal

```
gedit HelloWorld.java
```

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

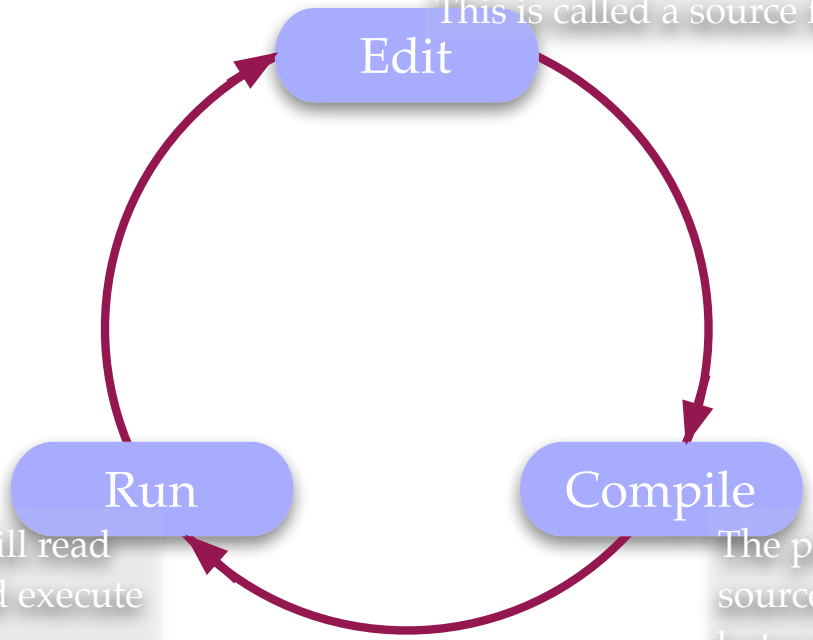
- Now that we have compiled code, we can run it.
- Use the `java` command in a terminal.

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

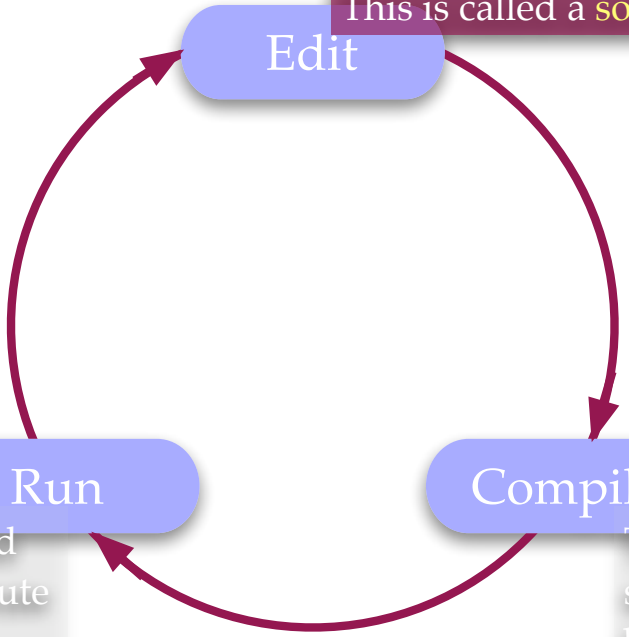


Type in the program using an editor and save the program to a file.
Use the name of the main class and the suffix .java for the file.
This is called a source file.

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

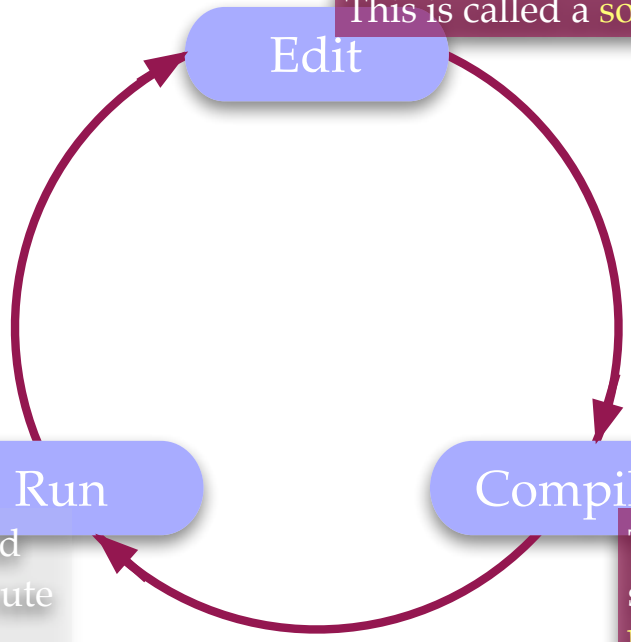


Type in the program using an editor and save the program to a file.
Use the name of the main class and the suffix .java for the file.
This is called a source file.

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

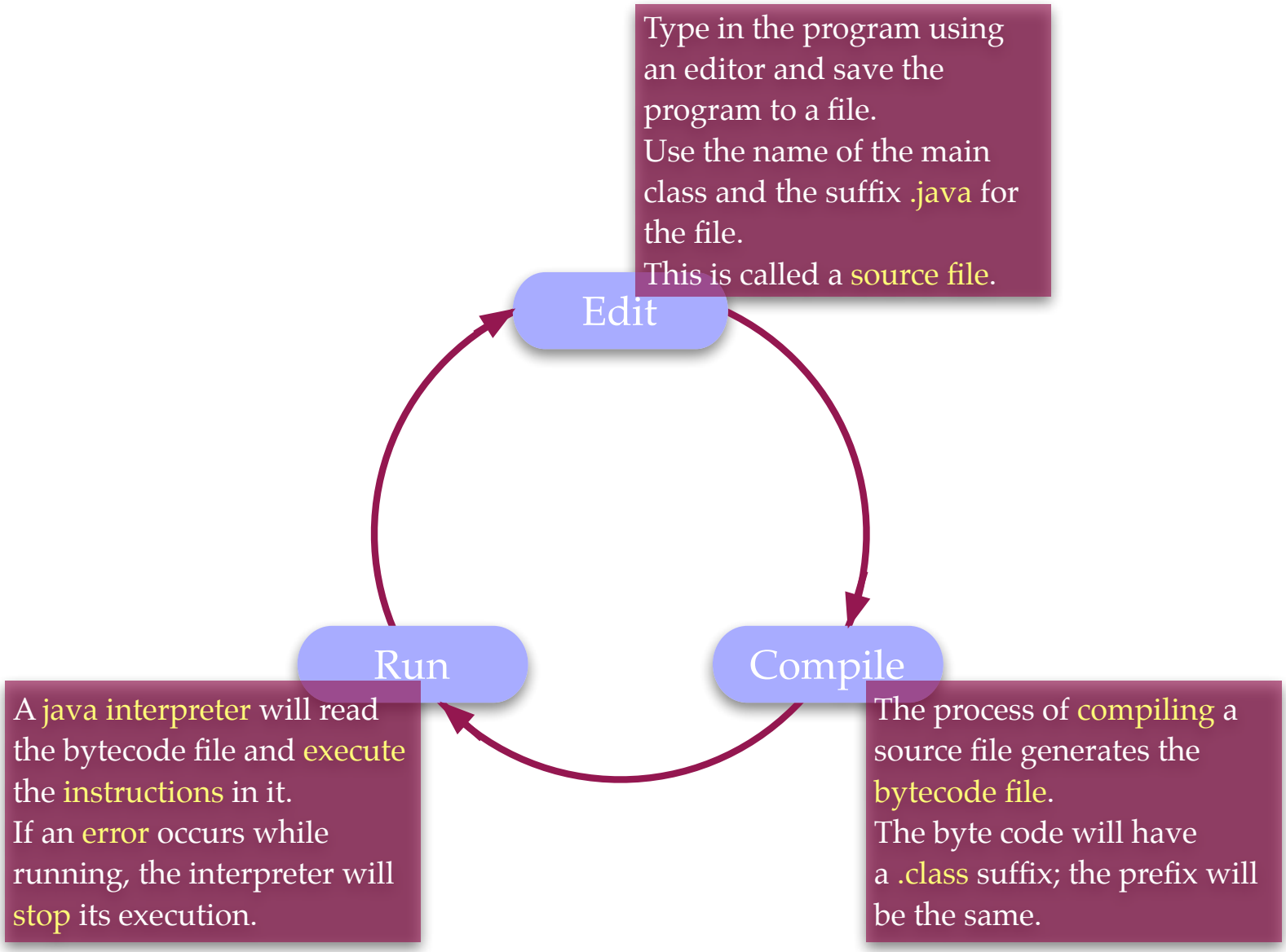


Type in the program using an editor and save the program to a file.
Use the name of the main class and the suffix .java for the file.
This is called a source file.

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



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.
- Eclipse will compile your code automatically.

Golden Rules of Programming

1. Compile often
2. Save regularly

Golden Rules of Programming

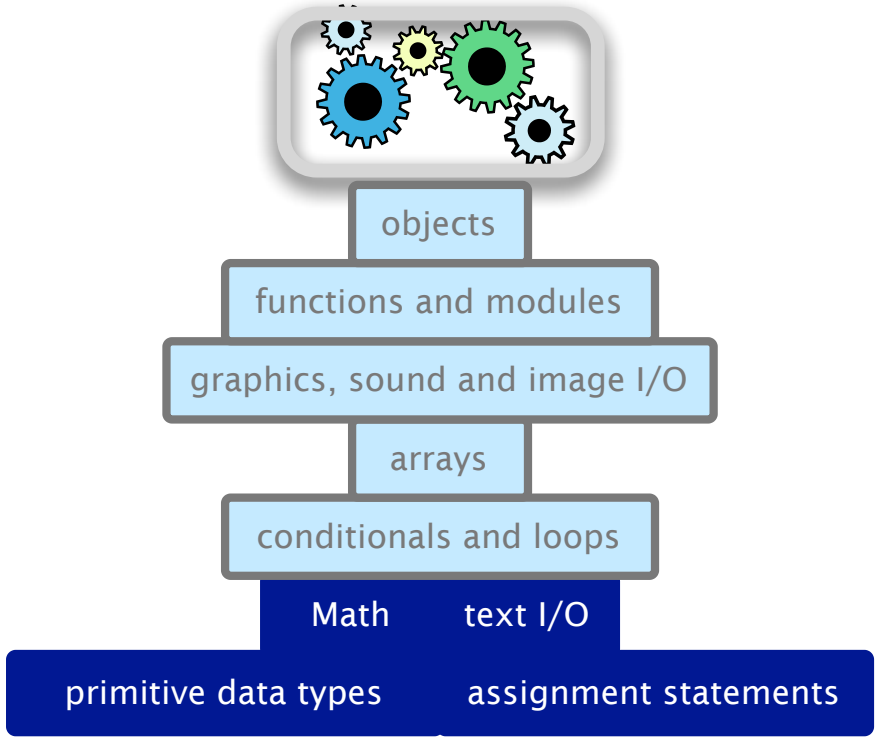
1. Compile often
2. Save regularly

Why? Detect errors early!

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

Basic Functionality

A Foundation for Programming



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 + "?");
   }
}
```

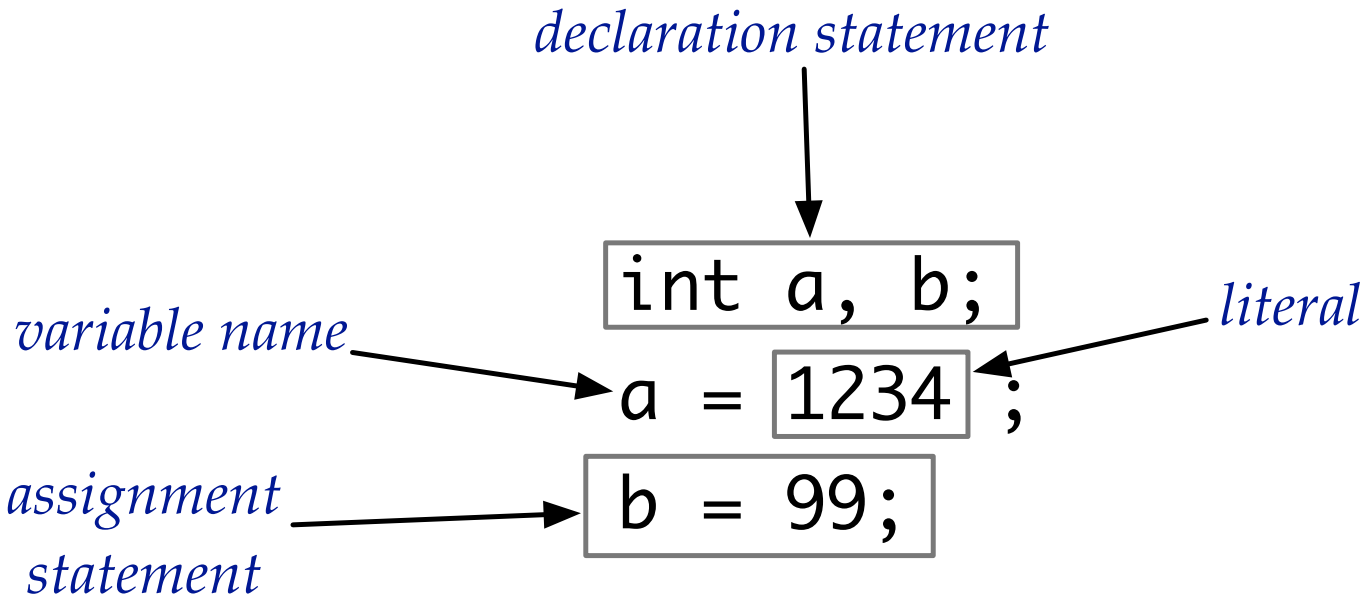
Output

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

Assignment: Basic Definitions

Variable: A name that refers to a value

Assignment Statement: Associates a value with a variable



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: Division

expression	*result type*	*value*
`5 / 2`		

Type Conversion: Division

expression	*result type*	*value*
`5 / 2`	`int`	2

Type Conversion: Division

expression	*result type*	*value*
`5 / 2`	`int`	2
`(double)(5 / 2)`		

Type Conversion: Division

expression	*result type*	*value*
`5 / 2`	`int`	2
`(double)(5 / 2)`	`double`	2.0

Type Conversion: Division

expression	*result type*	*value*
`5 / 2`	`int`	2
`(double)(5 / 2)`	`double`	2.0
`5 / 2.0`	`double`	2.5
`5.0 / 2`	`double`	2.5
`5.0 / 2.0`	`double`	2.5

Type Conversion: Division

expression	*result type*	*value*
`5 / 2`	`int`	2
`(double)(5 / 2)`	`double`	2.0
`5 / 2.0`	`double`	2.5
`5.0 / 2`	`double`	2.5
`5.0 / 2.0`	`double`	2.5

Moral: if you want a floating-point result from division, make at least one of the operands a `double`.

Command-line Arguments

Unix commands

```
mkdir MyJavaCode
```

`mkdir` is a command and `MyJavaCode` is an argument

Command-line Arguments

Unix commands

```
mkdir MyJavaCode
```

`mkdir` is a command and `MyJavaCode` is an argument

Using Java to carry out commands

```
% java Add 3 6
9
```

3 and 6 are command-line arguments for the program `Add`

Command-line Arguments

```
public class Add {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        System.out.println(a + b);
    }
}
```

Command-line Arguments

```
public class Add {
   public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        System.out.println(a + b);
    }
}
```

```
int a = Integer.parseInt(args[0]);
```

- This reads in a string (e.g., "3") from the command line,
- parses it as an int, and
- assigns this as the value of variable a.

Command-line Arguments

Missing an argument

```
% java Add 3
java.lang.ArrayIndexOutOfBoundsException: 1
```

This a run-time error — we didn't provide anything as a value for `args[1]`:

```
int b = Integer.parseInt(args[1]);
```

Summary

Recap: Learning Outcomes

- Use a text editor to create and modify simple Java programs which print strings to a terminal window.
- Use the command-line to compile and run Java programs.
- Declare `int`, `double` and `String` variables and assign values to them.
- Use Java's `main()` method to consume command-line arguments.
- Parse strings into values of type `int` and `double`.
- Carry out simple operations on `int`, `double` and `String` data values.
- Compute fractional results from division with integer values, using casting if necessary.

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 Eclipse, 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.