Programming Environment

Introduction to Compiler Construction

Outline

Outline

Programming Environment

Working with the *j--* Compiler

Working with the *iota* compiler

Outline

Programming Environment

Working with the *j*-- Compiler

Working with the *iota* compiler

Outline

Outline

Outline

Programming Environment

Working with the *j--* Compiler

Working with the *iota* compiler

Programming Environment

Programming Environment

Linux, Mac, or Windows operating system configured with the software needed for the course

Programming Environment

Linux, Mac, or Windows operating system configured with the software needed for the course

Tools we will use:

Programming Environment

Linux, Mac, or Windows operating system configured with the software needed for the course

Tools we will use:

- Visual Studio Code (aka VSCode)

Programming Environment

Linux, Mac, or Windows operating system configured with the software needed for the course

Tools we will use:

- Visual Studio Code (aka VSCode)
- File manager

Programming Environment

Linux, Mac, or Windows operating system configured with the software needed for the course

Tools we will use:

- Visual Studio Code (aka VSCode)
- File manager
- Terminal

Programming Environment

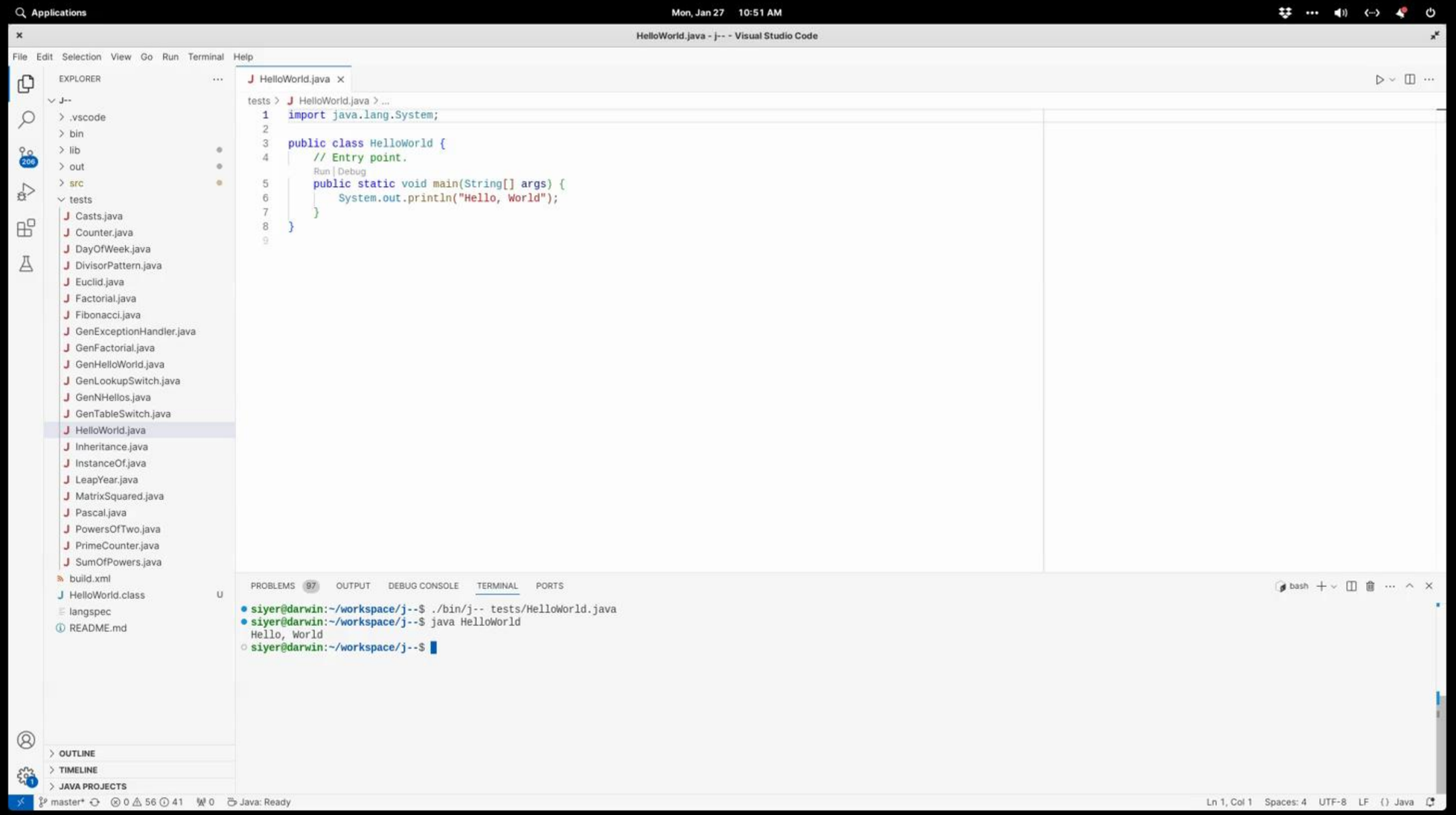
Linux, Mac, or Windows operating system configured with the software needed for the course

Tools we will use:

- Visual Studio Code (aka VSCode)
- File manager
- Terminal
- Web browser

Programming Environment

Programming Environment



Working with the *j*-- Compiler

Working with the *j--* Compiler

Suppose the compiler (`j--.zip`) was downloaded and extracted under `~/workspace` (aka `$j`)

Working with the *j--* Compiler

Suppose the compiler (`j--.zip`) was downloaded and extracted under `~/workspace` (aka `$j`)

Launch VSCode and open the folder `~/workspace/j--`

Working with the *j--* Compiler

Suppose the compiler (`j--.zip`) was downloaded and extracted under `~/workspace` (aka `$j`)

Launch VSCode and open the folder `~/workspace/j--`

To compile the compiler, run the following command in the VSCode terminal

```
$ _
```

Working with the *j--* Compiler

Suppose the compiler (`j--.zip`) was downloaded and extracted under `~/workspace` (aka `$j`)

Launch VSCode and open the folder `~/workspace/j--`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
```

Working with the *j--* Compiler

Suppose the compiler (`j--.zip`) was downloaded and extracted under `~/workspace` (aka `$j`)

Launch VSCode and open the folder `~/workspace/j--`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
Buildfile: ~/workspace/j--/build.xml
...
$ _
```

Working with the *j--* Compiler

Suppose the compiler (`j--.zip`) was downloaded and extracted under `~/workspace` (aka `$j`)

Launch VSCode and open the folder `~/workspace/j--`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
Buildfile: ~/workspace/j--/build.xml
...
$ _
```

To get the usage syntax for the compiler, run

```
$ _
```

Working with the *j--* Compiler

Suppose the compiler (`j--.zip`) was downloaded and extracted under `~/workspace` (aka `$j`)

Launch VSCode and open the folder `~/workspace/j--`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
Buildfile: ~/workspace/j--/build.xml
...
$ _
```

To get the usage syntax for the compiler, run

```
$ ./bin/j--
```

Working with the *j--* Compiler

Suppose the compiler (`j--.zip`) was downloaded and extracted under `~/workspace` (aka `$j`)

Launch VSCode and open the folder `~/workspace/j--`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
Buildfile: ~/workspace/j--/build.xml
...
$ _
```

To get the usage syntax for the compiler, run

```
$ ./bin/j--
Usage: j-- <options> <source file>
Where possible options include:
  -t  Tokenize input and print tokens to STDOUT
  -p  Parse input and print AST to STDOUT
  -pa Pre-analyze input and print AST to STDOUT
  -a  Analyze input and print AST to STDOUT
  -d  <dir> Specify where to place output (.class) files; default = .
$ _
```

Working with the *j*-- Compiler

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ _
```

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ ./bin/j-- tests/HelloWorld.java
```

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ ./bin/j-- tests/HelloWorld.java
$ _
```

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ ./bin/j-- tests/HelloWorld.java
$ _
```

To run the generated JVM program `HelloWorld.class`, run

```
$ _
```

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ ./bin/j-- tests/HelloWorld.java
$ _
```

To run the generated JVM program `HelloWorld.class`, run

```
$ java HelloWorld
```

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ ./bin/j-- tests/HelloWorld.java
$ _
```

To run the generated JVM program `HelloWorld.class`, run

```
$ java HelloWorld
Hello, World
$ _
```

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ ./bin/j-- tests/HelloWorld.java
$ _
```

To run the generated JVM program `HelloWorld.class`, run

```
$ java HelloWorld
Hello, World
$ _
```

To disassemble `HelloWorld.class`, run

```
$ _
```

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ ./bin/j-- tests/HelloWorld.java
$ _
```

To run the generated JVM program `HelloWorld.class`, run

```
$ java HelloWorld
Hello, World
$ _
```

To disassemble `HelloWorld.class`, run

```
$ javap -p -v HelloWorld.class
```

Working with the *j--* Compiler

To compile a *j--* program (eg, `$j/j--/tests/HelloWorld.java`) for the JVM, run

```
$ ./bin/j-- tests/HelloWorld.java
$ _
```

To run the generated JVM program `HelloWorld.class`, run

```
$ java HelloWorld
Hello, World
$ _
```

To disassemble `HelloWorld.class`, run

```
$ javap -p -v HelloWorld.class
Classfile ~/workspace/j--/HelloWorld.class
...
$ _
```

Working with the *iota* Compiler

Working with the *iota* Compiler

Suppose the compiler (`iota.zip`) was downloaded and extracted under `~/workspace`

Working with the *iota* Compiler

Suppose the compiler (`iota.zip`) was downloaded and extracted under `~/workspace`

Launch VSCode and open the folder `~/workspace/iota`

Working with the *iota* Compiler

Suppose the compiler (`iota.zip`) was downloaded and extracted under `~/workspace`

Launch VSCode and open the folder `~/workspace/iota`

To compile the compiler, run the following command in the VSCode terminal

```
$ _
```

Working with the *iota* Compiler

Suppose the compiler (`iota.zip`) was downloaded and extracted under `~/workspace`

Launch VSCode and open the folder `~/workspace/iota`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
```

Working with the *iota* Compiler

Suppose the compiler (`iota.zip`) was downloaded and extracted under `~/workspace`

Launch VSCode and open the folder `~/workspace/iota`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
Buildfile: ~/workspace/iota/build.xml
...
$ _
```

Working with the *iota* Compiler

Suppose the compiler (`iota.zip`) was downloaded and extracted under `~/workspace`

Launch VSCode and open the folder `~/workspace/iota`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
Buildfile: ~/workspace/iota/build.xml
...
$ _
```

To get the usage syntax for the compiler, run

```
$ _
```

Working with the *iota* Compiler

Suppose the compiler (`iota.zip`) was downloaded and extracted under `~/workspace`

Launch VSCode and open the folder `~/workspace/iota`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
Buildfile: ~/workspace/iota/build.xml
...
$ _
```

To get the usage syntax for the compiler, run

```
$ ./bin/iota
```

Working with the *iota* Compiler

Suppose the compiler (`iota.zip`) was downloaded and extracted under `~/workspace`

Launch VSCode and open the folder `~/workspace/iota`

To compile the compiler, run the following command in the VSCode terminal

```
$ ant
Buildfile: ~/workspace/iota/build.xml
...
$ _
```

To get the usage syntax for the compiler, run

```
$ ./bin/iota
Usage: iota <options> <source file>
Where possible options include:
  -g  Allocate registers using graph coloring method; default = naive method
  -v  Display intermediate representations and liveness intervals
  -d  <dir> Specify where to place output (.marv) file; default = .
$ _
```

Working with the *iota* Compiler

Working with the *iota* Compiler

To compile an *iota* program (eg, `$j/iota/tests/Factorial.iota`) for the Marvin machine, run

```
$ _
```

Working with the *iota* Compiler

To compile an *iota* program (eg, `$j/iota/tests/Factorial.iota`) for the Marvin machine, run

```
$ ./bin/iota tests/Factorial.iota
```

Working with the *iota* Compiler

To compile an *iota* program (eg, `$j/iota/tests/Factorial.iota`) for the Marvin machine, run

```
$ ./bin/iota tests/Factorial.iota
$ _
```

Working with the *iota* Compiler

To compile an *iota* program (eg, `$j/iota/tests/Factorial.iota`) for the Marvin machine, run

```
$ ./bin/iota tests/Factorial.iota
$ _
```

To run the generated Marvin program `Factorial.marv`, run

```
$ _
```

Working with the *iota* Compiler

To compile an *iota* program (eg, `$j/iota/tests/Factorial.iota`) for the Marvin machine, run

```
$ ./bin/iota tests/Factorial.iota
$ _
```

To run the generated Marvin program `Factorial.marv`, run

```
$ python3 ./bin/marvin.py Factorial.marv
```

Working with the *iota* Compiler

To compile an *iota* program (eg, `$j/iota/tests/Factorial.iota`) for the Marvin machine, run

```
$ ./bin/iota tests/Factorial.iota
$ _
```

To run the generated Marvin program `Factorial.marv`, run

```
$ python3 ./bin/marvin.py Factorial.marv
_
```

Working with the *iota* Compiler

To compile an *iota* program (eg, `$j/iota/tests/Factorial.iota`) for the Marvin machine, run

```
$ ./bin/iota tests/Factorial.iota
$ _
```

To run the generated Marvin program `Factorial.marv`, run

```
$ python3 ./bin/marvin.py Factorial.marv
5
```

Working with the *iota* Compiler

To compile an *iota* program (eg, `$j/iota/tests/Factorial.iota`) for the Marvin machine, run

```
$ ./bin/iota tests/Factorial.iota
$ _
```

To run the generated Marvin program `Factorial.marv`, run

```
$ python3 ./bin/marvin.py Factorial.marv
5
120
$ _
```