

# Object Oriented Programming

Week 1 Part 3  
Input and Output

# Lecture

- Output to the console
- Input from the console
- Creating a program to run from the console

# Output to the Console

# Java Output

- Printing output to the console
  - `System.out.println`
    - writes a string then newline on console
  - `System.out.print`
    - Writes a string on the console
  - `System.out.format`
    - Formats a string, like C, and writes it on the console

# Example

```
public class PrintDemo {  
    public static void main(String[] args) {  
        int i = 2;  
        double r = Math.sqrt(i);
```

Prints multiple strings

```
        System.out.print("The square root of ");  
        System.out.print(i);  
        System.out.print(" is ");  
        System.out.print(r);  
        System.out.println(".");
```

Prints single string made with +

```
        System.out.println("The square root of " + i + " is " + r + ".");
```

Creates and prints from format

```
        System.out.format("The square root of %d is %f.%n", i, r);
```

```
    }  
}
```

Output

```
The square root of 2 is 1.4142135623730951.  
The square root of 2 is 1.4142135623730951.  
The square root of 2 is 1.414214.
```

# Explanation

- System is a class in the java.lang package
  - It is imported by default
- System has a member variable out.
  - “out” is an object of class PrintWriter
  - A PrintWriter has methods
    - “print”
    - “println”
    - “format”

# Input from the Console

# Java Input

- Printing in
  - System.in
    - An input stream attached to the keyboard
  - Scanner
    - An object that contains methods to read from input stream
  - Scanner methods
    - “next()”: reads the characters up to the first white space
    - “hasNext()”: true if there is another string
      - Needs ctrl-D to indicate end of stream
    - “nextline()”: skips up to the next line



# Reading Input

# Input Example

Need to import Scanner Class

```
package scannerDemo;  
  
import java.util.Scanner;
```

Create Scanner object scan

```
public class ScannerDemo {  
  
    public static void main(String args[]) {  
        Scanner scan = new Scanner(System.in);  
        System.out.print("Enter your name: ");  
        String name = scan.next();  
        System.out.println("Hello " + name);  
        scan.close();  
    }  
}
```

Use Scanner object to read string

Close Scanner object to free resources

Output 1

```
Enter your name: Nat  
Hello Nat
```

Output 2: reads to space

```
Enter your name: Nat Martin  
Hello Nat
```

# Explanation

- A Scanner is a class that has an input stream as a member variable.
- It reads tokens from the stream such as integers, floating point number, and strings.
  - The tokens are separated by white space.
- It reads until get gets to the end of file
  - End of file is indicated by ctrl-d

# Input Example 2

Make new scanner from System.in

```
public static void main(String args[]) {  
    Scanner scan = new Scanner(System.in);  
    System.out.print("Enter your name: ");
```

Create array of two strings

```
    String name[] = {"first", "second"};  
    int i = 0;
```

Get strings until EOF (ctrl-d)

```
    while (scan.hasNext() && i < 2) {  
        name[i++] = scan.next();  
    }
```

Print first two strings

```
    System.out.print("Hello ");  
    i = 0;  
    while (i < 2) {  
        System.out.print(name[i++] + " ");  
    }  
    System.out.println("");  
    scan.close();  
}
```

Output

```
Enter your name: Nat Martin  
Hello Nat Martin  
|
```

# Explanation

- In Java, the number of elements in an array are set by the initialization
  - The array can hold enough to hold the initial array.
  - Adding more causes an exception (i.e. the program crashes)
- The scanner reads until it sees ctrl-d.
  - The array is protected by checking to see that we have not added more than two elements.
- The program prints two items from the array.

# Running Programs from the Console

# Executing Programs on the Console

- So far we have only run our programs in the IDE
- Java translates programs into an intermediate language which is interpreted by the Java Runtime Enviroment (JRE)
  - The JRE translates each intermediate language statement into machine language and executes it.
  - Slower, but easier to move to new machines.

# Executing Java

- To execute a java program called “Test.java” we
  - First compile it into an intermediate code
    - `javac Test.java`
  - Second we run it by passing the compiled code to it
    - `Java Test`



# Hello World Example

```
test — ubuntu@ip-172-31-23-72: /opt/git — nano — 80x24
GNU nano 2.0.6      File: Test.java

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

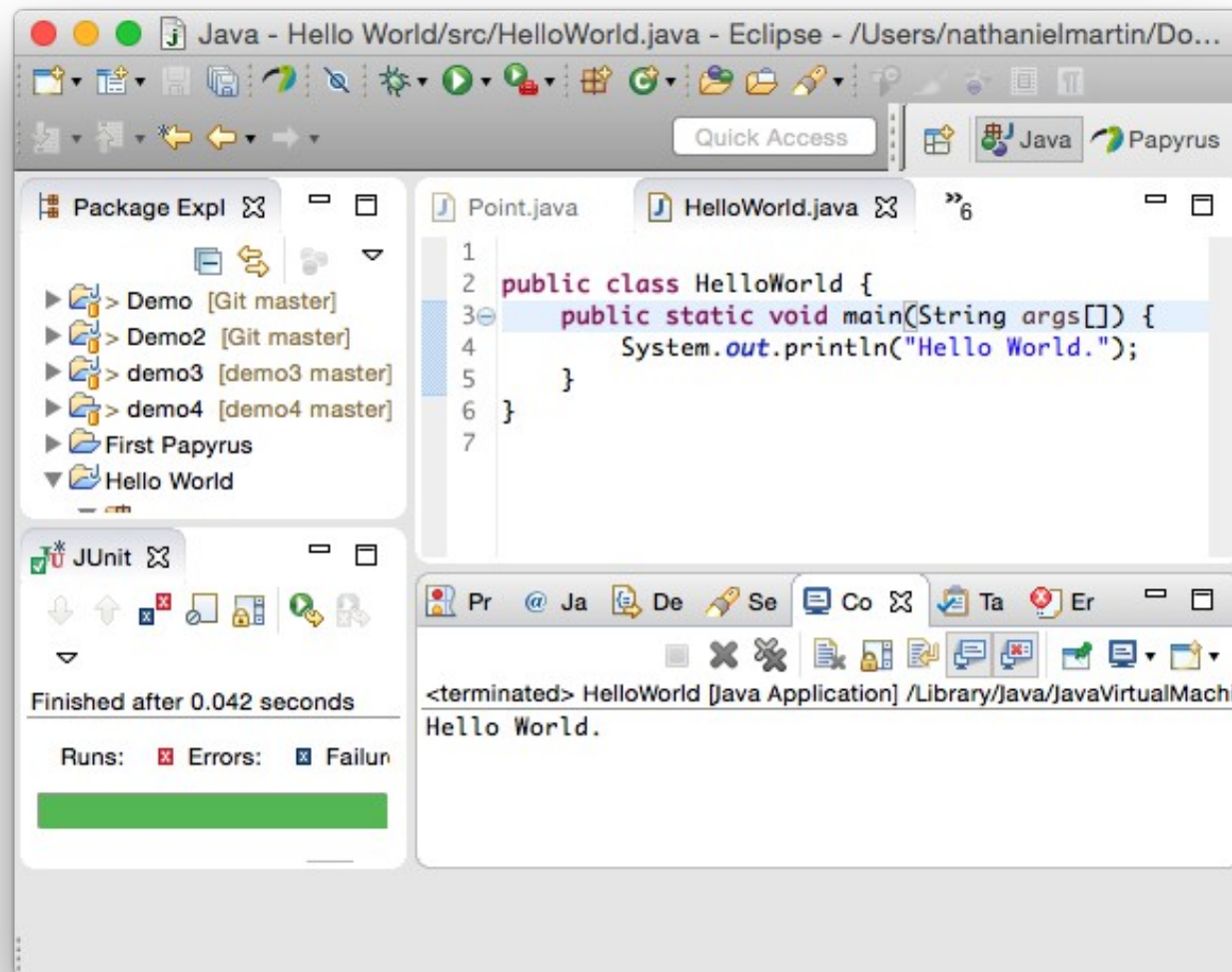
[ Read  
^G Get Help ^O WriteOut ^R Read File  
^X Exit ^J Justify ^W Where Is

```
test — ubuntu@ip-172-31-23-72: /opt/git — bash — 80x24
test> nano Test.java
test> ls
Test.java
test> javac Test.java
test> ls
Test.class      Test.java
test> java Test
Hello world.
test>
```

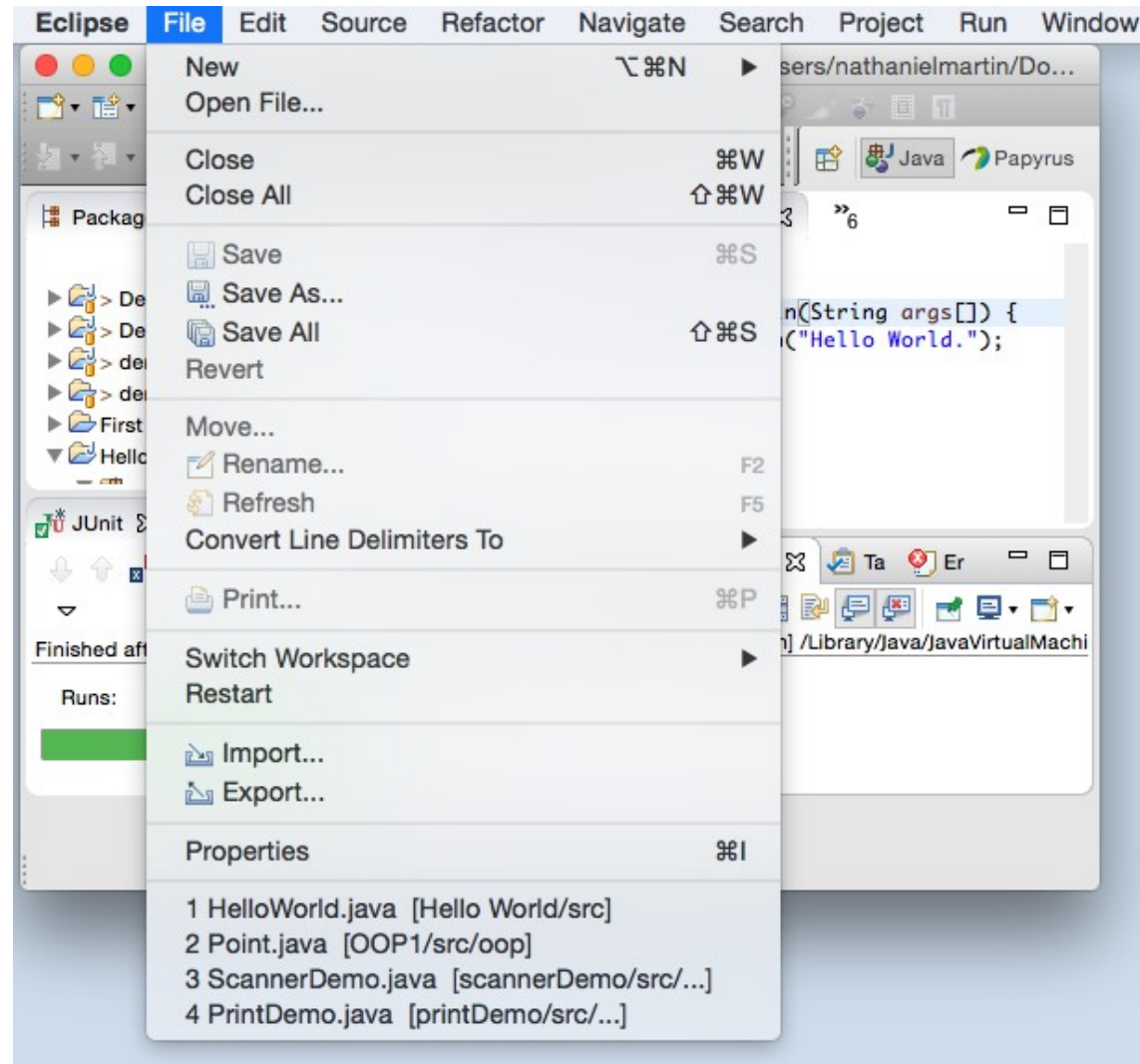
# Making a Real Program

- As with C, compiling a real Java program involves combining multiple files and libraries.
- It is easiest to compile the files using the IDE
- To do so, you create a “.jar” file that contains compiled libraries and files needed to run the program
  - Jar stands for Java ARchive
- Let's do the Hello World program first.
- Then we'll compile the program from last week

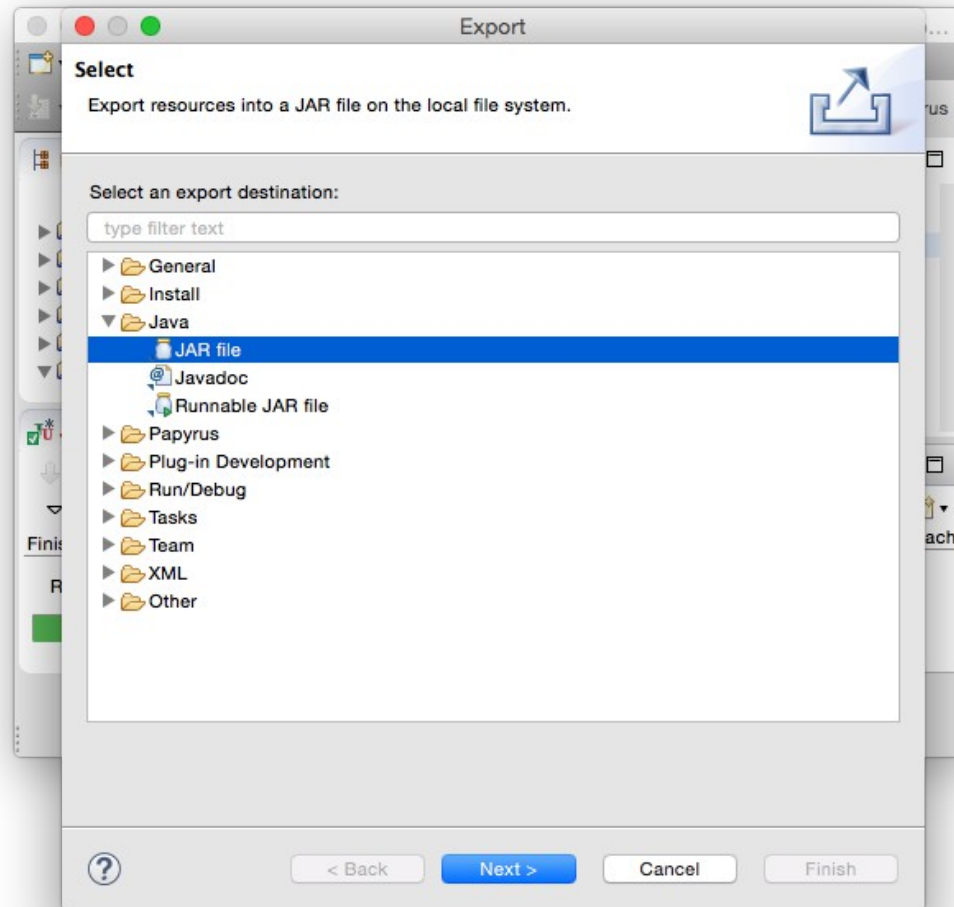
# Write and Run the Program



# Select File > Export



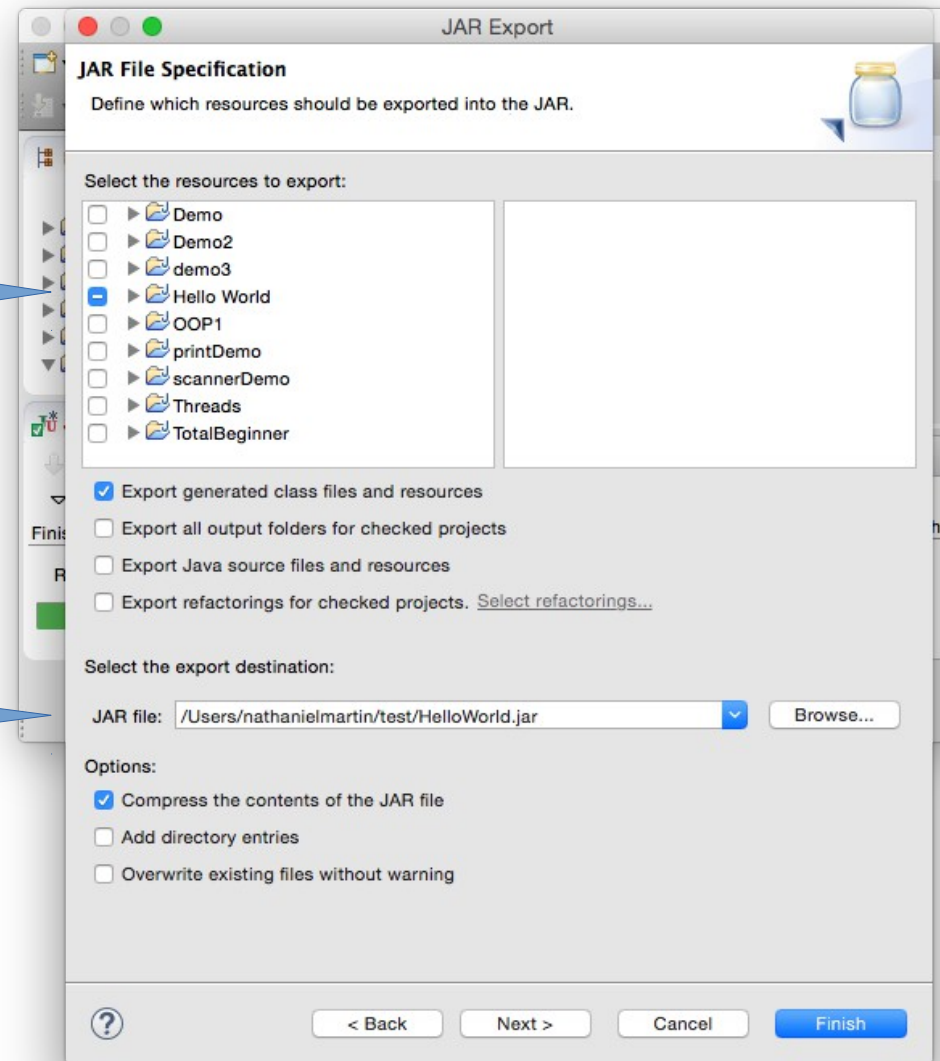
# Choose JAR File



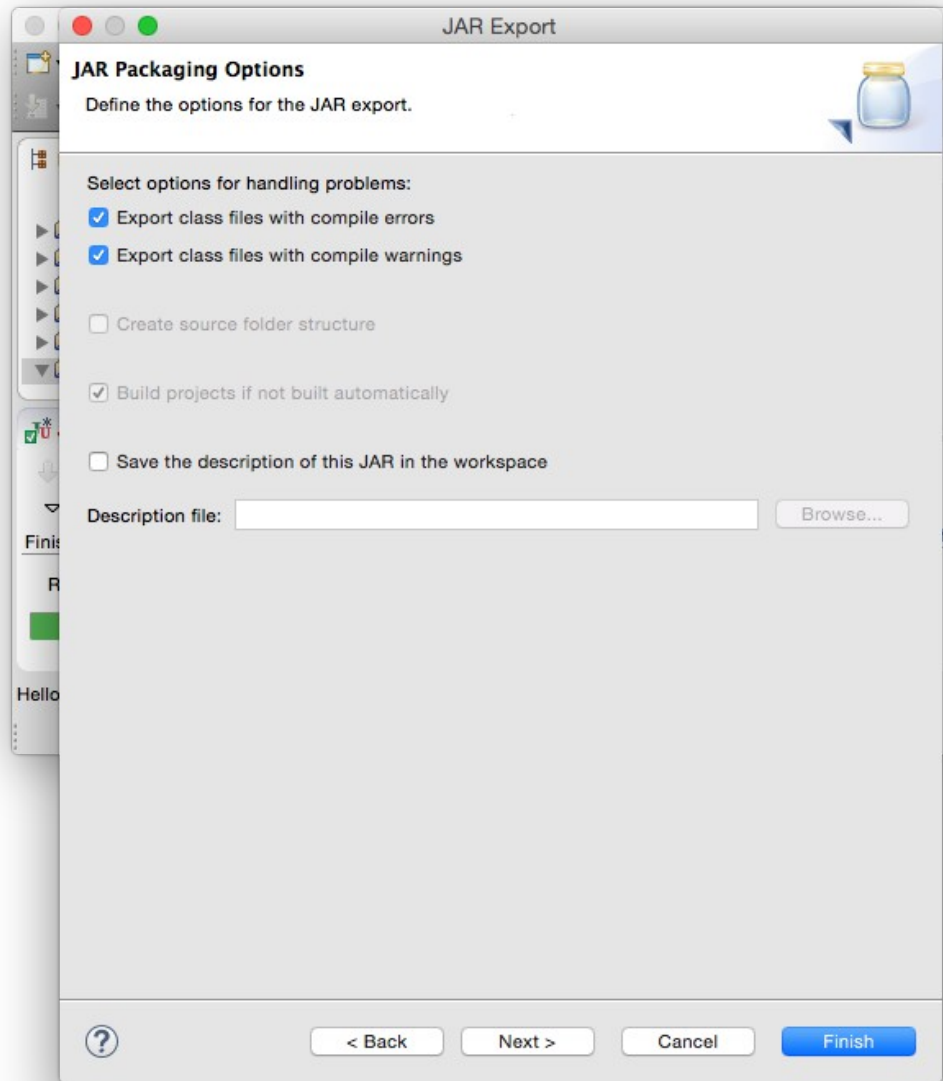
# Select Project and Output Location

Hello World project

To ~/test/HelloWorld.jar



# Click “Next” for defaults

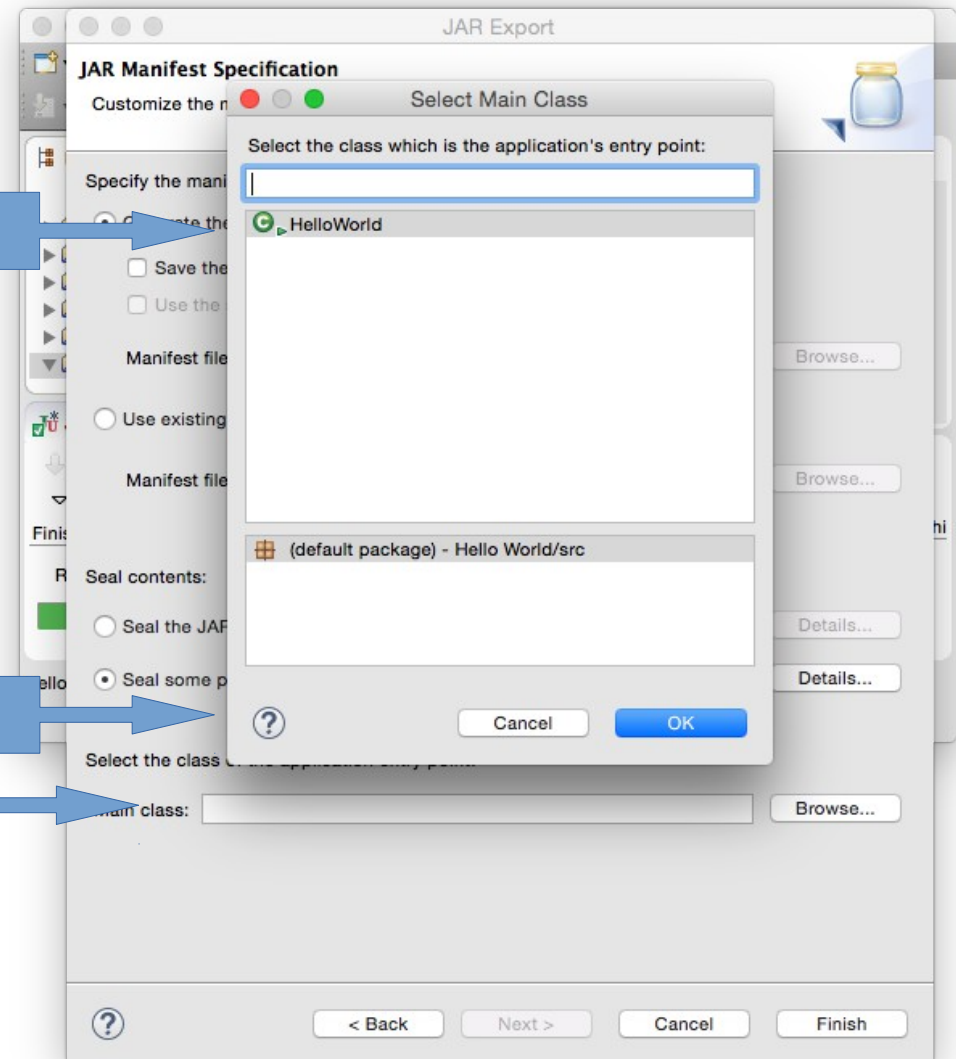


# Choose which “main” to run

2. Only possibility is HelloWorld

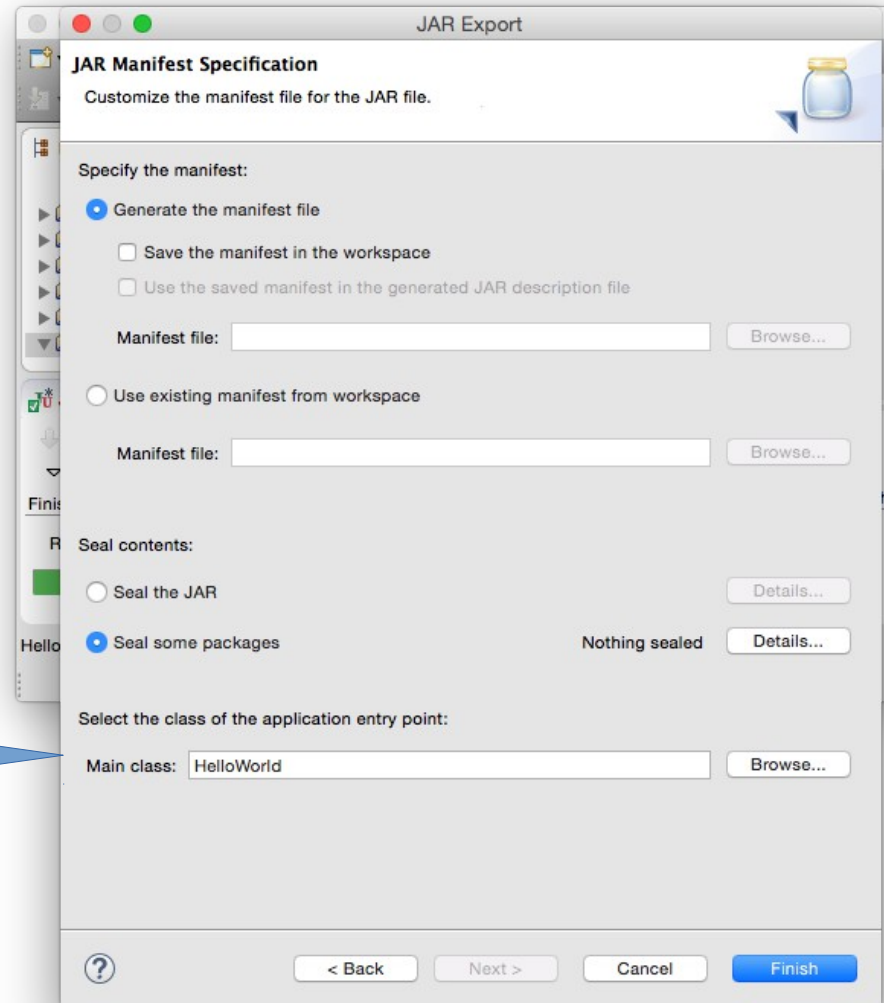
3. Click “OK”

1. Click Browse to find possibilities



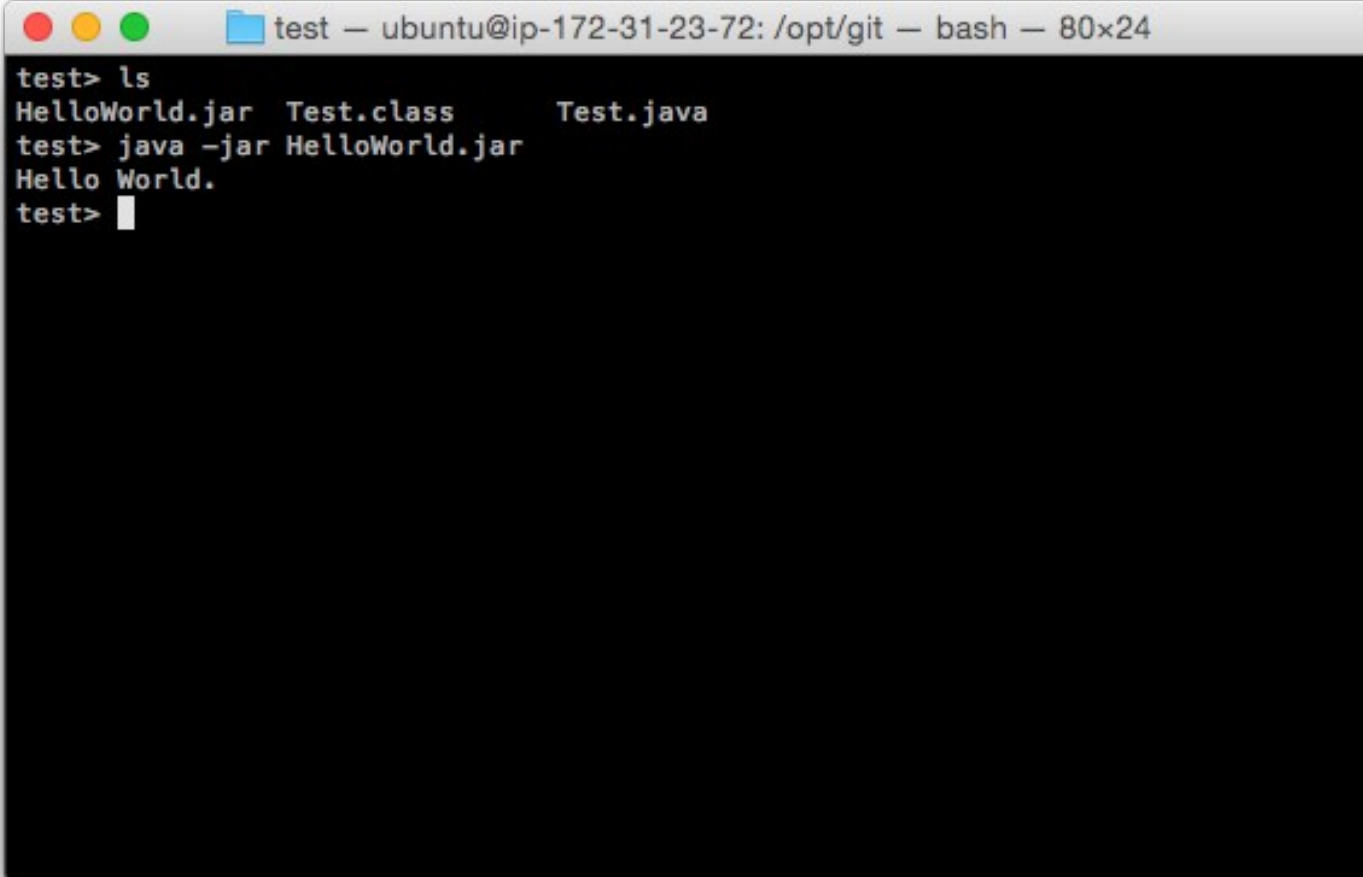


# Finish



HelloWorld main()

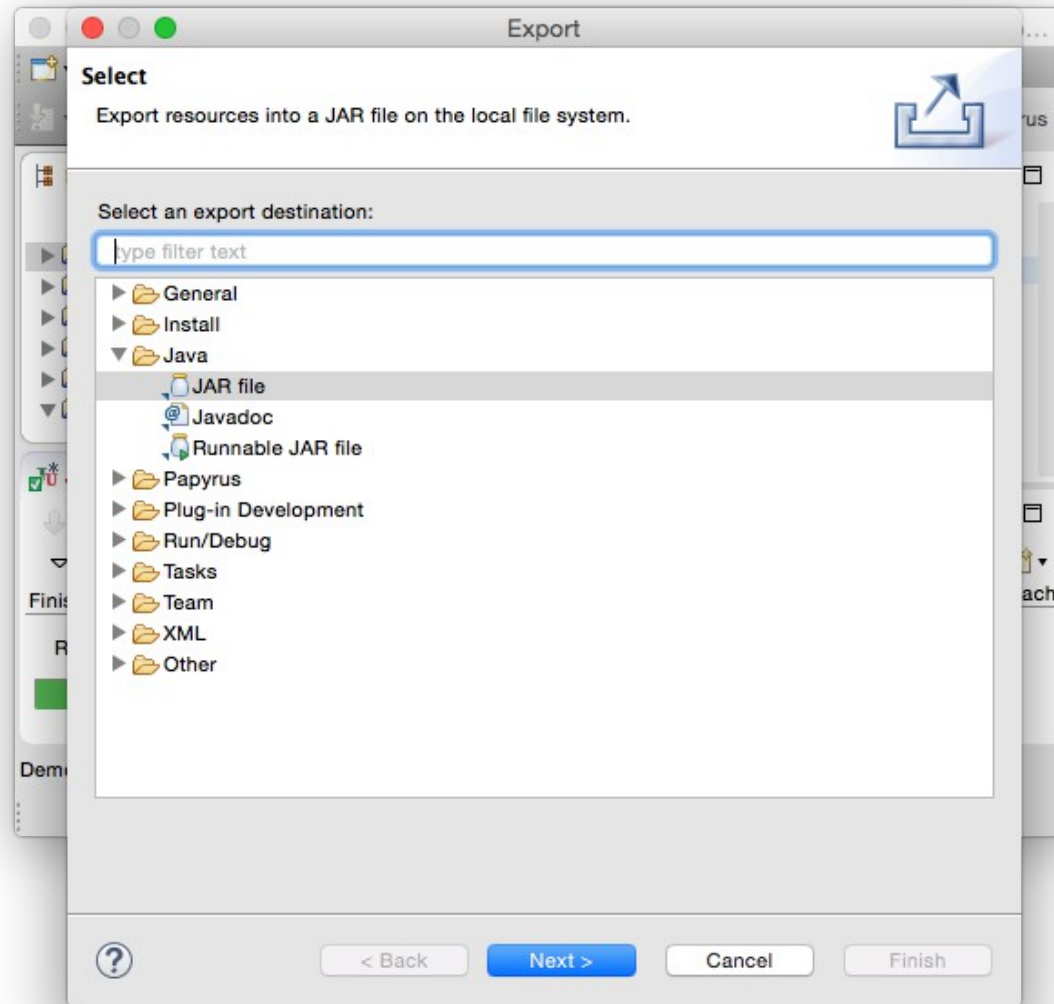
# “java -jar HelloWorld.jar” to Run

A terminal window with a title bar that reads "test — ubuntu@ip-172-31-23-72: /opt/git — bash — 80x24". The terminal has a black background with white text. The user has entered the following commands: "test> ls", "test> java -jar HelloWorld.jar", and "test> ". The output of the first command is "HelloWorld.jar Test.class Test.java". The output of the second command is "Hello World.". The cursor is at the end of the third command line.

```
test> ls
HelloWorld.jar Test.class Test.java
test> java -jar HelloWorld.jar
Hello World.
test> 
```

# Running Point class from Console

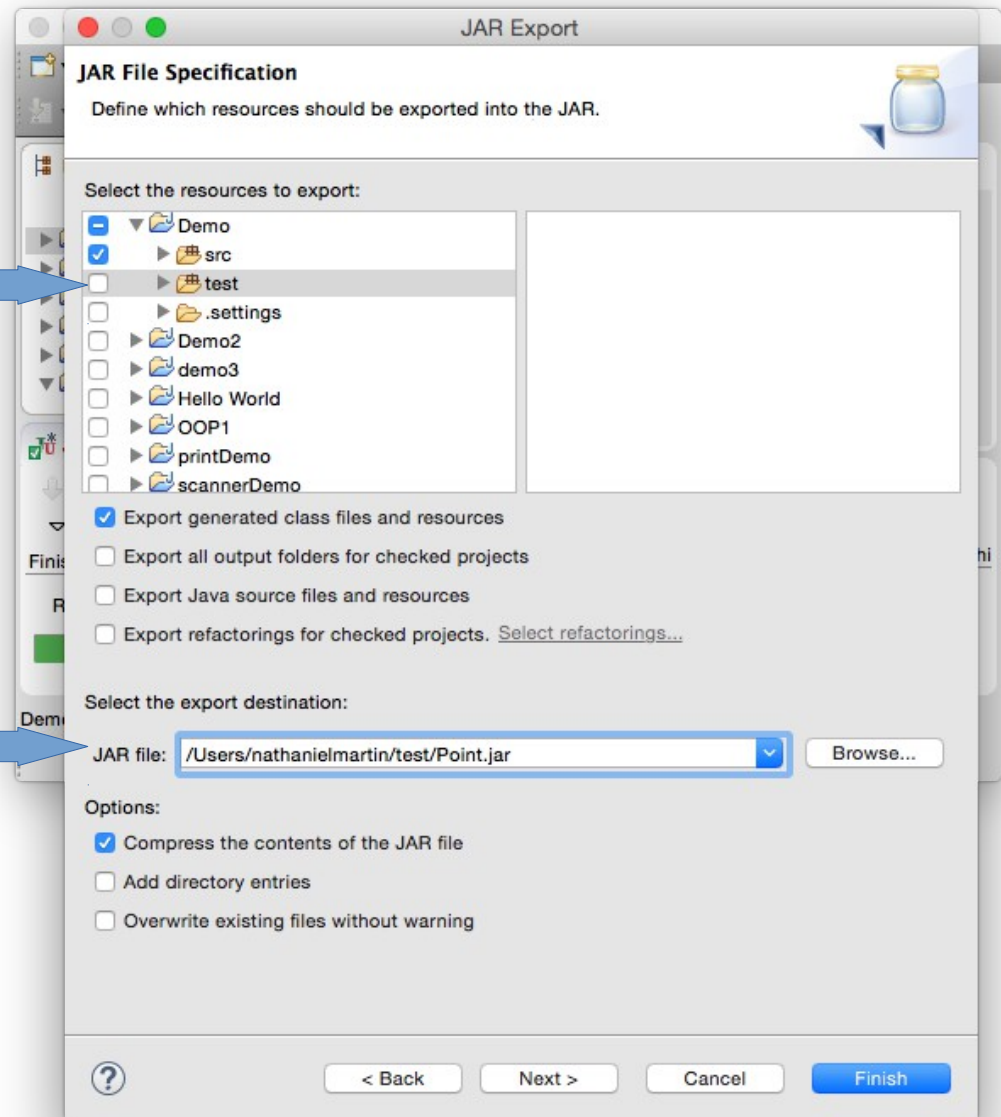
# Export as jar



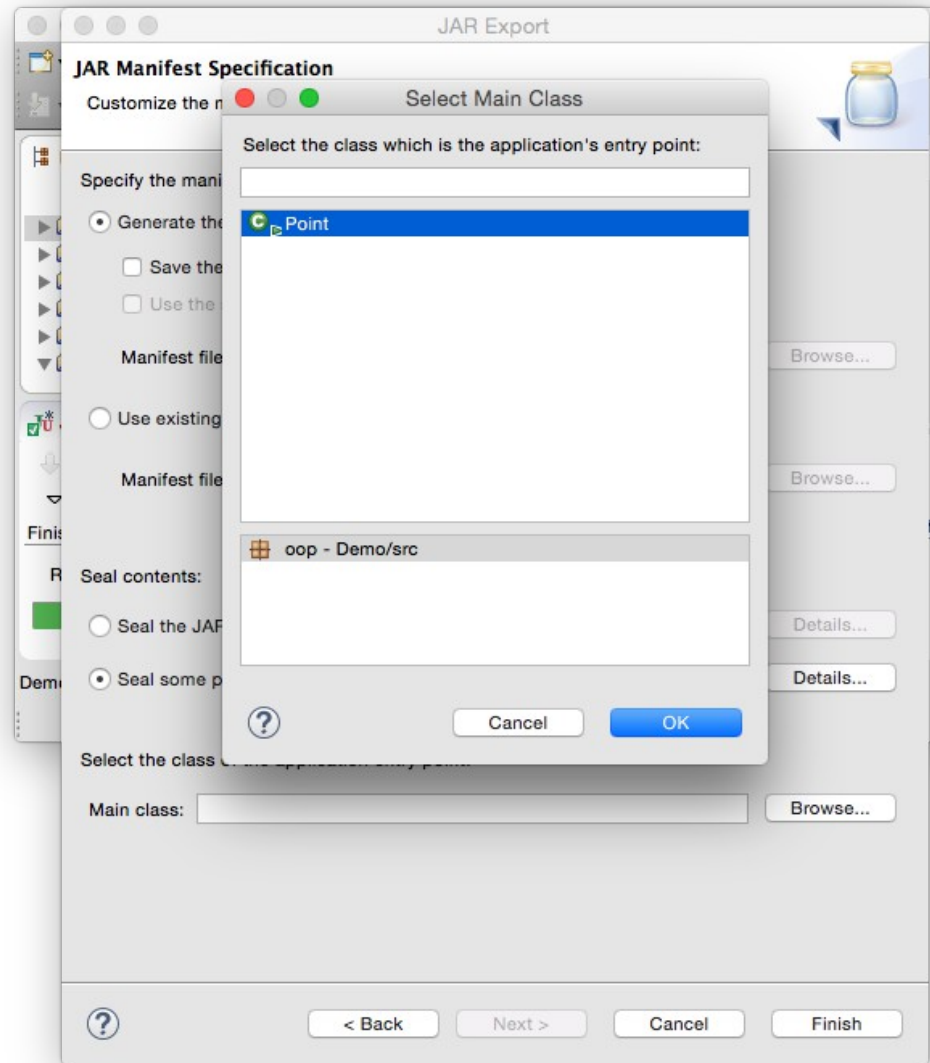
# Select only “src”

Deselect Tests (We don't need it)

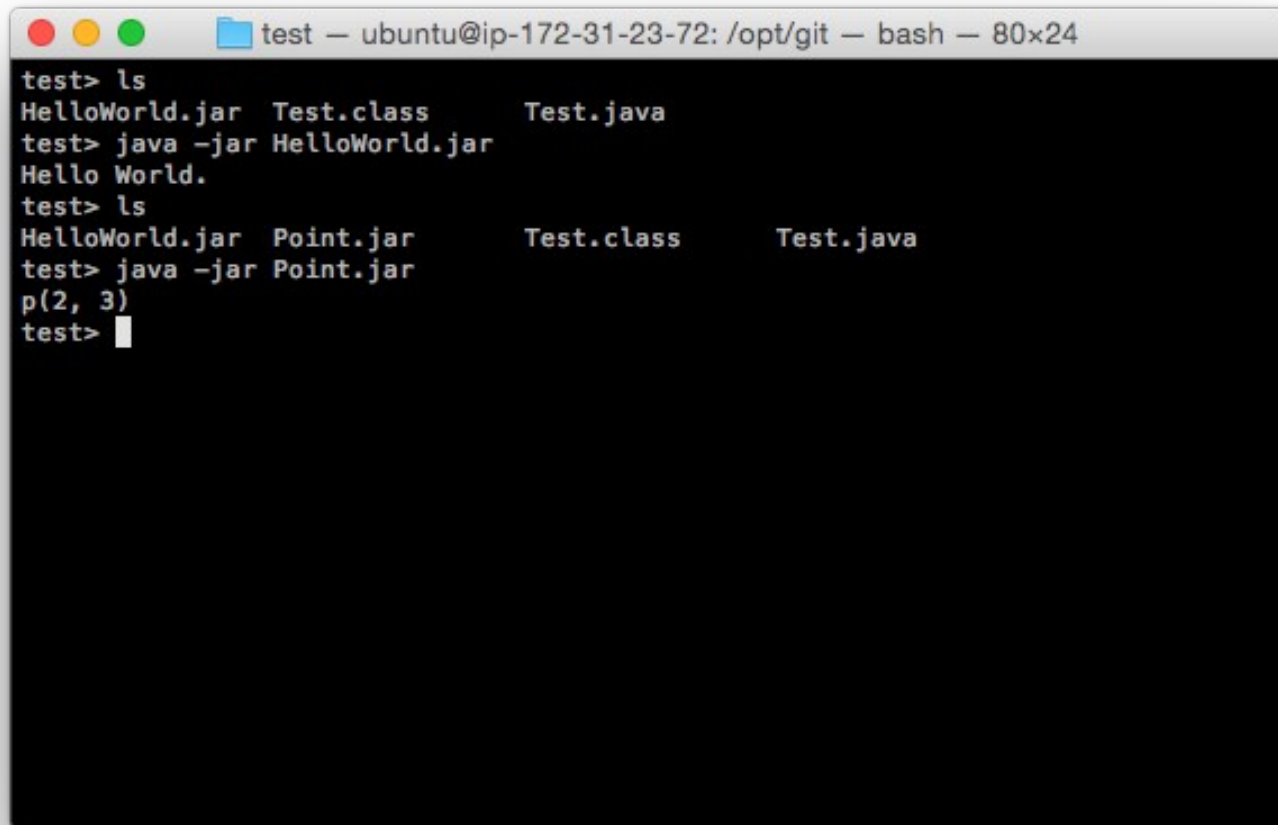
Give it a new name



# Tell it where main() is



# Run it



```
test — ubuntu@ip-172-31-23-72: /opt/git — bash — 80x24
test> ls
HelloWorld.jar  Test.class      Test.java
test> java -jar HelloWorld.jar
Hello World.
test> ls
HelloWorld.jar  Point.jar      Test.class      Test.java
test> java -jar Point.jar
p(2, 3)
test> 
```