Lecture 2: Tools

January 13, 2006

Announcements

- About assignments
  - I'm going to be handing back grades on the Wiki
    - No, editing them won't change your grade
  - You can see files you've turned in

- About slides
  - Slides will be available before class on the **Schedule** page

Announcements

- About readings
  - Readings should be done before class on the assigned day
  - Should be complementary to lectures

- You should
  - Be sure to sign up on the Wiki (most have)
  - Be sure to hand in Assignment 0 before next Wed.
    - Before 3am next Wed, to be precise

Overview

- **Today**:
  1. Some terms you've asked about
  2. Overview of how programs actually run on your computer
  3. Installing Java and Eclipse
  4. Writing and running a simple Java program with Eclipse

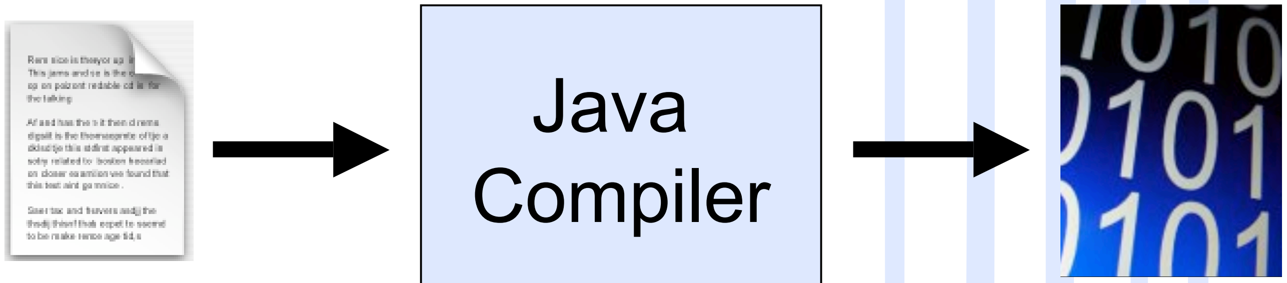
- Still orientation!
  - First graded assignment is next week

Terms

- People are asking me about these
  - Not central to the course, but just so you know what you downloaded, I've included them here.
- **JRE**
  - Java Runtime Environment
  - stuff you need to **run** java programs
- **SDK**
  - Software Development Kit
  - stuff you need to **write** programs
- **JDK**
  - Java Development Kit
  - stuff you need to write **Java** programs

What’s a Java program?

- Starts out as a **text file** (made by you)
  - These use the suffix *.java*
- Feed this to a *Compiler*
- Compiler chews it up and spits out a **class file**
  - Think of this as a version of your text file that is understandable by a machine



Text File

Class File

`Something.java`

`Something.class`

What about other languages

- Java is a *compiled* language
  - needs to be processed by a compiler to create code the machine can run
- Some languages don't need this step
  - Can be translated as they're run
- Java is a little different
  - Compiled code (class files) can run on lots of different machines
  - Languages like C and C++ are compiled, but their compiled code can only run on one kind of machine
  - Details of this are beyond this course

What’s the compiler doing, really?

- Machines don’t understand text files
  - Not even if they contain Java code!

- Compilers translate from Java to machine language
  - Machine language is nasty, hard to understand
  - Causes anxiety, premature baldness

- Compilers
  - Let us write code we can better understand (Java)
  - Translate to machine code for us
  - Tell us if there are syntax errors in our code
    - i.e. whether you used bad grammar
    - This helps us find bugs, but won’t find them all

So what do I need to know?

- Process of writing a Java program:
  1. Write a text file (.java)
  2. Compile it
     - Generate .class file
  3. Run it
     - Cross fingers, hope you don't have bugs

- You can do all these things in Eclipse

- What if I have bugs?
  - Back to the drawing board
  - Repeat above process

Syntax vs Semantics

- **Syntax**
  - Whether your code is a valid Java program
  - Did you use good "grammar"?
  - Compiler checks this for you
    - Yells at you if you said something nonsensical

- **Semantics**
  - Whether your code "does the right thing"
  - Need to run the program to check this

Ok, let's get our hands dirty

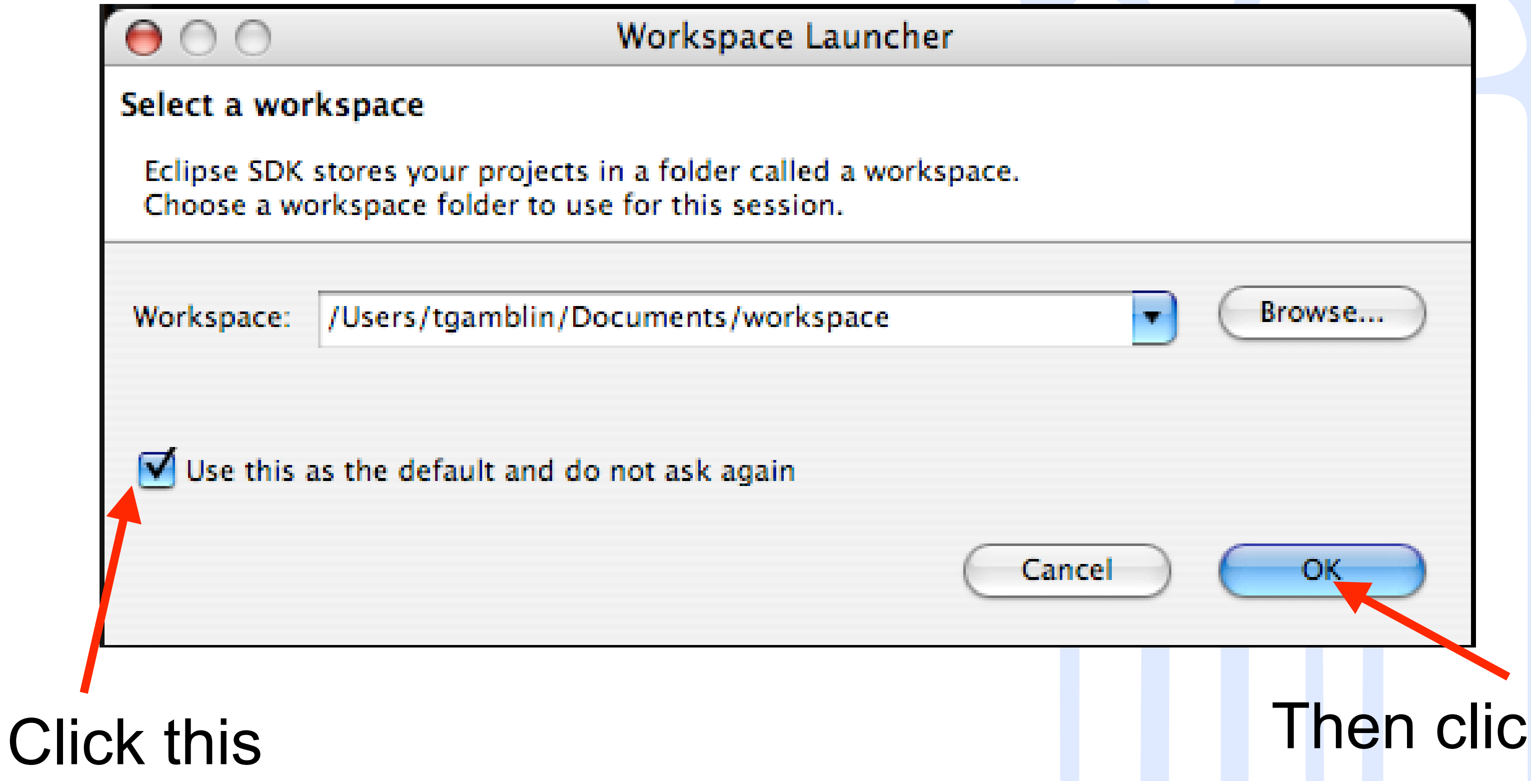
1. Download Java
   – http://java.sun.com/j2se/1.5.0/download.jsp

2. Download Eclipse
   – http://eclipse.org/downloads

3. Install both, and get Eclipse running

Launch Eclipse

- Asks you for a workspace
  - That's where it puts your files



Select a workspace

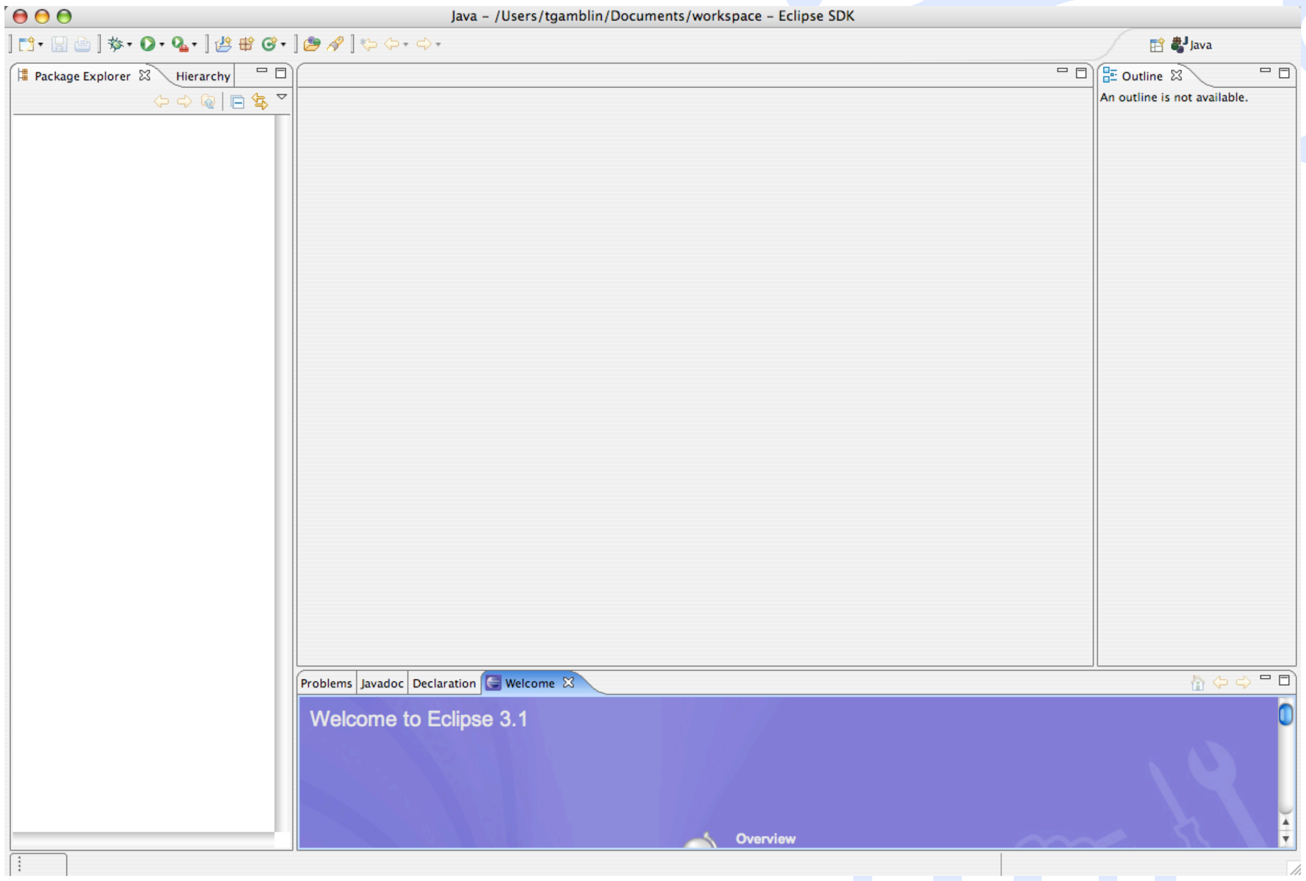
Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /Users/tgamblin/Documents/workspace

[✓] Use this as the default and do not ask again

Launch Eclipse

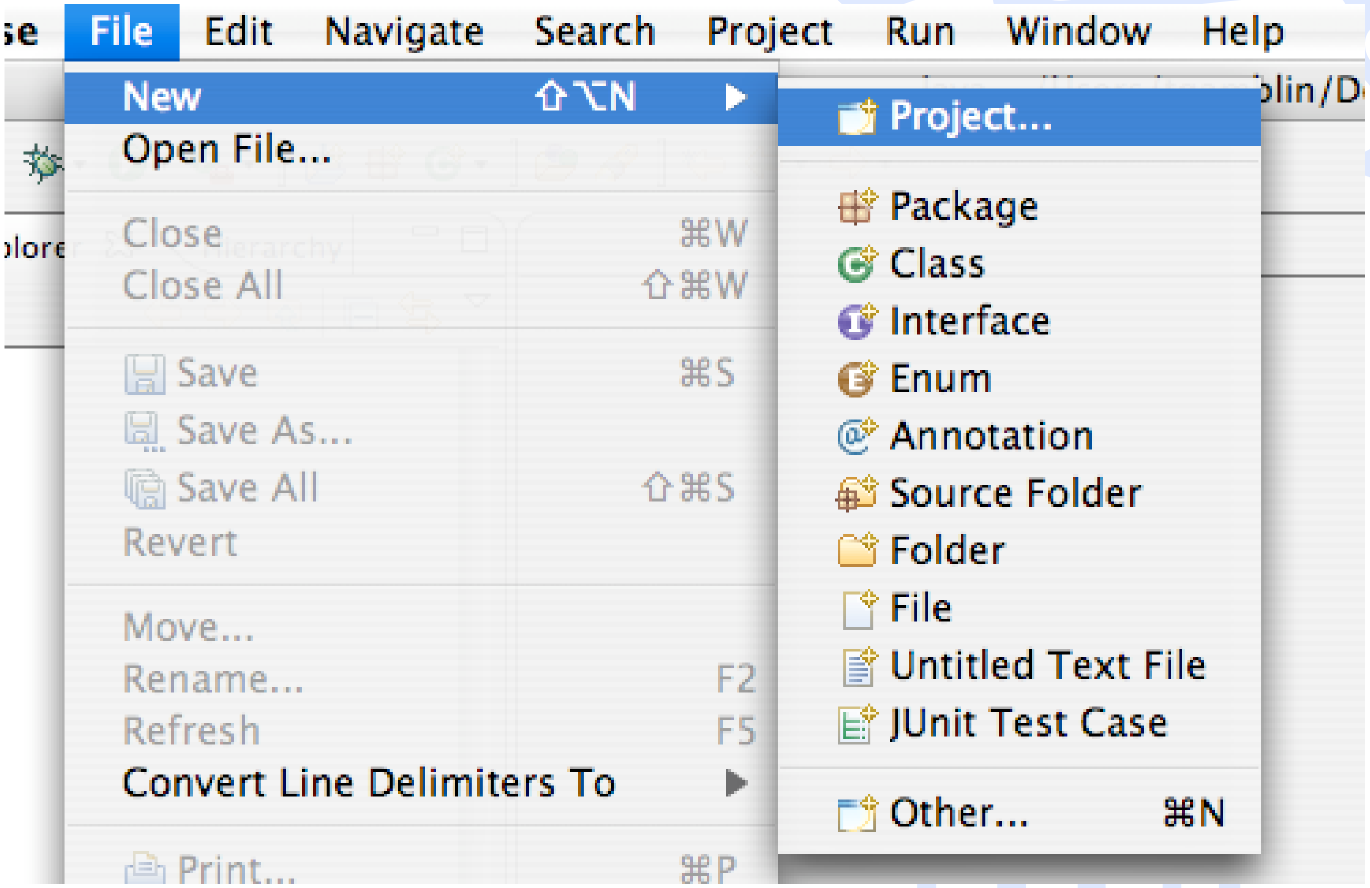
- You'll see this:



An outline is not available.

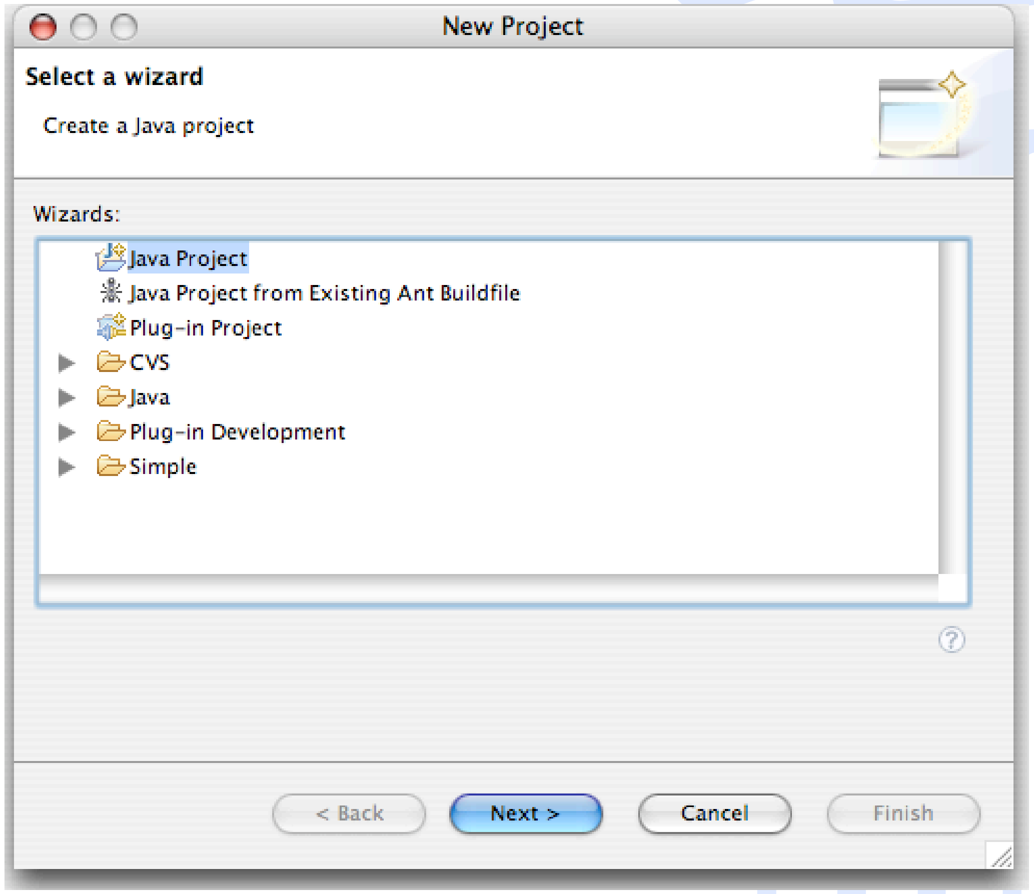
Make a new project

- Use file menu:



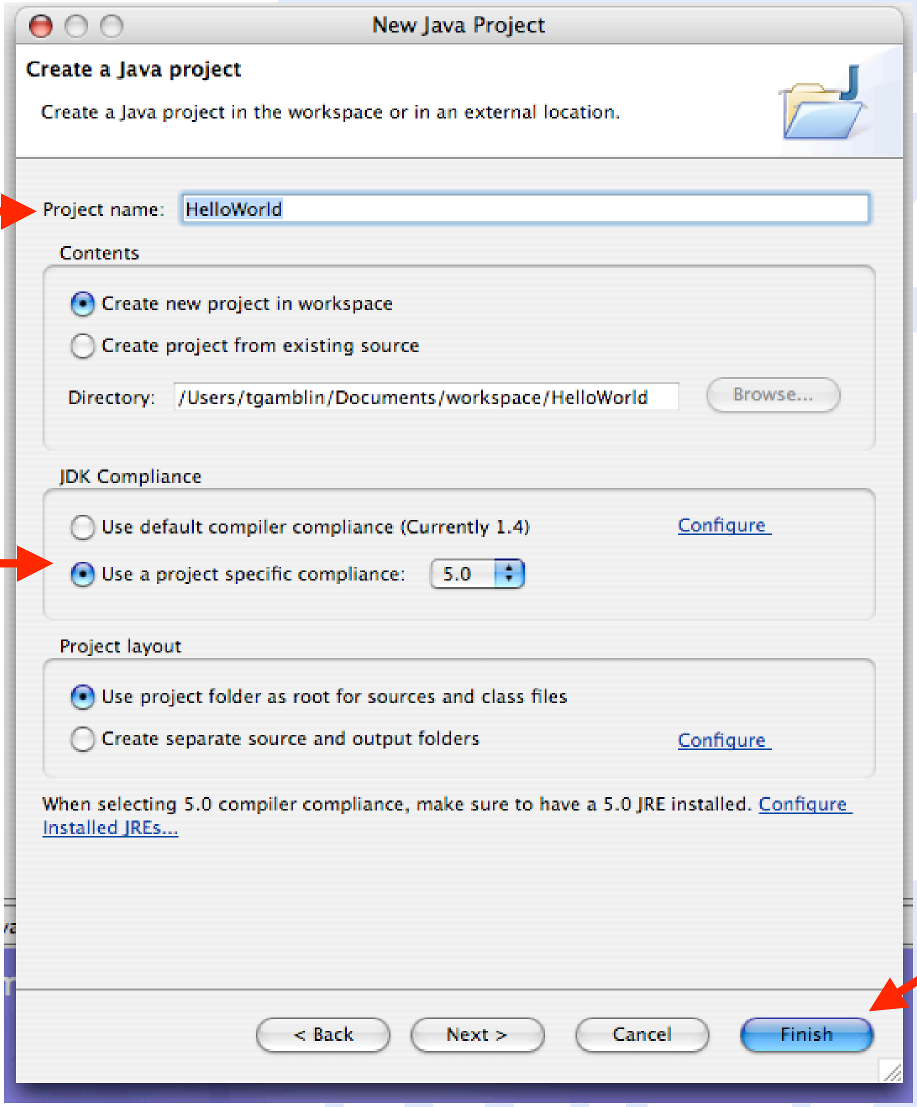
Make a new project

- Pick “Java Project”



Give it a name...

- Type "HelloWorld" for the name
- Select JDK 5.0 compliance
  - Don't worry about what that means
- Click "Finish"



New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: HelloWorld

Contents

- [✓ Create new project in workspace]
- Create project from existing source

Directory: /Users/tgamblin/Documents/workspace/HelloWorld

JDK Compliance

- Use default compiler compliance (Currently 1.4)
- [✓ Use a project specific compliance: 5.0]

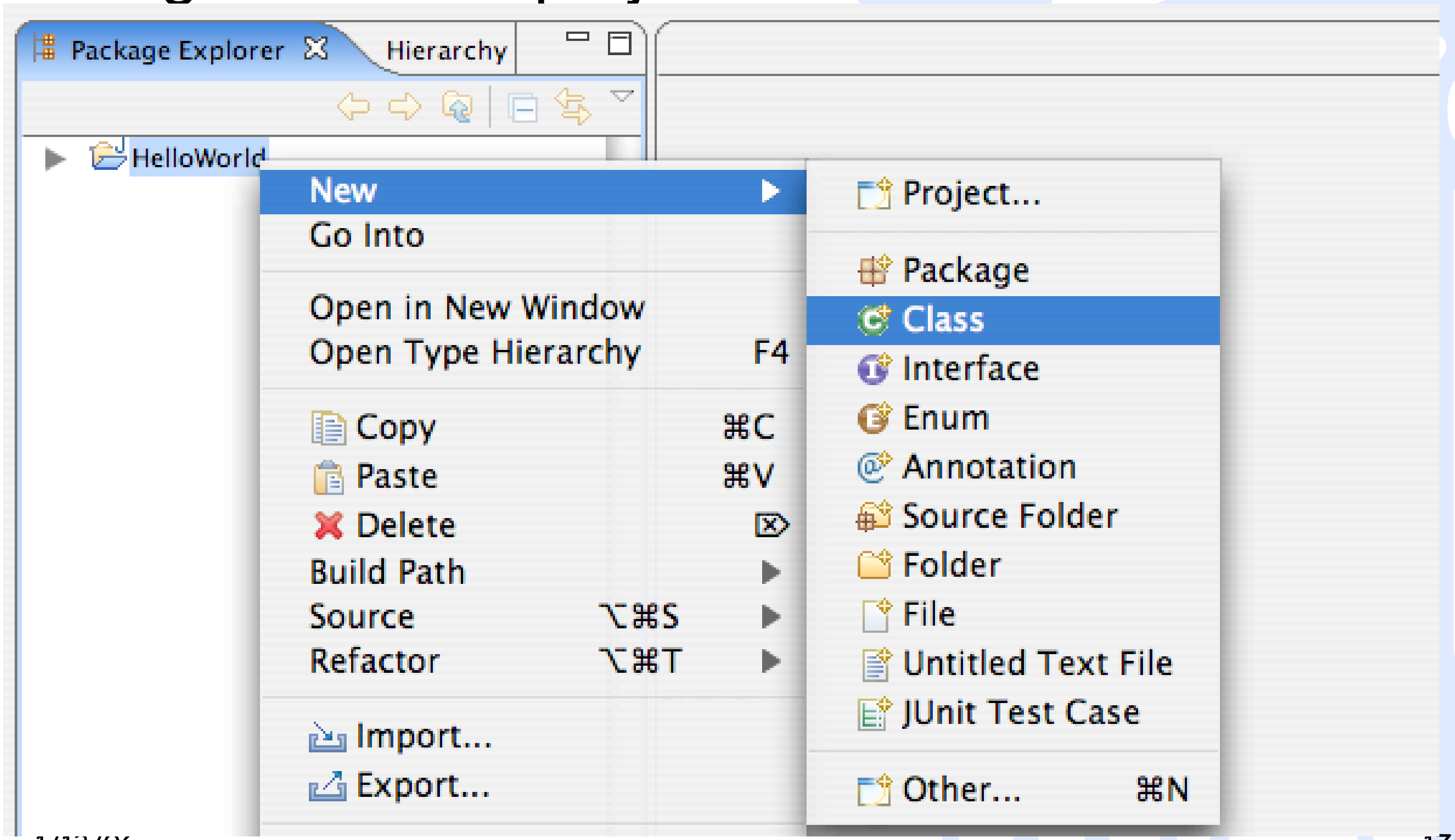
Project layout

- [✓ Use project folder as root for sources and class files]
- Create separate source and output folders

When selecting 5.0 compiler compliance, make sure to have a 5.0 JRE installed. Configure Installed JREs...

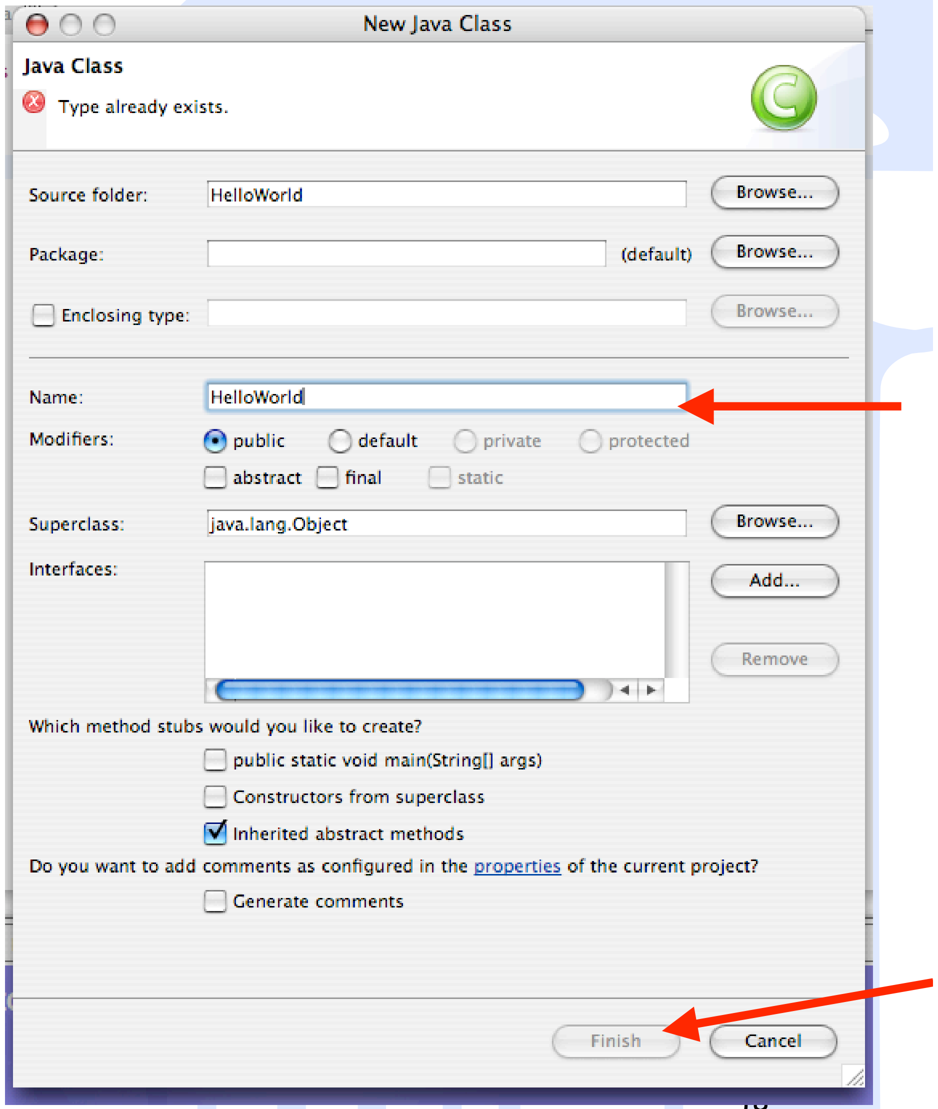
Make a new class

- Right-click the project folder:



Make a new class

- You only need to worry about the name
  - Call it “HelloWorld” too
- Click “Finish”



New Java Class

Java Class

Type already exists.

Source folder: HelloWorld

Package: (default)

☐ Enclosing type:

Name: HelloWorld

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

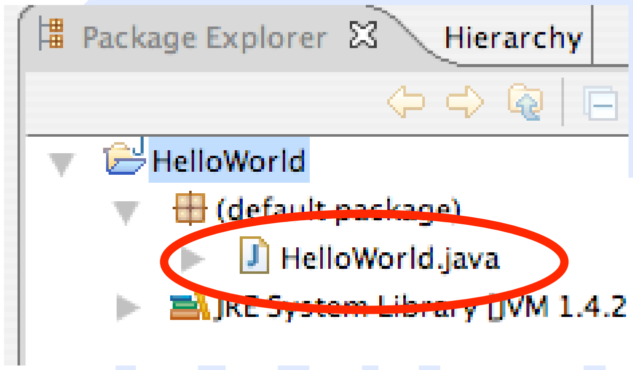
☐ Generate comments

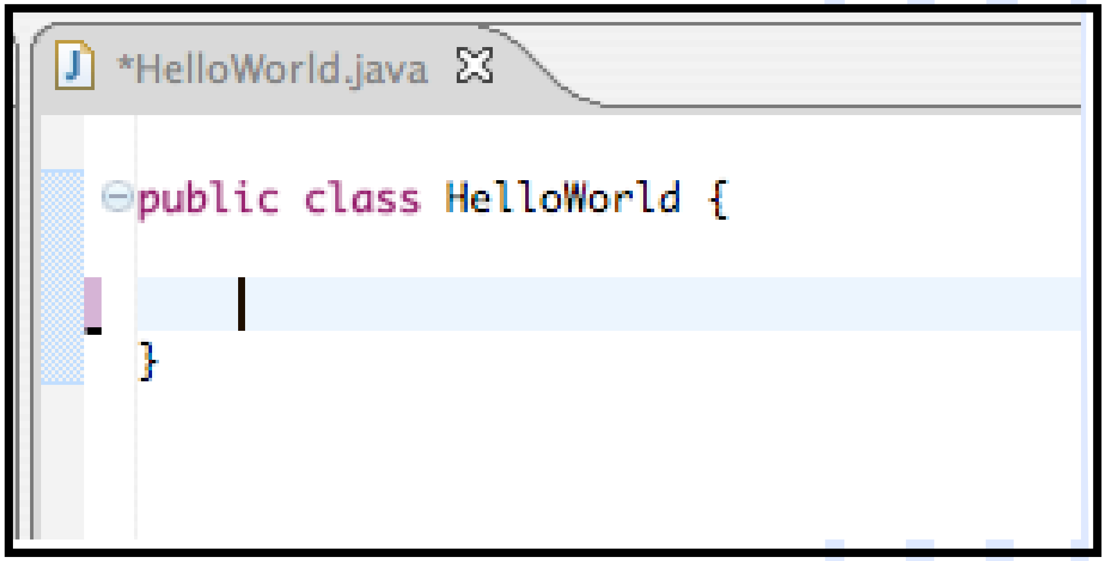
That was a bit of a pain…

- Eclipse is an industry tool
  - Comes with all the options
  - Sometimes gets to be a bit overwhelming
- We'll be importing projects in the future
  - I'll give you a project and tell you what part to work on
  - This is just to tell you how to write a program from scratch
  - Ok, so let's write something!

Classes

- Open your class
  - HelloWorld.java
- Should look like this:



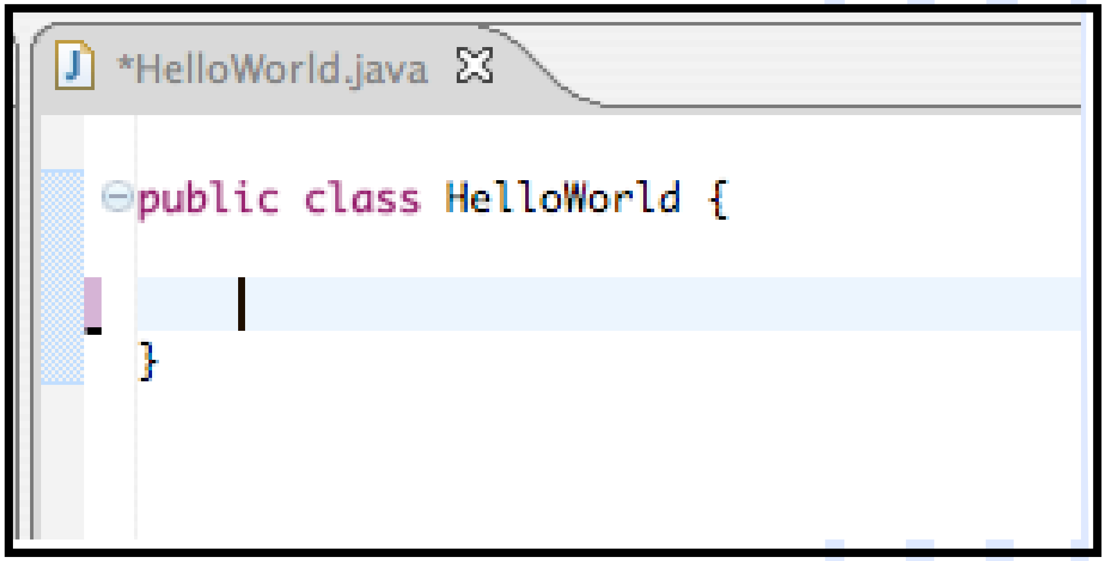


```
public class HelloWorld {

}
```

Classes

- A **class** defines an object
  - Don’t worry about this yet
  - just know that all Java programs are made up of classes
- Your HelloWorld class is about as simple as it gets:

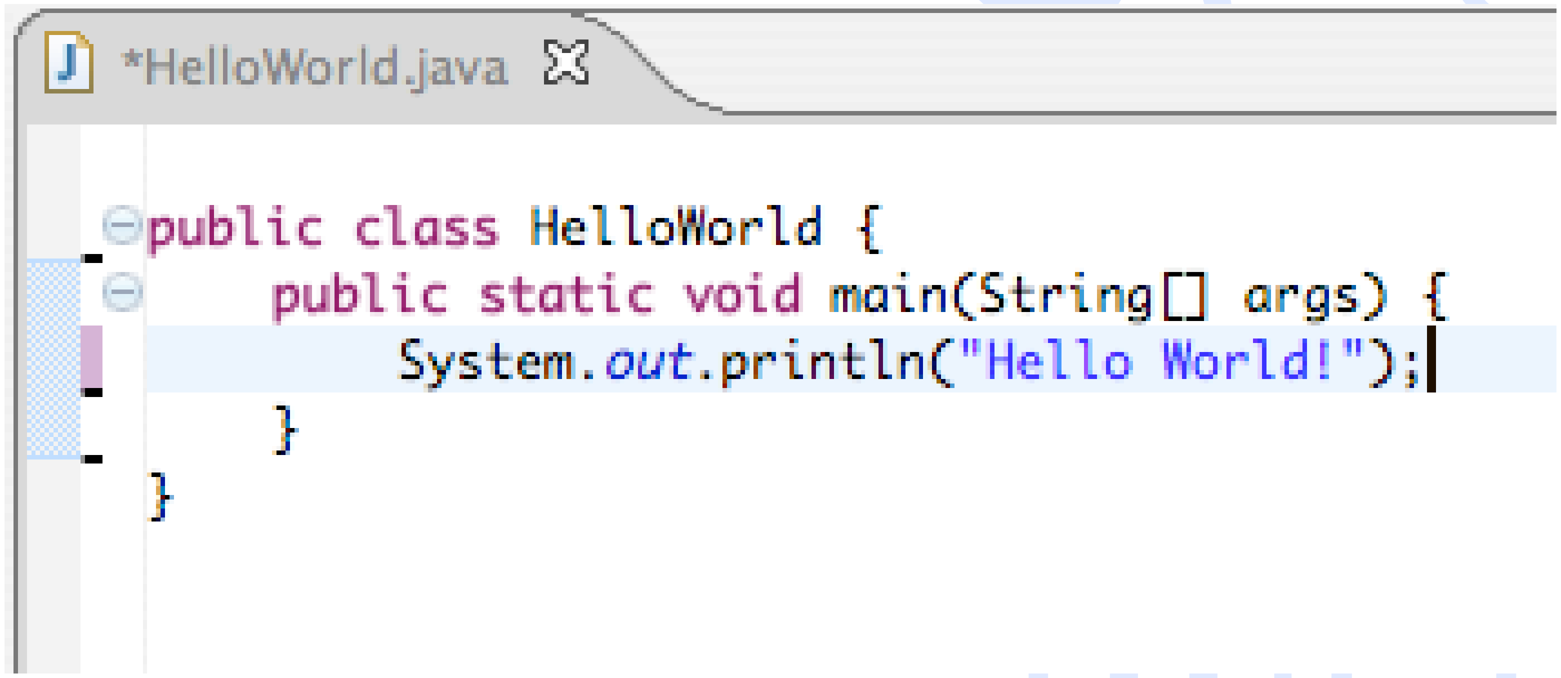


```
public class HelloWorld {

}
```

Main method

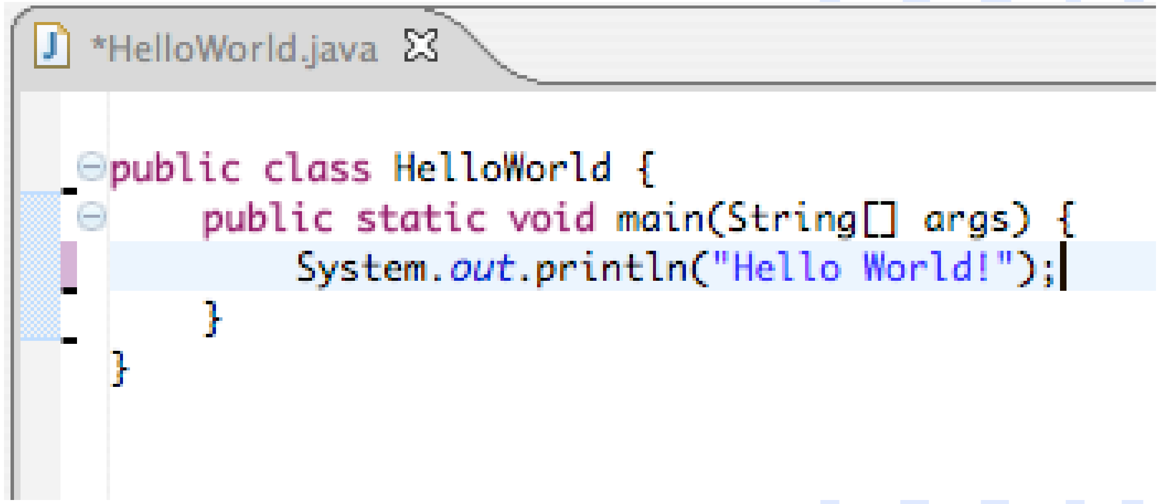
- Modify your class so it looks like this:



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

Simplest Java Program

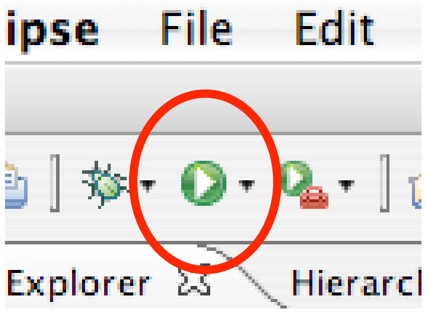
- This program just prints out “Hello World”
  - Well that was a lot of work just for that!
  - “public static void main(String[] args)” is a mouthful
    - Don’t worry about what it means today
  - Just know that all programs start with **main()**
  - And know that System.out.println() outputs a line of text to the screen



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

Let’s run it

- Once you’ve written your program, click the Run icon:



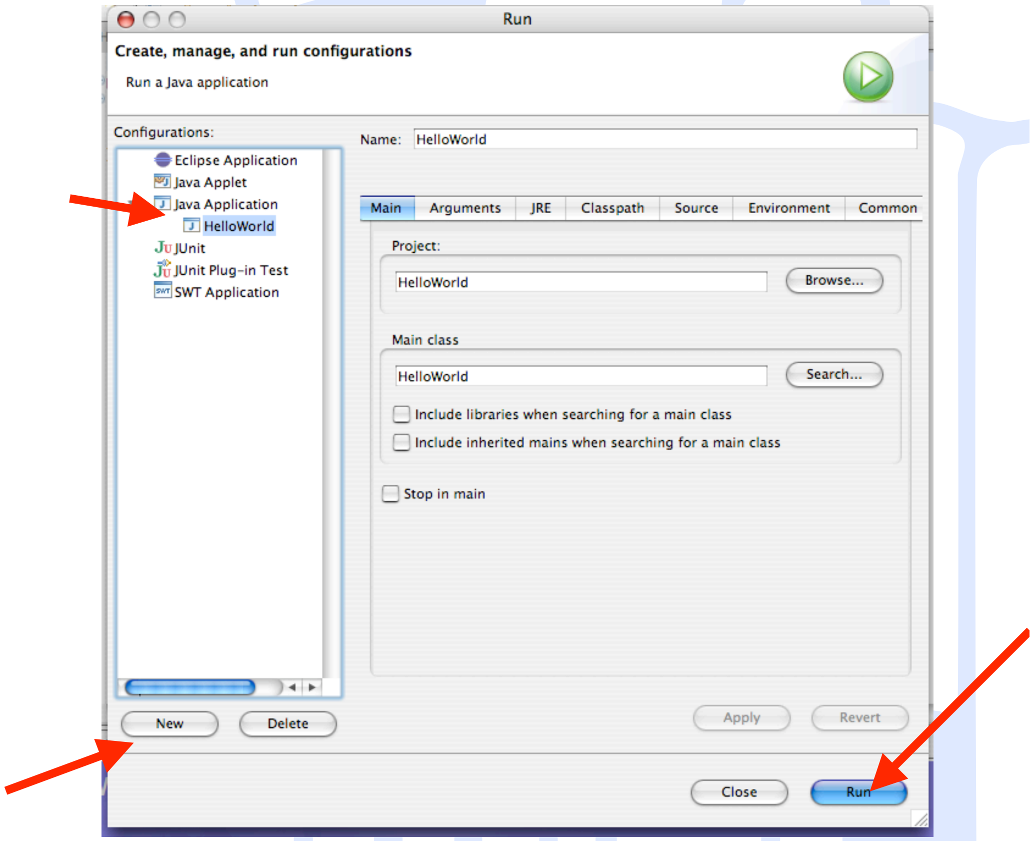
- When you click “Run”, Eclipse will compile **and** run!
  - Isn’t that nice?

Let's run it

- The first time you run something, you have to create a run configuration

1. Choose "Java Application"
2. Click "New"
3. Just click "Run"

- If you click the Run button again, you won't need to do all this.


Run

Create, manage, and run configurations

Run a Java application

Configurations:
- Eclipse Application
- Java Applet
- Java Application
  - HelloWorld
- JUnit
- JUnit Plug-in Test
- SWT Application

Name: HelloWorld

Main | Arguments | JRE | Classpath | Source | Environment | Common

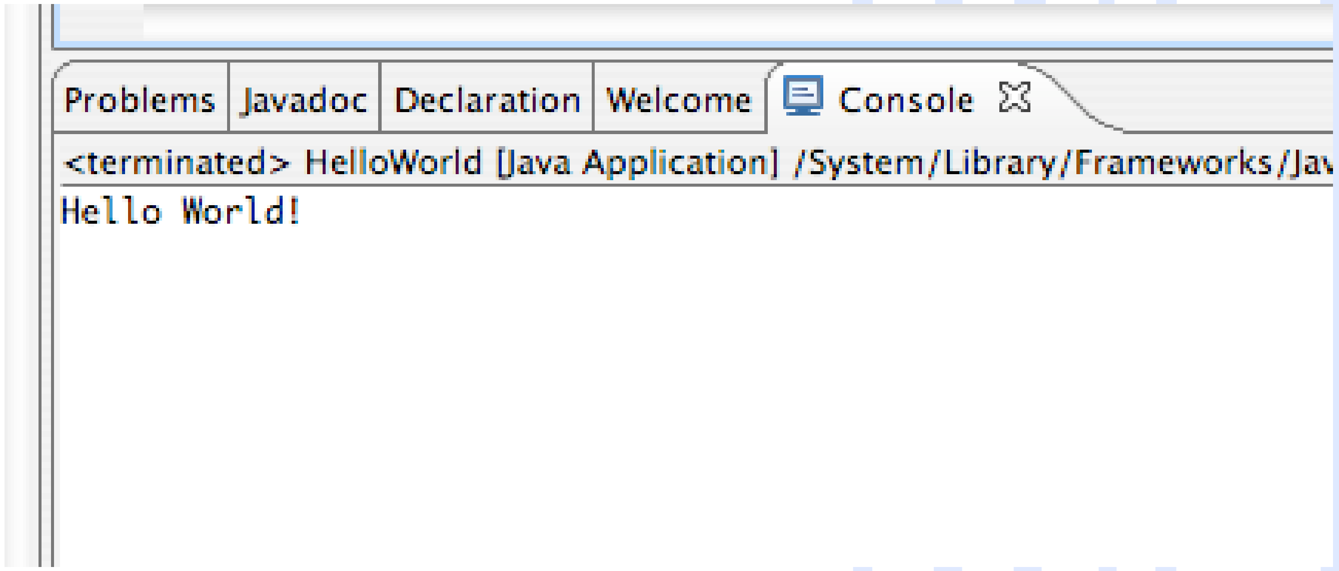
Project: HelloWorld

Main class: HelloWorld

- [ ] Include libraries when searching for a main class
- [ ] Include inherited mains when searching for a main class
- [ ] Stop in main

We did it!

- Your program should output “Hello World”
  - If it doesn’t, then talk to me



```
<terminated> HelloWorld [Java Application] /System/Library/Frameworks/Jav
Hello World!
```

A few words on Java

- Writing simple programs in Java isn't as easy as it could be
  - Things like "public static void main…" are awkward
  - To really understand what all those words mean, you need to know more Java
- For now, just accept that:
  - programs need to be in classes (public class…)
  - when you run a Java program, it looks for "main()" and starts running whatever is there

Hello World in other languages

Perl

```
print “Hello World!\n”;
```

Python

```
print “Hello World!”
```

C++

```
#include <iostream>
using namespace std;

int main(int argc, char **argv) {
    cout << "Hello World!" << endl;
}
```

C

```
#include <stdio.h>;


int main(char **argv) {
    printf(“Hello World!”);
}
```

Java

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

Pascal

```
program HelloWorld
begin
    writeln('Hello, World!');
end.
```

Hello World in other languages

PowerPC Assembler

```
 .section __TEXT,__text,regular,pure_instructions
    .section __TEXT,__picsymbolstub1,
             symbol_stubs,
             pure_instructions,32
.data
.cstring
    .align 2
LC0:
    .ascii "Hello World!\0"
.section __TEXT,__text,regular,pure_instructions
    .align 2
    .align 2
    .globl _main
.section __TEXT,__text,regular,pure_instructions
    .align 2
_main:
    mflr r0
    stmw r30,-8(r1)
    stw r0,8(r1)
    stwu r1,-80(r1)
    mr r30,r1
    bcl 20,31,"L00000000001$pb"
"L00000000001$pb":
    mflr r31
    stw r3,104(r30)
    stw r4,108(r30)
    addis r3,r31,ha16(LC0-"L00000000001$pb")
    la r3,lo16(LC0-"L00000000001$pb")(r3)
```

```
 bl L_printf$stub
    mr r3,r0
    lwz r1,0(r1)
    lwz r0,8(r1)
    mtlr r0
    lmw r30,-8(r1)
    blr
.data
.section __TEXT,__picsymbolstub1,
     symbol_stubs,pure_instructions,32
    .align 2
L_printf$stub:
    .indirect_symbol _printf
    mflr r0
    bcl 20,31,L0$_printf
L0$_printf:
    mflr r11
    addis r11,r11,ha16(L_printf$lazy_ptr-L0$_printf)
    mtlr r0
    lwzu r12,lo16(L_printf$lazy_ptr-L0$_printf)(r11)
    mtctr r12
    bctr
.data
.lazy_symbol_pointer
L_printf$lazy_ptr:
    .indirect_symbol _printf
    .long dyld_stub_binding_helper
    .subsections_via_symbols
```

Hello World in other languages

Intel Assembler

```
 .file   "test.c"
     .section    .rodata
.LC0:
     .string "Hello World!"
     .text
.globl main
     .type   main,@function
main:
     pushl   %ebp
     movl    %esp, %ebp
     subl    $8, %esp
     andl    $-16, %esp
     movl    $0, %eax
     subl    %eax, %esp
     subl    $12, %esp
     pushl   $.LC0
     call    printf
     addl    $16, %esp
     leave
     ret
.Lfe1:
     .size   main,.Lfe1-main
     .section    .note.GNU-
stack,"",@progbits
     .ident  "GCC: (GNU) 3.2.3
20030502 (Red Hat Linux 3.2.3-42)"
```

Comments

- Sometimes you might want to annotate your code
  - As you may have noticed, code isn't so easy to understand sometimes
- In Java, you can add comments 2 ways:

```
// this is a line comment
// line comments start with slashes and go to end of line

/* This is a longer comment */
/* Longer comments
   can span multiple lines
*/
```

Comments

- Say we wanted to annotate Hello World
  - Can add comments right in the code
  - Might help someone who read it understand.

```
public class HelloWorld {
    //This is the main method
    public void main(String[] args) {
        //This prints out some text
        System.out.println(“Hello World!”);
    }
}
```

So what was the point?

- This exercise was just to familiarize you with Eclipse and writing/running programs

- You now know how to:
  - Start a Java project
  - Write some very simple Java code
  - Compile and run your code
  - And, hopefully Eclipse is not quite as scary

- Things will be a little more gentle from now on
  - But you've had a taste of what's to come
- No grade on this, but it should help you with your first real assignment next Friday!