Advanced Java

LESSON 1: WELCOME, INTELLIJ, GITHUB, JAVADOC AND UNIT TEST / JUNIT

Creative Commons

Attribution 4.0 International (CC BY 4.0)

This workforce product was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The product was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The U.S. Department of Labor makes no guarantees, warranties, or assurances of any kind, express or implied, with respect to such information, including any information on linked sites and including, but not limited to, accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability, or ownership. This is an equal opportunity program. Assistive technologies are available upon request and include Voice/TTY (771 or 800-947-6644).

Agenda

- Welcome
- IntelliJ IDEA
- JavaDoc
- GitHub
- Unit Test / JUnit

Welcome

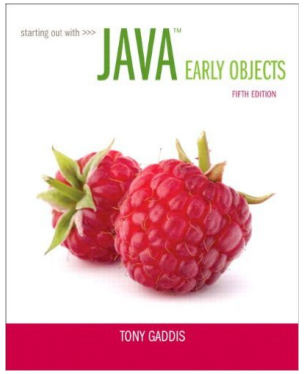
Introductions

- Name
- Development / Coding outside of class
- Hobby

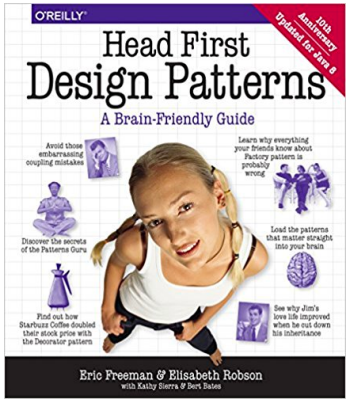
Recourses

Textbook:

Starting Out with Java: Early Objects (5th Edition), Tony Gaddis, Addison Wesley, ISBN-10: 0133776743



Head First Design Patterns, Elisabeth Freeman and Kathy Sierra, O'Reilly, ISBN-10: 0596007124



Internet Access to:

Blackboard http://wctc.blackboard.edu

Github http://github.com

Lucid Chart https://www.lucidchart.com

IntelliJ IDEA https://www.jetbrains.com/idea/download

Other Java Resources as prescribed by the instructor

Syllabus

- Attendance Policy
- Academic Integrity
- Special Needs
- Cell phones
- Lab Rules
- Assignments, Tests and Labs
- Tentative Course Schedule
- Bad Weather / School Closing

Syllabus

Hybrid Course – What does that mean?

- We meet once a week for 4 hours verses twice a week for 5 hours. (2.5 each meeting)
- The work during the week is not just homework – it is part of the lesson. (Hybrid courses require more time outside of the class)
- For each topic (week), the skills will be introduced through: a lecture / demo, in-class activities, Weekly Challenges and Unit Challenges
  - Weekly Challenges will consist of a reading (mostly textbook), some videos, coding challenges and questions
- Maybe dropped if you miss 2 consecutive lessons or 3 lesson.

Course Structure

Course Components
Units The course is divided into five Units: ▪ Exceptions and File I/O ▪ Collections ▪ Encapsulation ▪ Design ▪ Algorithms
Unit Challenges There is a project for each of the first four Units
Weekly Lessons Each week's lecture will include several short activities which are awarded points for participating
Weekly Challenges There will be a folder for each week that includes multiple graded activities.

Course Grading Policies

Course Points
Unit Challenges Four Challenges that are 100 points each for 400 points
Weekly Lessons Fifteen Lessons that are 10 points each for 150 points
Weekly Challenges Fifteen Challenges that are 30 points each for 450 points
Critical Life Skills Activity One Critical Life Skill for 50 points

Course Grading Policies

Course Components
Unit Challenges All Weekly Challenges and the Unit Challenges need to be submitted midnight the night before the lesson introducing the next topic. The Unit Challenge will not be considered submitted until all the Weekly Challenges for the Unit have been submitted. If the Unit Challenge is submitted a late, there will be a 20 point deduction. If the Unit Challenge is not submitted within two weeks, no other course work may be submitted until the student meets with the teacher and arranges an improvement plan. A Unit Challenge that is submitted on time but does not score well on the Rubrics may be offered an additional week to re-write the program based upon the feedback.
Weekly Lessons Points awarded for participation in weekly activities may not be made-up.
Weekly Challenges Questions may be submitted multiple times but the Coding Exercises may only be submitted once. Weekly Challenges need to be submitted midnight the night before the next lesson. Late Challenges are deducted by 10 points.
Critical Life Skills Activity Same rules as Unit Challenges.

Syllabus

Grades	Percent Value	Point Value
A	95-100	4.00
A-	93-94	3.67
B+	91-92	3.33
B	87-90	3.00
B-	85-86	2.67
C+	83-84	2.33
C	79-82	2.00
C-	77-78	1.67
D+	75-76	1.33
D	72-74	1.00
D-	70-71	0.67
F	69 or below	0.00
W	Withdrawal	0.00
Au	Audit	0.00

Course Expectations

Success in this Class

Time Expectations

- *The traditional out-of-class work-load for a college course is two to three hours out-of-class for every hour in class or credit. While many courses do not require this load of work, this one will. There will be weeks that take less time and some that require more but the average should be about eight hours per week.*

What to DO Outside of Class

- *Read the Text Book (Take Notes, try the sample code, etc…)*
- *Complete Weekly Challenges on time*
- *Complete the Unit Challenges on time*

Course Expectations

Success in this Class

Do Not Miss Class

- *All class periods have partition points that can not be made-up*
- *The rules have been tightened from previous semesters because very few students who missed class succeeded*

Turn Everything on time (#1 Reason for past failures)

- *WCTC has a very tight grading scale (A 95-100, C 79-82) there is no room for missed work*
- *Take advantage of extra-credit opportunities*

Course Expectations

Success in this Class

What to DO in Class

- *Watch Lectures (We will be using a broadcasting tool to see the lectures on your own screen – use it)*
- *Listen to Lectures (#2 Reason for past failures)*
  - *Do not try to follow along during the lecture*
  - *Opportunities to try demonstrated techniques are planned at a regular frequency*
  - *The Class Pattern is: Instructor Describes, Instructor Demonstrates, Student Tries, Student is Assessed*
    - *Interrupting the Instructor during Demonstrations so that a single student may follow along negatively impacts the entire class*
    - *It is entirely appropriate to ask questions directly related to the Demonstration during the Demonstration*

Course Expectations

Asking Questions

- *During Lectures and Demonstrations*
  - *Interaction is encouraged, please ask questions that clarify the concepts being introduced*
  - *Keep all questions related to the material being presented*
    - *Unless you cannot see the presentation or something else that is [pertinent to the beginning of the class*
    - *Wait until and Activity or Lab to ask about anything on your computer except your ability to see the presentation*
- *During Activities and Labs*
  - *Feel free to ask for clarification on what you are being asked to do*
  - *If you need help with tasks, the instructor will ask you*
    - *What does the PPT say?*
    - *What does the Book say?*
    - *What have you tried?*

IntelliJ IDEA

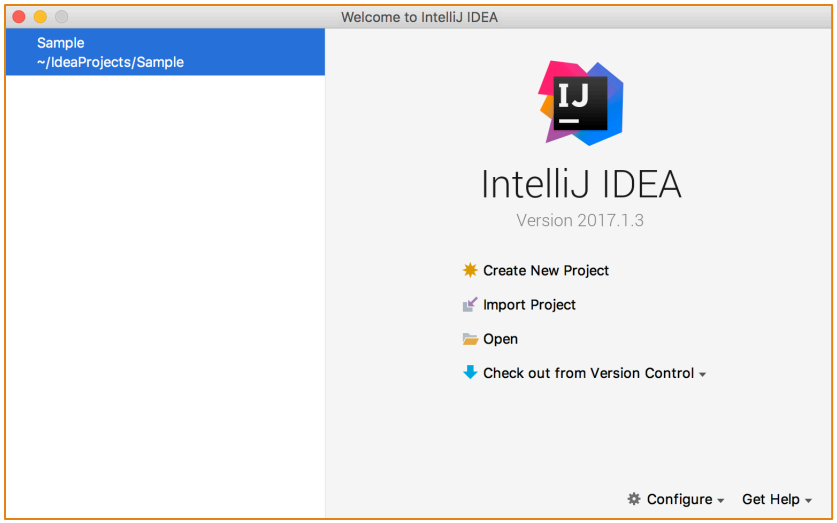
What is IntelliJ IDEA?

Java integrated development environment (IDE) developed by JetBrains (Software development company from Prague, Czech Republic).

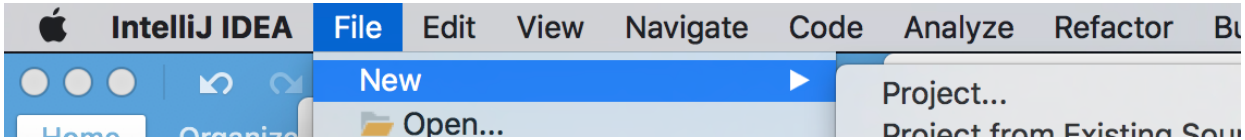
- Written in Java
- Base for Google's Android Studio
- Professional IDE with a community version

What is IntelliJ IDEA?

1. Select “Create New Project”

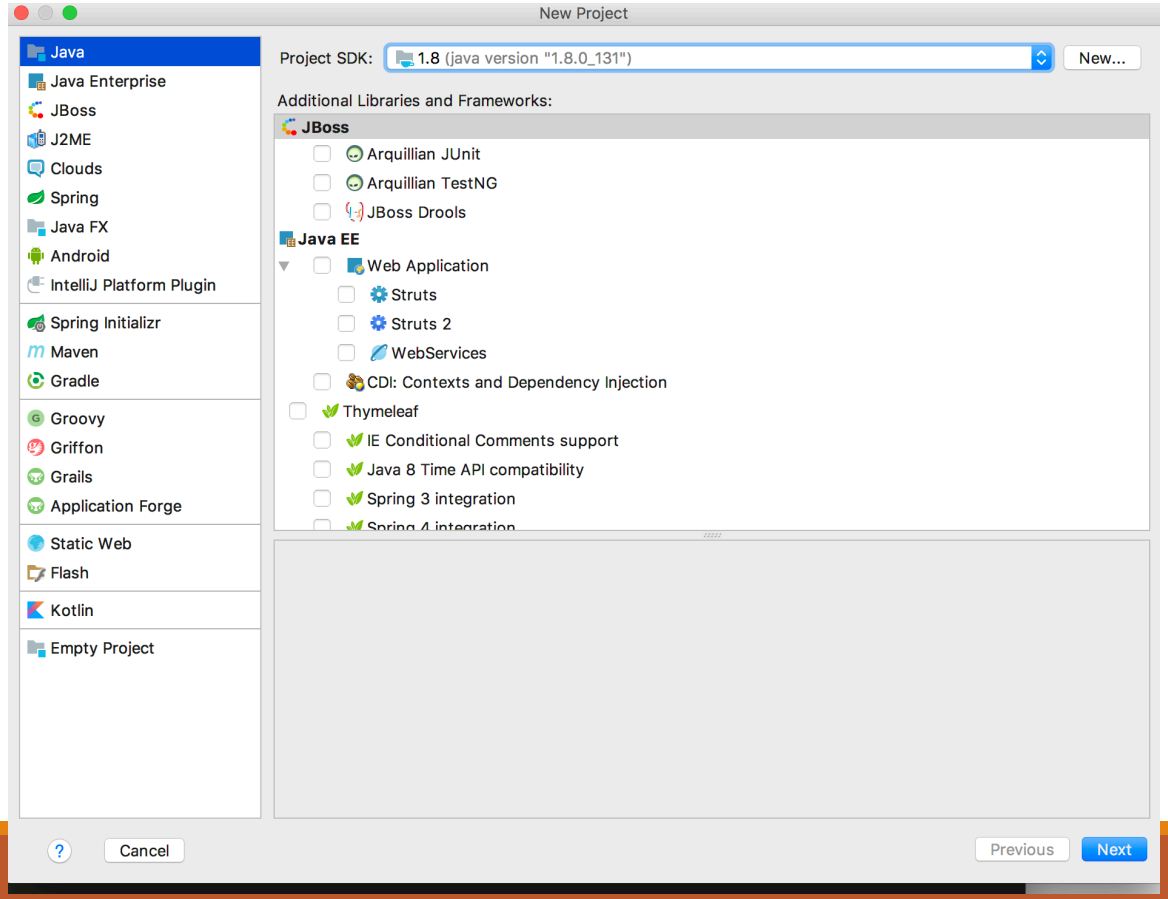


or



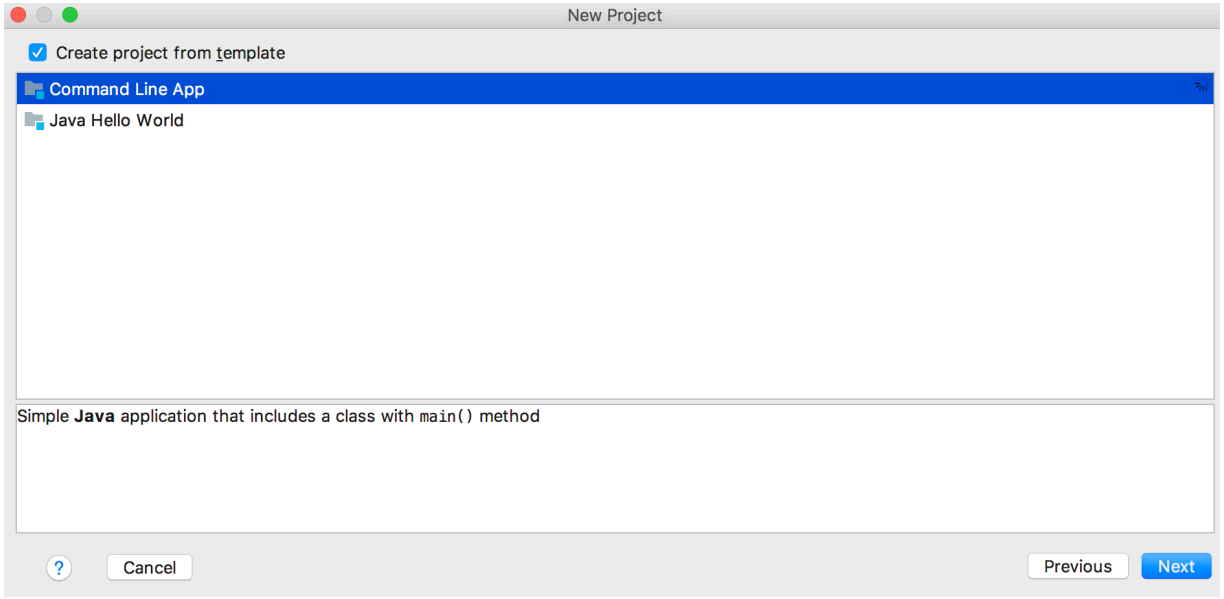
What is IntelliJ IDEA?

2. Select Java
3. Select “Next”

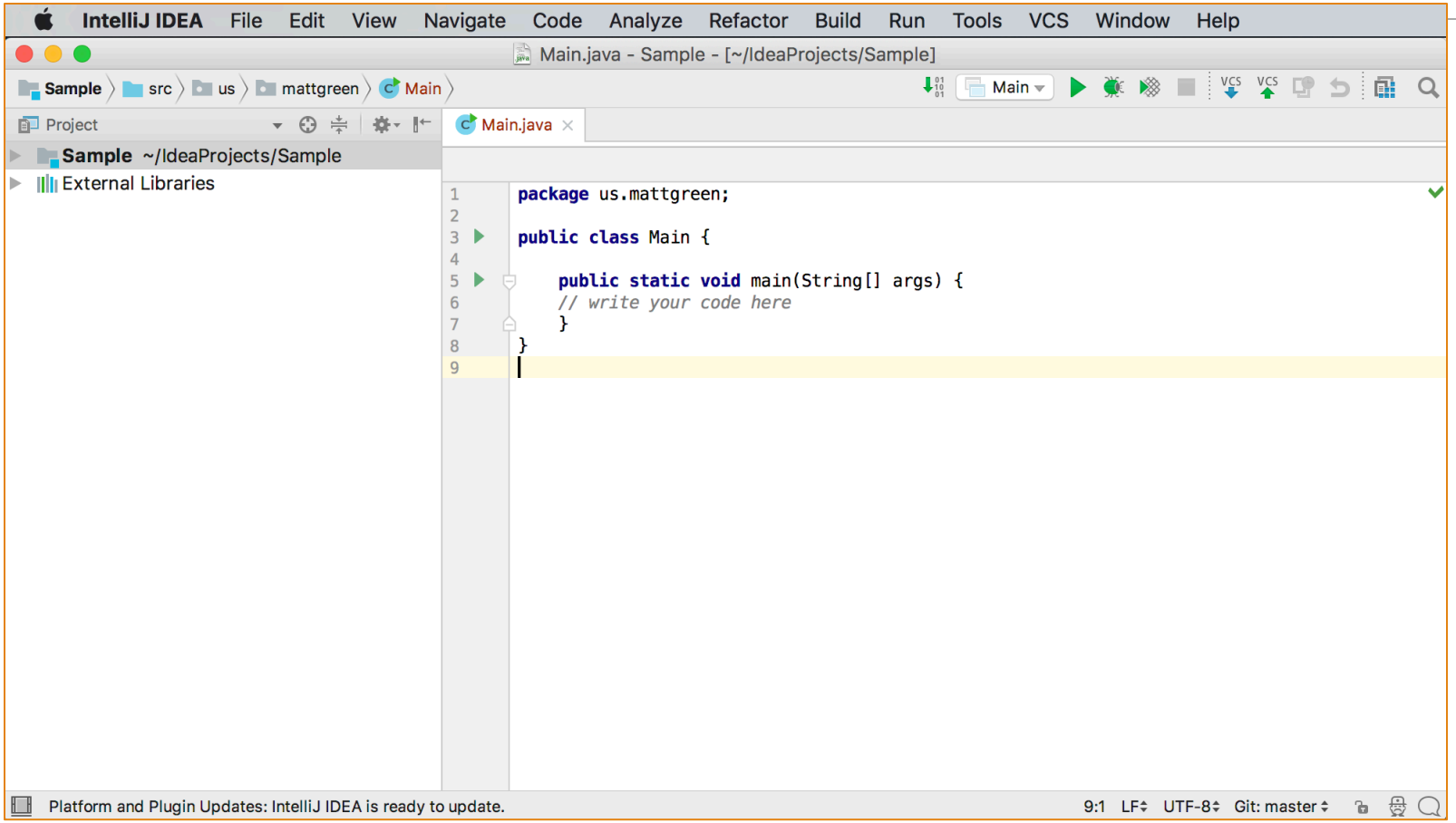


What is IntelliJ IDEA?

4. Check “Create project from template”
5. Select “Command Line App”
6. Select “Next”



What is IntelliJ IDEA?



In-Class Activity

1. Download Netbeans Project, Animals, from Blackboard.
2. Create a new Project, Animals, in IntelliJ
3. Create all of the Java Files from the old project into the new project (Several ways but brute force is the quickest)
4. Run your program and try debugging it

JavaDoc

Comments

- Comments are:
  - notes of explanation that document lines or sections of a program.
  - part of the program, but the compiler ignores them.
  - intended for people who may be reading the source code.
- In Java, there are three types of comments:
  - Single-line comments
  - Multiline comments
  - Documentation comments

Single-Line Comments

Code Listing 2-24 (Comment1.java)

```
1  // PROGRAM: Comment1.java
2  // Written by Herbert Dorfmann
3  // This program calculates company payroll
4
5  public class Comment1
6  {
7     public static void main(String[] args)
8     {
9        double payRate;       // Holds the hourly pay rate
10       double hours;         // Holds the hours worked
11       int employeeNumber;   // Holds the employee number
12
13       // The remainder of this program is omitted.
14    }
15 }
```

- Place two forward slashes (//) where you want the comment to begin.
  - The compiler ignores everything from that point to the end of the line.

Multiline Comments

Code Listing 2-25 (Comment2.java)

```
1  /*
2     PROGRAM: Comment2.java
3     Written by Herbert Dorfmann
4     This program calculates company payroll
5  */
6
7  public class Comment2
8  {
9     public static void main(String[] args)
10    {
11       double payRate;      // Holds the hourly pay rate
12       double hours;        // Holds the hours worked
13       int employeeNumber;  // Holds the employee number
14
15       // The remainder of this program is omitted.
16    }
17 }
```

- Start with /* (a forward slash followed by an asterisk) and end with */ (an asterisk followed by a forward slash).
  - Everything between these markers is ignored.
  - Can span multiple lines

Block Comments

Table 2-16 Block comments

```
/**                                    //**********************************
 *   This program demonstrates the      //    This program demonstrates the *
 *   way to write comments.             //    way to write comments.        *
 */                                     //**********************************

////////////////////////////////////   //--------------------------------
//    This program demonstrates the    //    This program demonstrates the
//    way to write comments.           //    way to write comments.
////////////////////////////////////   //--------------------------------
```

- Many programmers use asterisks or other characters to draw borders or boxes around their comments.
- This helps to visually separate the comments from surrounding code.

Documentation

- Any comment that starts with `/**` and ends with `*/` is considered a documentation comment.

- You write a documentation comment just before:
  - a class header, giving a brief description of the class.
  - each method header, giving a brief description of the method.

- *Documentation comments* can be read and processed by a program named javadoc, which comes with the Sun JDK.

JavaDoc

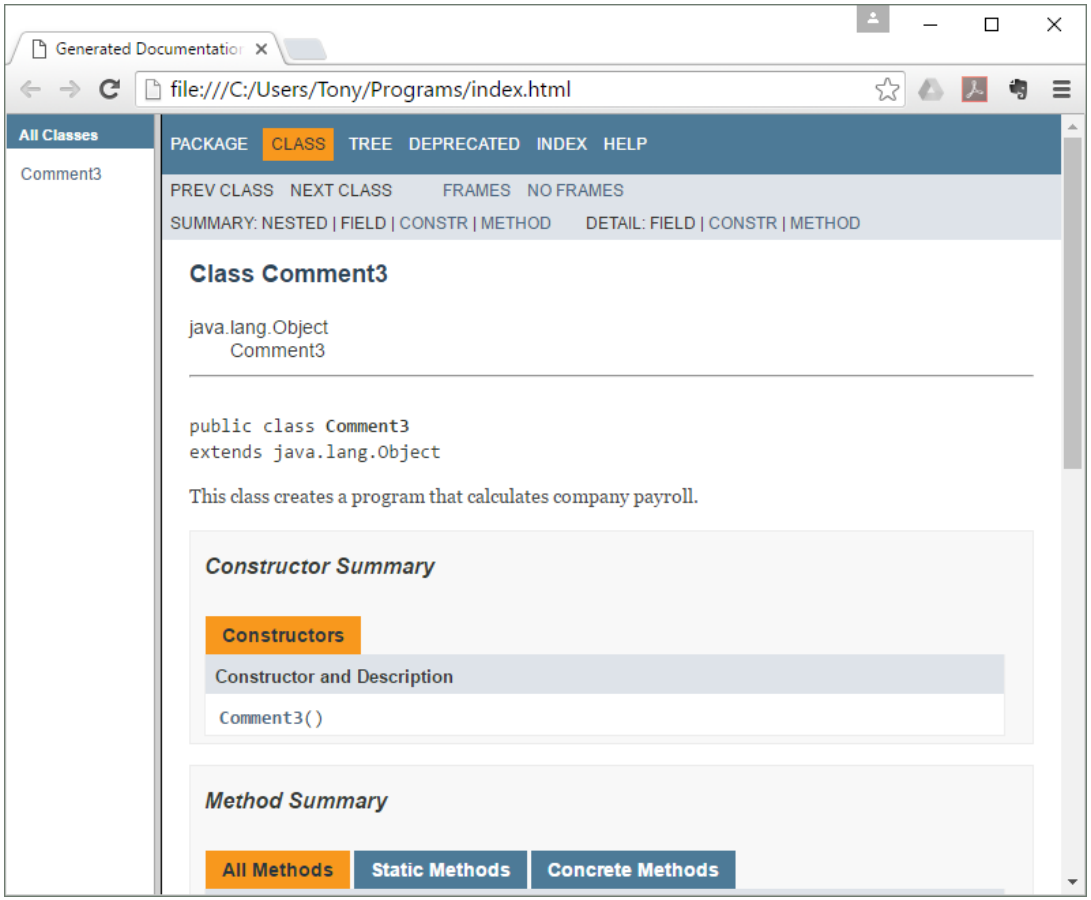
- **Javadoc:** The program to generate java code documentation.
- **Input:** Source files (.java)
- **Output:** Documentation web site comporised of HTML documents

JavaDoc - Input

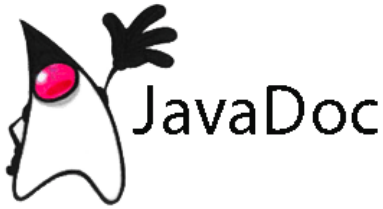
Code Listing 2-26 (Comment3.java)

```
1  /**
2     This class creates a program that calculates company payroll.
3  */
4
5  public class Comment3
6  {
7     /**
8        The main method is the program's starting point.
9     */
10
11    public static void main(String[] args)
12    {
13       double payRate;        // Holds the hourly pay rate
14       double hours;          // Holds the hours worked
15       int employeeNumber;    // Holds the employee number
16
17       // The Remainder of This Program is Omitted.
18    }
19 }
```

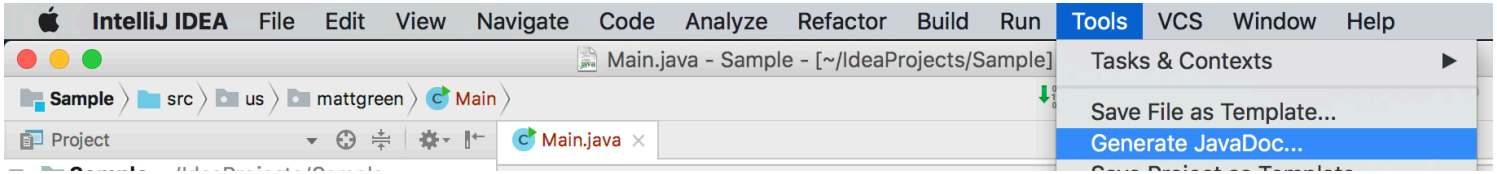
JavaDoc - Output



JavaDoc - IntelliJ IDEA

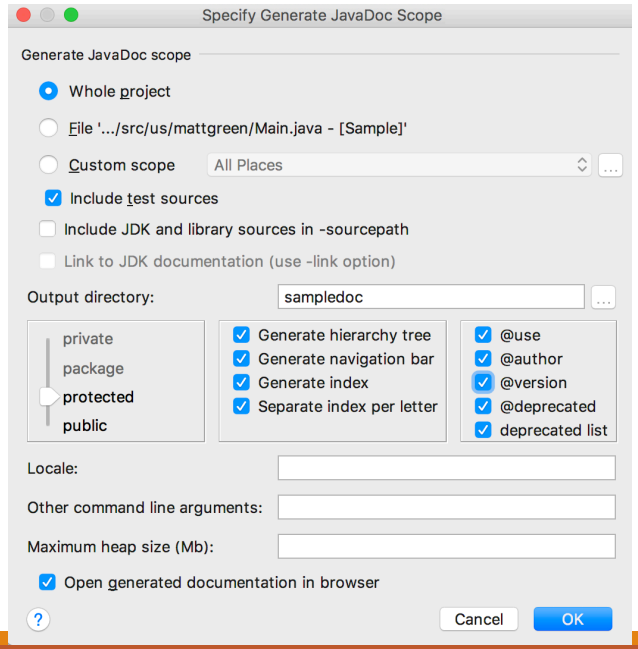


1. Select Tools > Generate JavaDoc from the menu



2. Identify the folder for the documentation

3. Press “Ok”



Structure of Comments

Main Description

Tag Section

```
/**
 * Returns the character at the specified index. An index
 * ranges from <code>0</code> to <code>length() - 1</code>.
 *
 * @param     index  the index of the desired character.
 * @return    the desired character.
 * @exception StringIndexOutOfRangeException
 *              if the index is not in the range <code>0</code>
 *              to <code>length()-1</code>.
 * @see       java.lang.Character#charValue()
 */
public char charAt(int index) {
    ...
}
```

Comments are HTML


```
/**
 * This is a <b>doc</b> comment.
 * @see java.lang.Object
 */
```

Note that tag names are case-sensitive. @See is a mistaken usage - @see is correct.

@author

Adds an "Author" entry with the specified *author's-name* to the generated docs when the -author option is used.

A doc comment may contain multiple @author tags.

You can specify one name per @author tag or multiple names per tag.

- @author Matt Green
  @author Jeff Grissom

- @author Matt Green, Jeff Grissom

Main Description

The section that introduces the class, method of field. It comes first in the specification before the tag section.

The first sentence of each doc comment should be a summary sentence, containing a concise but complete description of the declared entity. It will be placed in package overview and class overview.

@param

@param ***parameter-name description***

Adds a parameter to the "Parameters" section. The description may be continued on the next line. This tag is valid only in a doc comment for a method or constructor.

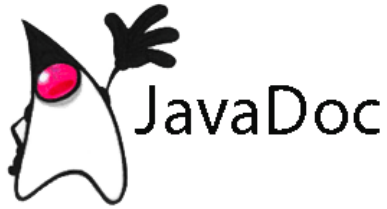
```
/**
 * Tests a character and notifies an observer
 * according to the value of the character
 * @param ch the character to be tested
 * @param obs the observer to be notified
 */
public void testCharacter( char ch , Observer obs )
```

@return

Adds a "Returns" section with the *description* text. This text should describe the return type and permissible range of values. This tag is valid only in a doc comment for a method.

```
/**
 * Tests a character and notifies an observer
 * according to the value of the character
 * @param ch the character to be tested
 * @param obs the observer to be notified
 * @return result of the action performed by the
 * observer on the character
 */
public int testCharacter(char ch, Observer obs)
```

@throws/@exception



The @throws and @exception tags are synonyms.

Adds a "Throws" subheading to the generated documentation, with the *class-name* and *description* text.

The *class-name* is the name of the exception that may be thrown by the method.

Multiple @throws tags can be used in a given doc comment for the same or different exceptions.

To ensure that all checked exceptions are documented, if a @throws tag does not exist for an exception in the throws clause, the Javadoc tool automatically adds that exception to the HTML output (with no description) as if it were documented with @throws tag.

@throws/@exception


```
/**
 * Draws the shape of the object on the screen
 * instance given in the parameter.
 * @throws NullPointerException If the screen
 * object is null.
 * @throws ScreenSizeException If the object does
 * not fit on the screen.
 */
public boolean drawShape(Screen screen)
 throws NullPointerException, ScreenSizeException{
...
}
```

@version

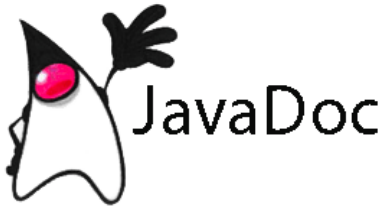
Adds a "Version" subheading with the specified *version-text* to the generated docs when the -version option is used. This tag is intended to hold the current version number of the software that this code is part of (as opposed to @since, which holds the version number where this code was introduced). The *version-text* has no special internal structure.

A doc comment may contain multiple @version tags.

Which Tags go where?

Overview / Package	Class/Interface	Field / Property	Method/Constructor
@since @author @version	@since @deprecated @author @version	@since @deprecated	@since @deprecated @param @return @throws / @exception

Tags Required in this Course



Class/ Interface	Method/ Constructor
@author @version	@param @return @throws / @exception

In-Class Activity

1. Add all required JavaDoc comments
2. Create Documentation site
3. Show site to the instructor

GitHub

Git & GitHub

- Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people.
- Git was created by Linus Torvalds in 2005 for development of the Linux kernel.

GitHub

- A Web-based Git version control repository hosting service.

Git & GitHub

IntelliJ - GitHub

Official Documentation

https://www.jetbrains.com/help/idea/manage-projects-hosted-on-github.html

In-Class Activity

1. Create or open your GitHub account
2. Connect your IntelliJ project to a GitHub repository
3. Share with your instructor
4. Show your repository to your instructor

Unit Test / JUnit

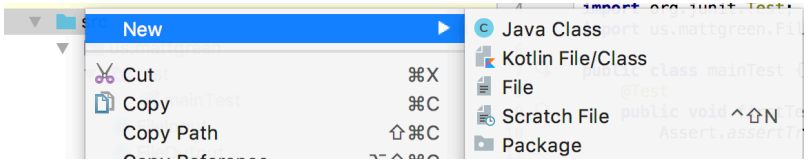
Unit Testing History

- Kent Beck developed the first xUnit automated test tool for Smalltalk in mid-90's
- Beck and Gamma (of design patterns Gang of Four) developed JUnit on a flight from Zurich to Washington, D.C.
- Martin Fowler: "Never in the field of software development was so much owed by so many to so few lines of code."
- Junit has become the standard tool for Test-Driven Development in Java (see Junit.org)
- Junit test generators now part of many Java IDEs (Eclipse, BlueJ, Jbuilder, DrJava)
- Xunit tools have since been developed for many other languages (Perl, C++, Python, Visual Basic, C#, …)

JUnit

To set-up Junit:

1. create a package under source, src. Right-click ”Src”, select “New”, then “Package”.
2. Name the package the same as the package your code is in followed by “.test”. (ex: us.mattgreen.test)
3. Create a class with a similar name to the class you will be testing. Name of class is important – should be of the form ***Test***MyClass or MyClass***Test.***




```
import org.junit.*;
public class MyClassTest {
    @Test
    public void testSomething() {
    }
}
```

4. Fix errors by clicking “@Test” and then fix errors by clicking F2.

JUnit - Writing

Pattern follows ***programming by contract*** paradigm:

- Set up **preconditions**
- Exercise functionality being tested
- Check **postconditions**

```
public void testEmptyList() {
   Bowl emptyBowl = new Bowl();
   assertEquals("Size of an empty list should be zero. ",
         0, emptyList.size());
   assertTrue("An empty bowl should report empty. ",
         emptyBowl.isEmpty());
}
```

Things to notice:

- Specific method signature – public void ***test***Whatever()
- Allows them to be found and collected automatically by JUnit
- Coding follows pattern
- Notice the assert-type calls…

JUnit - Writing

Each assert method has parameters like these:
message, expected-value, actual-value

Assert methods dealing with floating point numbers get an additional argument, a tolerance

Each assert method has an equivalent version that does not take a message – however, this use is not recommended because:

- messages helps documents the tests
- messages provide additional information when reading failure logs

Assert methods

- assertTrue(String *message*, Boolean *test*)
- assertFalse(String *message*, Boolean *test*)
- assertNull(String *message*, Object *object*)
- assertNotNull(String *message*, Object *object*)
- assertEquals(String *message*, Object *expected*, Object *actual*) (uses equals method)
- assertSame(String *message*, Object *expected*, Object *actual*) (uses == operator)
- assertNotSame(String *message*, Object *expected*, Object *actual*)

Assert methods

Suppose you want to test a class Counter

```
public class CounterTest
                extends junit.framework.TestCase {
```

- This is the unit test for the Counter class

```
public CounterTest() { } //Default constructor
```

```
protected void setUp()
```

- Test *fixture* creates and initializes instance variables, etc.

```
protected void tearDown()
```

- Releases any system resources used by the test fixture

```
public void testIncrement(), public void testDecrement()
```

- These methods contain tests for the Counter methods increment(), decrement(), etc.
- Note capitalization convention

JUnit tests for Counter


```
public class CounterTest extends
junit.framework.TestCase {
   Counter counter1;
   public CounterTest() { }

   protected void setUp() {
        counter1 = new Counter();
   }

   public void testIncrement() {
       assertTrue(counter1.increment() == 1);
       assertTrue(counter1.increment() == 2);
    }

   public void testDecrement() {
       assertTrue(counter1.decrement() == -1);
   }
}
```

Note that each test begins with a *brand new* counter

This means you don't have to worry about the order in which the tests are run

TestSuites

TestSuites collect a selection of tests to run them as a unit

Collections automatically use TestSuites, however to specify the order in which tests are run, write your own:

```
public static Test suite() {
    suite.addTest(new TestBowl("testBowl"));
    suite.addTest(new TestBowl("testAdding"));
    return suite;
}
```

Should seldom have to write your own TestSuites as each method in your TestCase should be independent of all others.

JUnit

IntelliJ

Official Documentation

https://www.jetbrains.com/help/idea/configuring-testing-libraries.html

https://www.jetbrains.com/help/idea/tutorial-test-driven-development.html#d759853e7

In-Class Activity

1. Create Unit Tests for the project
2. Run Junit tests