

COMP 249: Object Oriented Programming II

Tutorial 1:

Review of COMP248 and Javadoc

Question 1

Assume a rectangle class with two attributes, a and b representing the size of the rectangle. What is the output of this code,

```
public void myMethod(Rectangle rect) {  
    rect.a = 15;  
    rect.b = 15;  
}  
  
public static void main(String[] args) {  
    Rectangle r = new Rectangle(10, 10);  
    MyClass c = new MyClass();  
  
    c.myMethod(r);  
    System.out.println(r.toString()); // Rectangle size  
}
```

Question 2

What is the output of this code, assuming previous rectangle class:

```
public void myMethod(Rectangle rect1, Rectangle rect2) {  
    rect1 = rect2;  
}  
  
public static void main(String[] args) {  
    Rectangle r1 = new Rectangle(10, 10);  
    Rectangle r2 = new Rectangle(15, 15);  
    MyClass c = new MyClass();  
  
    c.myMethod(r1, r2);  
    System.out.println(r1.toString()); // Rectangle size  
    System.out.println(r2.toString()); // Rectangle size  
}
```

Question 3: Consider these two classes

```
public class Animal {
    private int age;
    private String name, color;

    public Animal(int age, String name, String color) {
        this.age = age;
        this.color = color;
        this.name = name;
    }

    public String toString() {
        return "Animal: Name: " + this.name + ", Age: " +
            this.age + ", Color: " + this.color;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

```
public class House {
    private String address;
    private Animal animal;

    public House(String address, Animal animal) {
        this.address = address;
        this.animal = animal;
    }

    public String toString() {
        return "House: Address: " + this.address
            + ",Contains: " + this.animal;
    }

    public String getAddress() {
        return this.address;
    }

    public Animal getAnimal() {
        return this.animal;
    }
}
```

Question 3:

What would be the output of:

```
class driver {  
    public static void main(String[] arg) {  
        Animal a = new Animal(2, "Emma", "Red");  
        House h1 = new House("Montreal", a);  
        a.setAge(3);  
        a.setName("Liam");  
        a.setColor("White");  
        House h2 = new House("Toronto", a);  
        System.out.println(h1 + "\n" + h2);  
    }  
}
```

Question 4: Coding exercise

Write a Student class which keeps track of grades and generates a final mark.

You should store:

- ▶ 3 quiz scores, an array of int between 0 and 20;
- ▶ 1 midterm score, an int between 0 and 50;
- ▶ 1 final score, an int between 0 and 100;
- ▶ the overallScore (double) and letter grade (char).

also create the appropriate accessor and mutator methods.

Question 5: Coding exercise (cont'd)

Student should also include the methods:

```
public void calculateOverallScore() { ... }
```

*Quizzes are worth 15% of the grade,
Midterm is worth 35% of the grade,
the Final is worth 50% of the grade.*

```
public void finalLetterGrade() { ... }
```

100 ~ 90: A 70 ~ 80: C 0 ~ 60: F

90 ~ 80: B 60 ~ 70: D

Comments in Java

3 types of comments in Java:

- ▶ `//` single line comments
- ▶ `/*` Multiple lines comment.
Useful to "erase" a block
of code from compilation `*/`
- ▶ `/**`
 - `*` JavaDoc comments
 - `*` Can be used to generate html documentation
 - `*/`

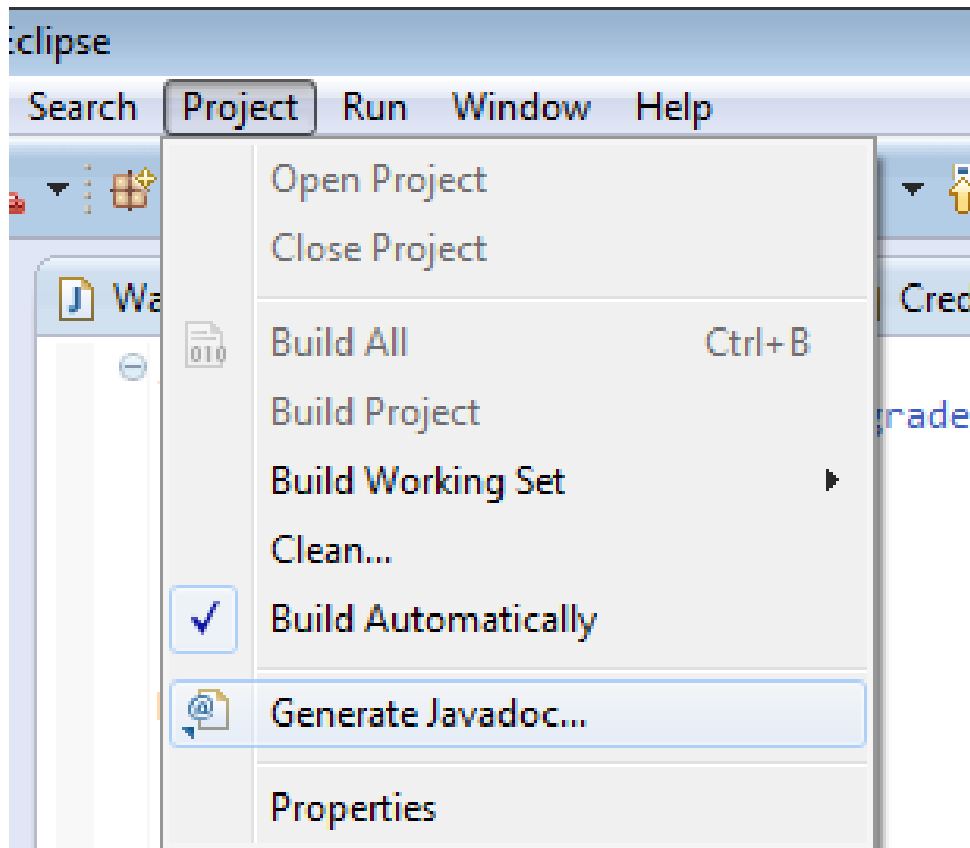
Javadoc: what is it?

- ▶ A standard for documenting Java programs
- ▶ A system which allows to attach descriptions to classes, constructors, fields, interfaces, and methods, in a generated HTML document.
- ▶ This is done by placing appropriate comments directly before the declaration of the item they describe:

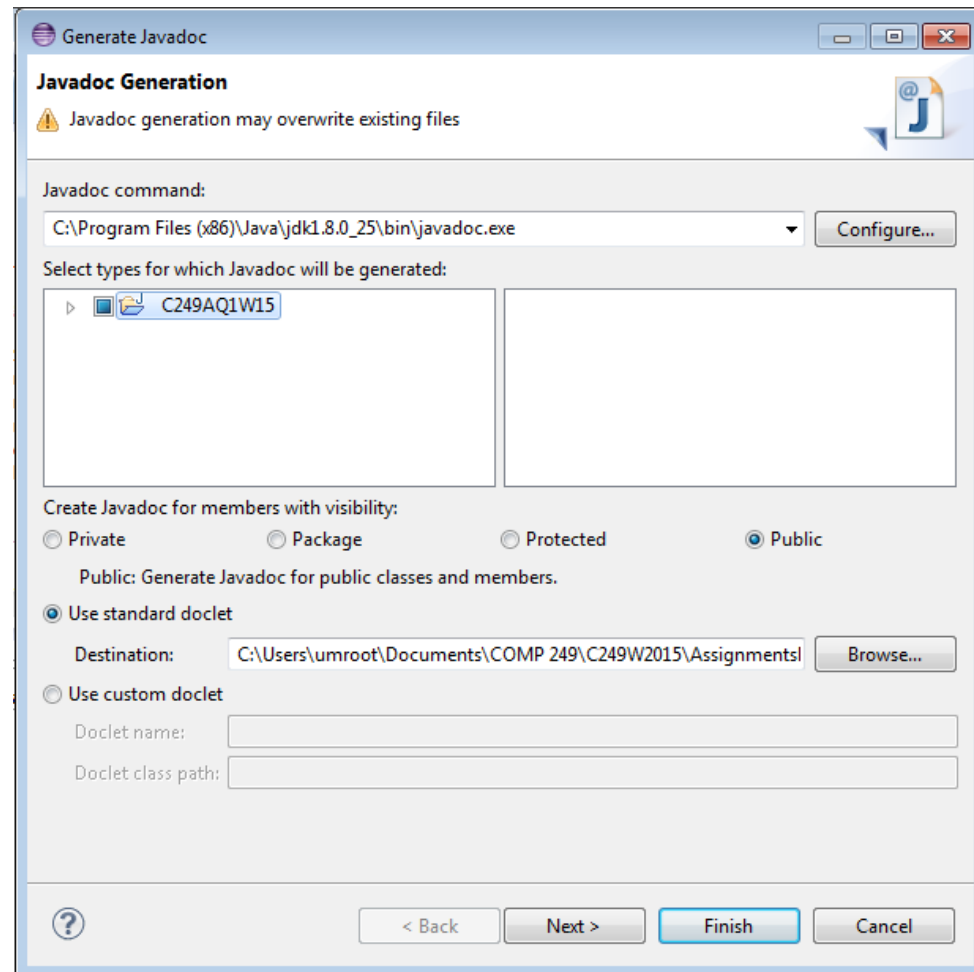
```
/**  
 * The Student class implements methods to  
 * calculate a student's grades  
 */  
public class Student { ... }
```

JavaDoc: generating documentation

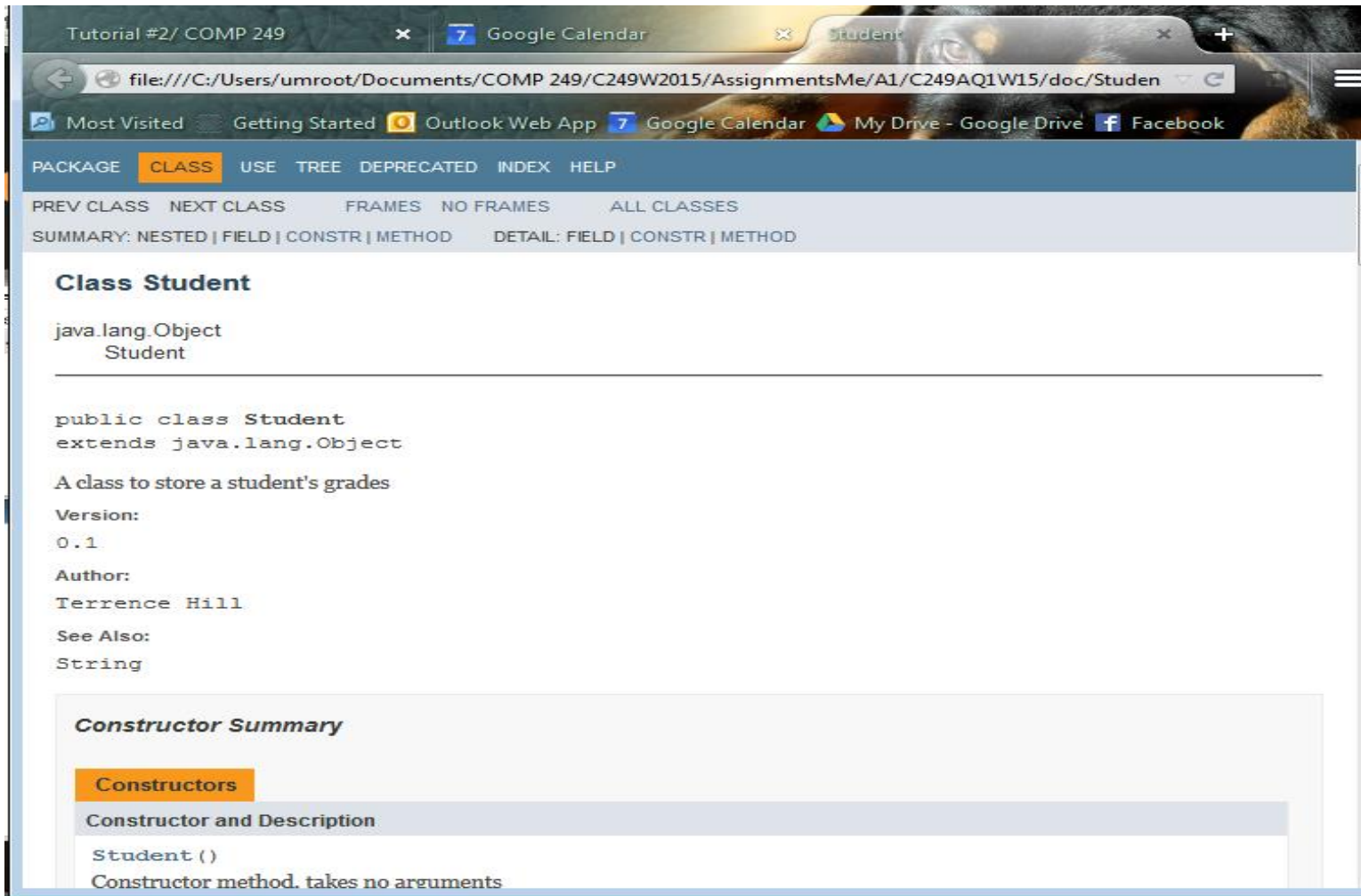
► In Eclipse



```
private double overallScore;  
private char letterScore;
```



JavaDoc: generated documentation



The screenshot shows a web browser window with multiple tabs. The active tab is titled "Student" and displays the JavaDoc documentation for the `Student` class. The browser's address bar shows a file path: `file:///C:/Users/umroot/Documents/COMP 249/C249W2015/AssignmentsMe/A1/C249AQ1W15/doc/Studen`. The page has a navigation bar with tabs for "PACKAGE", "CLASS" (which is selected), "USE", "TREE", "DEPRECATED", "INDEX", and "HELP". Below this, there are links for "PREV CLASS", "NEXT CLASS", "FRAMES", "NO FRAMES", and "ALL CLASSES". A summary bar indicates the current view: "SUMMARY: NESTED | FIELD | CONSTR | METHOD" and "DETAIL: FIELD | CONSTR | METHOD".

The main content area is titled "Class Student" and shows the inheritance hierarchy: `java.lang.Object` and `Student`. Below this, the source code for the class is displayed:

```
public class Student
extends java.lang.Object
```

A description follows: "A class to store a student's grades". Below the description, the version is listed as "0.1", the author as "Terrence Hill", and a "See Also" section points to `String`.

At the bottom, there is a section titled "Constructor Summary" with a sub-tab "Constructors" selected. It shows the constructor `Student()` with the description "Constructor method. takes no arguments".

JavaDoc: general form

```
/**
```

```
* One sentence ending with a period describing the  
purpose.
```

```
* Additional lines giving
```

```
* more details (html tags can be included)
```

```
*
```

```
* javadoc tags to specify more specific information,
```

```
* such as parameters and return values for a method
```

```
*
```

```
* @Tag Description ...
```

```
* @Tag Description ...
```

```
*
```

```
*/
```

Javadoc: Some tags

Tags are used to specify specific information in the HTML documentation.

Some common tags:

- ▶ For files, classes, and interfaces:
 - ▶ `@author` name
 - ▶ `@version` number
- ▶ For methods:
 - ▶ `@param` name description
 - ▶ `@return` description
 - ▶ `@exception` `exceptionClass` description
 - ▶ `@deprecated` description
- ▶ For everything:
 - ▶ `@see` `relatedReference` (ex. other class name)

Javadoc: Simple example

Here's a simple Javadoc comment describing a class:

```
/**
 * The Foobar class does things.
 * Amazing things, in fact.
 *
 * @author Bob
 * @version 1.1
 * @see String
 */
public class Foobar { ... }
```

Javadoc: Simple example

Here's a simple Javadoc comment describing a method:

```
/**
 * Takes two integers and uses them to make the
 * calculations.
 *
 * @param firstValue an integer value
 * @param secondValue another integer value
 * @return a double calculated from the two values provided
 */
public double makeCalculations(int firstValue, int
secondValue) { ... }
```

Question 6: Javadoc

Go back to your Student class and add Javadoc comments for each of your classes and methods, compile the documentation and take a look at the result.