Documenting your Java program

Yulia Newton, Ph.D.

CS151, Object Oriented Programming

San Jose State University

Spring 2020

Agenda

- Javadoc comments
- Code annotations

Why document our program?

- Good API documentation is essential for successful software
- Javadoc tool helps generating program documentation from comments
  - Standardized way of documenting code
  - Documentation is kept with the source (comments)
  - Documentation can be associated with packages, classes, methods, constructors, and fields
- Java Virtual Machine Specification
  - https://docs.oracle.com/javase/specs/

Commenting in your program

- Javadoc comments begin with /** … */

```
// single line comment

/*
multi-line comment
*/

/**
Javadoc comment
*/
```

Example: documenting HelloWorld with Javadoc

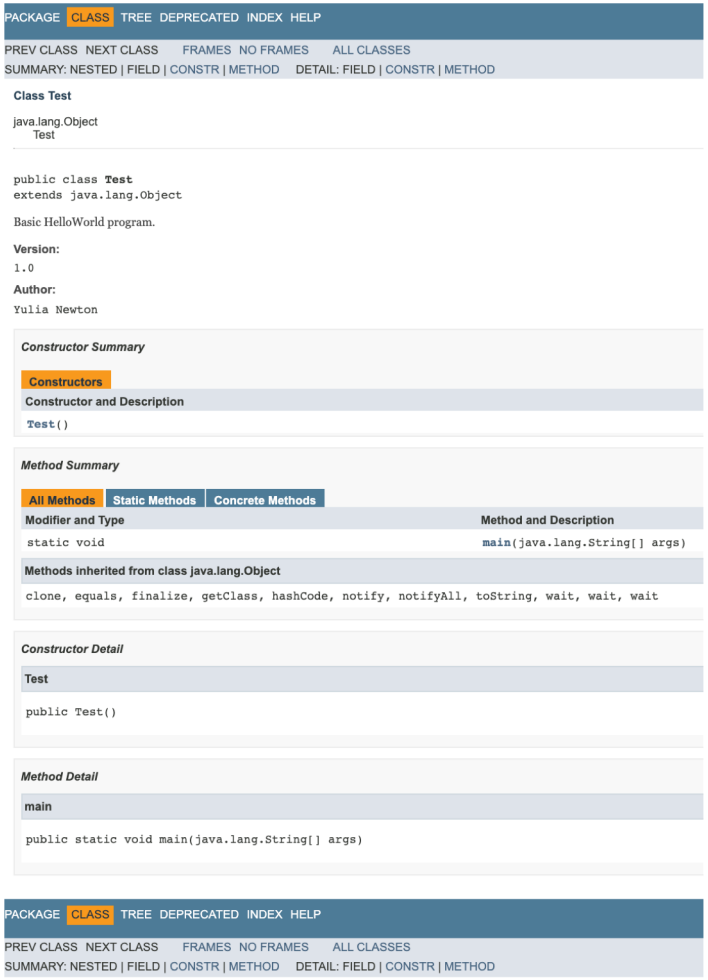
```java
/**
* Basic HelloWorld program.
*
* @author  Yulia Newton
* @version 1.0
* @since   2019-11-10
*/
public class Test {

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

Generating documentation using Javadoc tool

- On command line
  - javadoc Test.java
    - Use *–author* and *–version* options to include version and author information into hmtl output
    - Use *–d* option to specify output directory for documentation files
    - Use *javadoc –help* to see what else your command line Javadoc tool can offer
- Many IDE tools provide menu options for generating API documentation
- View documentation by opening index.html file in the browser of your choice
  - Or specify *–link* option in *javadoc* command

Example: documentation for HelloWorld program



PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Test

java.lang.Object
  Test

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

Basic HelloWorld program.

Version:

1.0

Author:

Yulia Newton

Constructor Summary

Constructors

Constructor and Description
`Test()`

Method Summary

All Methods **Static Methods** **Concrete Methods**

Modifier and Type	Method and Description
static void	`main(java.lang.String[] args)`

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Test

```
public Test()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Example: using HTML tags in Javadoc comments

```java
/**
* @author  Yulia Newton
* @version 1.0
* @since   2019-11-10
*
* <h1>Hello World program</h1>
* <h2>Introduction</h2>
* This is a hello world program, the first program any
* new CS student at San Jose State University writes when
* they start to learn Java.
* <h2>Compile instructions</h2>
* Compile by running the following command:
* <i>javac Test.java</i>
* <h2>Execute instructions</h2>
* Execute by running the following command:
* <i>java Test</i>
* <p>
* It is important to format your Javadoc comments in a way
* that the API documentation can display them in a clear
* and easy to understand manner. Put yourself into user's shoes!
*
*/
public class Test {

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

Output: …

public class **Test**
extends java.lang.Object

Since:

2019-11-10

Hello World program

Introduction

This is a hello world program, the first program any new CS student at San Jose State University writes when they start to learn Java.

Compile instructions

Compile by running the following command: *javac Test.java*

Execute instructions

Execute by running the following command: *java Test*

It is important to format your Javadoc comments in a way that the API documentation can display them in a clear and easy to understand manner. Put yourself into user's shoes!

Version:

1.0

Author:

Yulia Newton

…

Example: using @code tag

```java
/**
* @author  Yulia Newton
* @version 1.0
* @since   2019-11-10
*
* Hello World program
* <p>
* This is a hello world program, the first program any
* new CS student at San Jose State University writes when
* they start to learn Java.
* <p>
* Compile as {@code javac Test.java}
* <p>
* Run as {@code java Test}
*
*/
public class Test {

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

Output: …

public class **Test**
extends java.lang.Object

Since:

2019-11-10 Hello World program

This is a hello world program, the first program any new CS student at San Jose State University writes when they start to learn Java.

Compile as javac Test.java

Run as java Test

Version:

1.0

Author:

Yulia Newton

…

@code indicates text that will be displayed in code font; displays special characters without the need to escape them

Example: including code snippets into your Javadoc comments using *pre* HTML tag and @code Javadoc tag

```
/**
* @author  Yulia Newton
* @version 1.0
* @since   2019-11-10
*
* Hello World program
* <p>
* This is a hello world program, the first program any
* new CS student at San Jose State University writes when
* they start to learn Java.
* <p>
* <b>My previously defined Employee class:</b>
* <pre> {@code
 class Employee{
    private String name;
    private int id;
    private String employer;

    private Employee(){
        this.name = "";
        this.id = 0;
        this.employer = "";
    }

    public Employee(String name, int id, String employer){
        this.name = name;
        this.id = id;
        this.employer = employer;
    }

    public String getName(){return this.name;}
    public int getID(){return this.id;}
    public String getEmployer(){return this.employer;}

    public void setName(String name){this.name = name;}
    public void setID(int id){this.id = id;}
    public void setEmployer(String employer){this.employer = employer;}

    public String toString(){
        return this.name + " (" + this.id + ") at "+this.employer;
    }
 }
}</pre>
*
*/
public class Test {

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

text inside pre HTML tag preserves line breaks and spaces

public class **Test**
extends java.lang.Object

Since:

2019-11-10 Hello World program

This is a hello world program, the first program any new CS student at San Jose State University writes when they start to learn Java.

My previously defined Employee class:

```
 class Employee{
        private String name;
        private int id;
        private String employer;

        private Employee(){
                this.name = "";
                this.id = 0;
                this.employer = "";
        }

        public Employee(String name, int id, String employer){
                this.name = name;
                this.id = id;
                this.employer = employer;
        }

        public String getName(){return this.name;}
        public int getID(){return this.id;}
        public String getEmployer(){return this.employer;}

        public void setName(String name){this.name = name;}
        public void setID(int id){this.id = id;}
        public void setEmployer(String employer){this.employer = employer;}

        public String toString(){
                return this.name + " (" + this.id + ") at "+this.employer;
        }
 }
```

Version:

1.0

Author:

Yulia Newton

Example: including code snippets into your Javadoc comments using *pre* and *code* HTML tags

```
/**
* @author  Yulia Newton
* @version 1.0
* @since   2019-11-10
*
* Hello World program
* <p>
* This is a hello world program, the first program any
* new CS student at San Jose State University writes when
* they start to learn Java.
* <p>
* <b>My previously defined Employee class:</b>
* <pre><code>
 class Employee{
    private String name;
    private int id;
    private String employer;

    private Employee(){
        this.name = "";
        this.id = 0;
        this.employer = "";
    }

    public Employee(String name, int id, String employer){
        this.name = name;
        this.id = id;
        this.employer = employer;
    }

    public String getName(){return this.name;}
    public int getID(){return this.id;}
    public String getEmployer(){return this.employer;}

    public void setName(String name){this.name = name;}
    public void setID(int id){this.id = id;}
    public void setEmployer(String employer){this.employer = employer;}

    public String toString(){
        return this.name + " (" + this.id + ") at "+this.employer;
    }
 }
* </code></pre>
*
*/
public class Test {

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

code HTML tag does the same job

public class **Test**
extends java.lang.Object

Since:

2019-11-10 Hello World program

This is a hello world program, the first program any new CS student at San Jose State University writes when they start to learn Java.

My previously defined Employee class:

```
 class Employee{
        private String name;
        private int id;
        private String employer;

        private Employee(){
                this.name = "";
                this.id = 0;
                this.employer = "";
        }

        public Employee(String name, int id, String employer){
                this.name = name;
                this.id = id;
                this.employer = employer;
        }

        public String getName(){return this.name;}
        public int getID(){return this.id;}
        public String getEmployer(){return this.employer;}

        public void setName(String name){this.name = name;}
        public void setID(int id){this.id = id;}
        public void setEmployer(String employer){this.employer = employer;}

        public String toString(){
                return this.name + " (" + this.id + ") at "+this.employer;
        }
 }
```

Version:

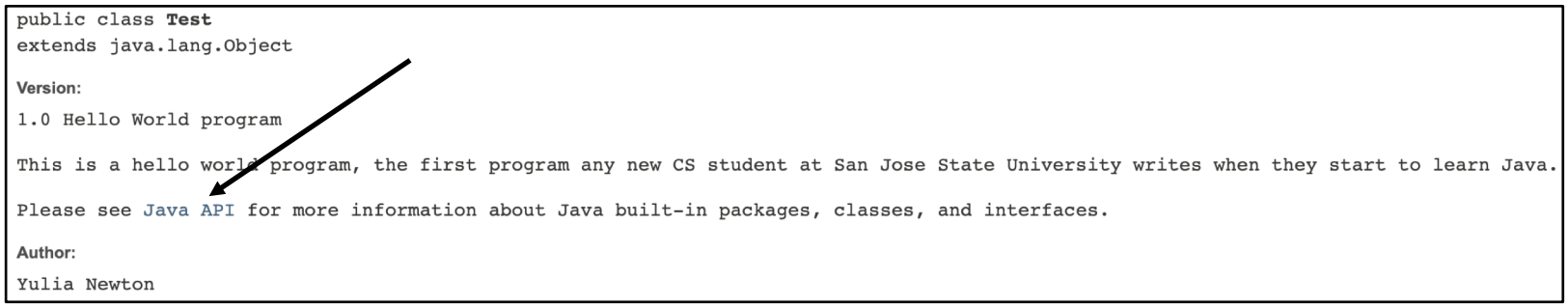
1.0

Author:

Yulia Newton

Example: including external links into Javadoc comments

```java
/**
* @author  Yulia Newton
* @version 1.0
*
* Hello World program
* <p>
* This is a hello world program, the first program any
* new CS student at San Jose State University writes when
* they start to learn Java.
* <p>
* Please see <a href="https://docs.oracle.com/javase/specs/jls/se13/html/index.html">Java API</a>
* for more information about Java built-in packages, classes, and interfaces.
*/
public class Test{
    public static void main(String[] args){
        //throws ArithmeticException{
       System.out.println("Hello World!");
       //throw new ArithmeticException();
   }
}
```



public class **Test**
extends java.lang.Object

Version:

1.0 Hello World program

This is a hello world program, the first program any new CS student at San Jose State University writes when they start to learn Java.

Please see Java API for more information about Java built-in packages, classes, and interfaces.

Author:

Yulia Newton

Example: adding “see also” section

```java
/**
* @author  Yulia Newton
* @version 1.0
*
* Hello World program
* <p>
* This is a hello world program, the first program any
* new CS student at San Jose State University writes when
* they start to learn Java.
* <p>
* Please see <a href="https://docs.oracle.com/javase/specs/jls/se13/html/index.html">Java API</a>
* for more information about Java built-in packages, classes, and interfaces.
* @see java.lang.String
*/
public class Test{
    public static void main(String[] args){
        //throws ArithmeticException{
      System.out.println("Hello World!");
      //throw new ArithmeticException();
   }
}
```

same as example above with this line added

public class **Test**
extends java.lang.Object

Version:

1.0 Hello World program

This is a hello world program, the first program any new CS student at San Jose State University writes when they start to learn Java.

Please see Java API for more information about Java built-in packages, classes, and interfaces.

Author:

Yulia Newton

See Also:

String

Example: documenting methods

```java
/**
* @author  Yulia Newton
* @version 1.0
*
* Hello World program
* <p>
* This is a hello world program, the first program any
* new CS student at San Jose State University writes when
* they start to learn Java.
*/
public class Test{
    /**
    * Divides the first floating point number by another floating point number.
    *
    * @param numerator numerator in the division operation
    * @param denominator denominator in the division operation
    * @return floating point result of the division
    * @exception ArithmeticException when denominator is zero
    * @author Yulia Newton
    */
    public double divide(double numerator, double denominator)
        throws ArithmeticException{
        if(denominator == 0){throw new ArithmeticException();}

        return numerator / denominator;
    }

    public static void main(String[] args){
        double a = 3.0;
        double b = 8.2;
        Test t = new Test();

        try{
            System.out.println(a+" / "+b+" = "+t.divide(a,b));
        }catch(ArithmeticException e){
            System.out.println(e);
        }
   }
}
```

can alternatively use *@throws* tag

Method Summary

All Methods	Static Methods	Instance Methods	Concrete Methods

Modifier and Type	Method and Description
double	**divide**(double numerator, double denominator) Divides the first floating point number by another floating point number.
static void	**main**(java.lang.String[] args)

Method Detail

divide

```
public double divide(double numerator,
                     double denominator)
              throws java.lang.ArithmeticException
```

Divides the first floating point number by another floating point number.

Parameters:

`numerator` - numerator in the division operation

`denominator` - denominator in the division operation

Returns:

floating point result of the division

Throws:

`java.lang.ArithmeticException` - when denominator is zero

Example: documenting fields

```java
import java.io.*;
/**
* @author  Yulia Newton
* @version 1.0
*
* Hello World program
* <p>
* This is a hello world program, the first program any
* new CS student at San Jose State University writes when
* they start to learn Java.
*/
public class Test{
    /**
    * private String attribute
    */
    private String privateString;
    /**
    * protected primitive int attribute
    */
    protected int protectedInt;
    /**
    * public FileInputStream attribute
    */
    public FileInputStream publicFileStream;
    /**
    * default access modifier double attribute
    */
    double defaultAccessDouble;

    public static void main(String[] args){
        System.out.println("Attributes in this class do nothing.");
    }
}
```

Field Summary

Fields

Modifier and Type	Field and Description
protected int	**protectedInt** protected primitive int attribute
java.io.FileInputStream	**publicFileStream** public FileInputStream attribute

Notice that private String and default access modifier double are not listed. This is because they are not accessible to classes and methods outside of this class (for String) and outside of this file (for double). They are irrelevant to your class API documentation.

Some other useful Javadoc tags

- @deprecated – adds explanation that this part of API should no longer be used
- @exception and @throws are synonymous
- @inheritDoc – inherits Javadoc comments from immediate superclass
- @link – adds a link to the documentation for another entity (package, class, attributes, methods)
- @value – to specify values of static attributes

Closing remarks about Javadoc comments

- Documenting your code is important!
- Any successful program is well documented
  - How are users supposed to use your code if they do not know what it does and how it does it?
- Add useful comments that will provide information to the user
  - Do not just retype what they can guess from method names and other available information
  - Maybe add use case scenarios if it helps to clarify the purpose and intended use of the class/method
- Use HTML tags to help format the text

Java annotations

- Used to provide metadata about your code
- Special instructions in your code
  - Annotations do not directly affect code execution
- *java.lang.annotation* package
- Java annotations are used for the following instruction types:
  - Compiler instructions
  - Build-time instructions
  - Runtime instructions
    - Can be read at runtime by your program code or third party programs
- Annotations are put in front of declarations
  - Class, method, parameter
    - Most often annotations are used sparingly
  - Preceded by “@” character
- Not equivalent to comments as they change the way annotated declarations are treated by compiler

Built-in JDK annotations

- *@Override*
  - Put in front of method declarations indicating that the method overrides a method with the same name and parameters in the superclass
  - If we put this annotation before method declaration without this method being available in the superclass we will get a compile error
  - Useful for when a method in the superclass changed, compiler catches this at compile time
- *@Depreciated*
  - Marks a method as depreciated
  - Gives a warning to a developer making a call to a method marked as depreciated that maybe they should use a different method
    - Depreciated does not mean it does not work, just no longer maintained
- *@SuppressWarnings*
  - Indicates to the compiler to suppress compile warnings
  - Also can specify the types of warnings to suppress *@SuppressWarnings("unchecked")* or *@SuppressWarnings("*overrides*")* or @SuppressWarnings({"checked", "deprecation"})
- *@Documented*
  - Indicates that annotations need to be documented by Javadoc
- *@Inherited*
  - Indicates that annotations for the superclass are inherited by the subclass
- *@Retention*
  - *Longevity of annotation retention (SOURCE – retained only in source, CLASS – retained at compile time only, RUNTIME – retained at runtime)*

Example: use of *@Depreciated* annotation

```java
class Animal
{
    @Deprecated
    public void introduce()
    {
        System.out.println("This method in Animal class has been depreciated");
    }
}

public class Test
{
    @SuppressWarnings({"deprecation"})
    public static void main(String args[])
    {
        Animal a1 = new Animal();
        a1.introduce();
    }
}
```

No warnings during the compile time

```java
class Animal
{
    @Deprecated
    public void introduce()
    {
        System.out.println("This method in Animal class has been depreciated");
    }
}

public class Test
{
    //@SuppressWarnings({"deprecation"})
    public static void main(String args[])
    {
        Animal a1 = new Animal();
        a1.introduce();
    }
}
```

Compile time warning:

```
Note: Recompile with -Xlint:deprecation for details.
```

Types of annotations

- Marker annotations
  - No value is specified in parentheses
    - Parentheses can be omitted
  - E.g. *@Override*
- Single value annotations
  - Single value is specified in paretheses
  - E.g. *@MyAnnotation(“test”)*
- Multiple value annotations
  - Specify multiple key-value pairs separated by comma
  - E.g. *@Ownership(name=“Yulia Newton”, company=“self”, purpose=“CS151”)*

Example: custom annotation

```java
import java.lang.annotation.*;

class Animal
{
    public void introduce()
    {
        System.out.println("I am animal");
    }
}

public class Test
{
    @Documented
    @Retention(RetentionPolicy.RUNTIME)
    @interface Ownership
    {
        String name();
        String company();
        String purpose();
    }

    @Ownership(name="Yulia Newton", company="self", purpose="CS151")
    public static void main(String args[])
    {
        Animal a1 = new Animal();
        a1.introduce();
    }
}
```

can have access modifiers, like public, etc.

Types of annotation fields can be anything (e.g. int, String[], etc.)

Output:

```
I am animal
```

Closing remarks about annotations

- Annotations provide special instructions for how to treat code associated with them
- They do not affect the logic of the code at run-time
- Help document the code through tags