Lesson 3-1: Implementing Classes

Computer Science 46A: Introduction to Programming

San José State University

Announcements

- Homework #1 is due by 2/17
  - Document says Homework 3 – ignore that
  - Need to use CodeCheck – instructions to use CodeCheck will follow
  - Builds on a previous Homework which we did not cover. I will provide solutions to that
- Lab #3 is this Friday (2/14)
- New reading assignment posted

Learning Objectives

By the end of this lesson, you should be able to:

1. Utilize a stepwise refinement approach to write classes
2. Comment Java code in a format that will generate Javadocs
3. Write generalized classes using instance variables, constructors, and methods

In Lecture 2-1 and 2-2:

We learned that classes have the following components:

1. Constructors
    - Used to create objects that store data
2. Methods
    - Used to access or modify the data in an object

We used classes such as `Day` in our own code

Today, we will start to write our own classes and test them

Example: A `Circle` class

- Suppose you wanted to create a class called Circle with the following:
  - A constructor that takes in the radius
  - A getCircumference method that returns the circumference
  - A getArea method that returns the area

- Check out `CircleTester` to see the expected behavior of the `Circle` class

Steps To Implement Your Own Class

1. Create an outline for your class
2. Document what each component of the class will do
3. Fill in the components of your class
4. Test the class to verify that it has worked as expected

1. Creating an outline for your class: Stepwise Refinement

Stepwise Refinement

- As you start to develop more complicated programs, you will have several objects, methods, and variables
  - This is a lot to keep track of!
- It's helpful to generate an outline of your program with the object constructors and methods and to successively fill them in
- Analogy: when you sit down to write an essay, you start with an outline with your thesis statement and key points for your introduction, body paragraphs, and conclusion
- Stepwise Refinement refers to the successive improvements you make to your program as you develop it

Stubs

- A stub is a piece of code that will help your code compile and run, but isn’t yet full developed
- Stubs can be used for constructors or for methods

Constructor Stub

Example: The Day class has the following constructor

```
public Day(int aYear, int aMonth, int aDayOfMonth)
{
   year = aYear;
   month = aMonth;
   date = aDayOfMonth;
}
```

As a stub, this constructor is identical, but without the contents:

```
public Day(int aYear, int aMonth, int aDayOfMonth)
{

}
```

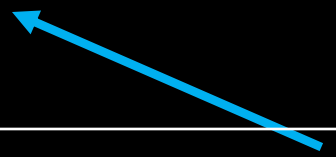
Method Stub

Example: The Day class has the following method

```
public int getYear()
{
    return year;
}
```

As a stub, this method is similar, but the main action of the method is not yet implemented:

```
public int getYear()
{
    return 0;
}
```



A method stub must still return the expected output type of the method or else the code will not compile

Poll Everywhere

Poll Everywhere: Q1

The Day class has the following method:

```
public void addDays(int numberOfDays)
{
   while (numberOfDays > 0)
   {
      nextDay();
      numberOfDays--;
   }
   while (numberOfDays < 0)
   {
      previousDay();
      numberOfDays++;
   }
}
```

Which of the following stubs will compile?

A)

```
public void addDays(int numberOfDays)
{
   return numberOfDays;
}
```

B)

```
public void addDays(int numberOfDays)
{
   return 1;
}
```

C)

```
public void addDays(int numberOfDays)
{


}
```

2. Documenting class components: Javadocs

Revisiting Documentation: Headers

When working with previous classes:

- The top comment block tells us what a class does

```
/**
 * A class represents days in the Julian/Gregorian calendar.
 *
 * @author  Cay Horstmann, Kathleen O'Brien, Qi Yang
 * @version 2022-01-12
 */
public class Day
```

Header comment block of the Day class

Always has @author and @version lines

Revisiting Documentation: Constructors and Methods

When working with previous classes:

- The comment blocks above constructors and methods tells us what their argument are and what they do

Comment blocks above constructors and methods have
@param tags for each input argument
@return tag for the output

```
/**
 * Constructs a day with a given year, month, and day
 * of the Julian/Gregorian calendar. The Julian calendar
 * is used for all days before October 15, 1582'
 *
 * @param aYear a year (any number other than 0)
 * @param aMonth a month between 1 and 12
 * @param aDayOfMonth a day of the month between 1 and 31
 */
public Day(int aYear, int aMonth, int aDayOfMonth)
{
    year = aYear;
    month = aMonth;
    date = aDayOfMonth;
}

/**
 * Returns the year of this day.
 * @return the year
 */
public int getYear()
{
    return year;
}
```

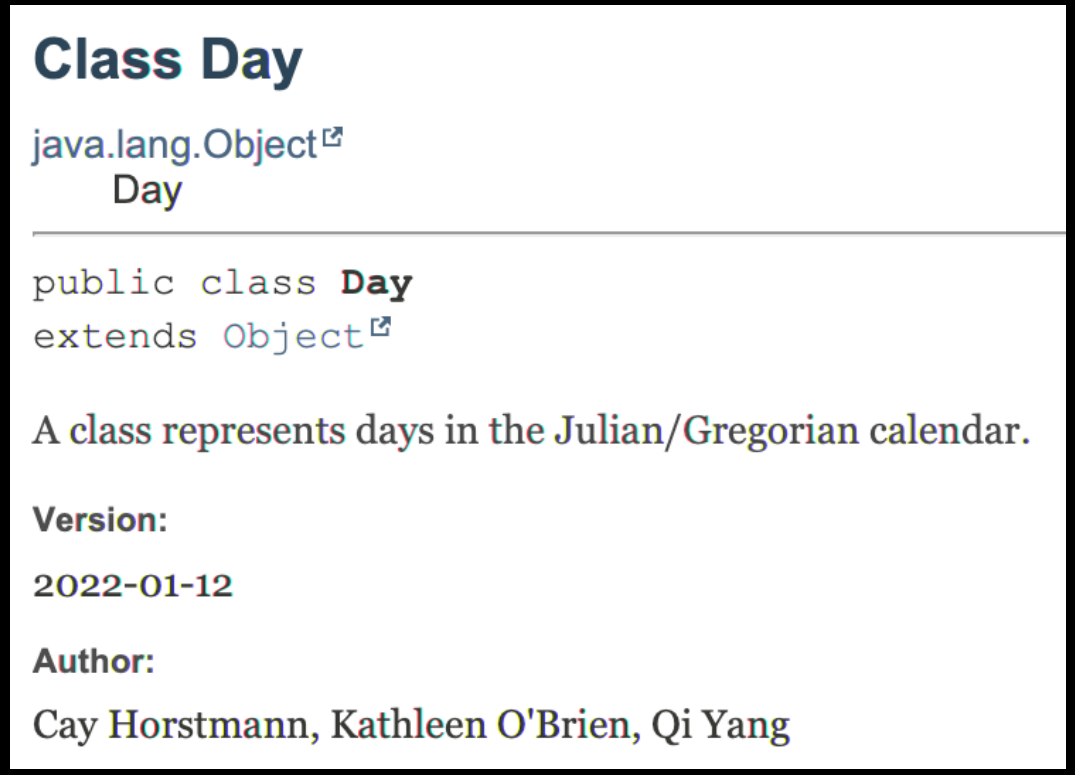
The Day constructor and getYear method in the Day class

Javadocs

- Java has functionality to turn comments in a code into easy-to-read documentation
- Th functionality will look for the following:
  - Class declaration
    - And header comment blocks with @author and @version
  - Constructors
    - And @param lines in the comment blocks
  - Methods
    - And @param and @return lines in the comment blocks

Javadocs in BlueJ

- Open BlueJ
- Open the Day class in the Editor (e.g. from par02-2)
- Press CTRL-J (command-J on iOS)



Class Day

java.lang.Object
    Day

```
public class Day
extends Object
```

A class represents days in the Julian/Gregorian calendar.

Version:

2022-01-12

Author:

Cay Horstmann, Kathleen O'Brien, Qi Yang

Javadocs header for the Day class

Poll Everywhere: Q2

The Day class has the following method:

```
public void addDays(int numberOfDays)
{
   while (numberOfDays > 0)
   {
      nextDay();
      numberOfDays--;
   }
   while (numberOfDays < 0)
   {
      previousDay();
      numberOfDays++;
   }
}
```

Which of the following is a correct header for this method?

A)
```
/**
 * Add a specified number of days to Day
 * @return numberOfDays the number of days to add
 */
```

B)
```
/**
 * Add a specified number of days to Day
 * @return Day the day object with days added
 */
```

C)
```
/**
 * Add a specified number of days to Day
 * @param numberOfDays the number of days to add
 */
```

Style Alert: Javadocs

When writing classes, all classes should be completed with documentation with Javadocs tags

- The header should have a descriptive statement and `@author` and `@version` tags
- Each method and constructor should have a descriptive statement and `@param` and `@return` tags as necessary

CodeCheck will look for your Javadocs!

Step 3: Filling in the components of your class

Instance Variables

- Instance variables carry the data in an object and have the following characteristics:
  - They are declared in the class *outside* of the constructor and methods
  - Their values are assigned by the constructor
  - They are accessible by all class methods
  - They are specific to the *instance* of an object
  - The come in public and private varieties
    - `public` instance variables are directly accessible from outside the class
    - `private` instance variables can only be modified within the class
  - Generally, we will declare instance variables as `private`

Encapsulation

- In Java, we practice Object Oriented Programming (OOP)
  - OOP is the idea that we use objects in our code to carry and manipulate data
- “Encapsulation” is one of the main pillars of OOP
  - Encapsulation is the idea that the data is contained in an object is accessed through the methods of the object

Instance variables for the `Circle` class

An object created by the `Circle` class keeps track of the radius

The `radius` instance variable has been declared under the class declaration – this will be filled in by the constructor and accessed by the methods

Filling in the constructor

- The arguments to the constructor are used to assign values to the instance variables
- The constructor should *assign* the input arguments to the instance variables so that the data is accessible to the methods

- Constructor declaration statement:

```
public [class name](input type inputs)
```

Filling in the methods

- The methods are functions which can access or mutate the instance variables

- Method declaration statement:

```
public [output type] [method name](input type inputs)
```

Constructors/Methods for the `Circle` class

An object created by the `Circle` class keeps track of the radius

The `radius` instance variable has been added in the class and is accessible by the methods (getCircumference and getArea)

When you create other classes, look at the constructor to determine what the type of the name variable should be

E.g. In the circle class, the constructor has an argument of type double and the instance variable also has type double

4. Verifying your class works as expected: Unit Tests

Units tests

- Unit tests are stand-alone pieces of code to test a component of your program
- In this class, we will use “Tester” code to test our classes

Next, you'll try it for yourself!

As a reminder, us the following steps to implement your class:

1. Create an outline for your class
2. Document what each component of the class will do
3. Fill in the components of your class
4. Test the class to verify that it has worked as expected

Submit Participation Exercise 3-1a: `Product`

Write a class called Product which has:
A constructor:

- `Product()`

Three methods

- `getName()`
- `getPrice()`
- `setPrice()`

Then, check your code with
`ProductTester.java`

Codecheck Link: HERE and on Canvas

```
iPad
Expected: iPad
iPhone
Expected: iPhone
450.0
Expected: 450.0
949.99
Expected: 949.99
405.0
Expected: 405.0
899.99
Expected: 899.99
```

Results of `ProductTester.java`

Submit Participation Exercise 3-1b: Student

Write a class called Student which has:

A constructor:

- `Student()`

Four methods

- `getName()`
- `getTotalScore()`
- `addScore()`
- `startAssignment()`

Then, check your code with `StudentTester.java`

Codecheck Link: HERE and on Canvas

```
37
Expected: 37
38
Expected: 38
Maria
Expected: Maria
Frank
Expected: Frank
0
Expected: 0
0
Expected: 0
Yes, Maria will start Par06 today!
Expected: Yes, Maria will start Par06 today!
Yes, Frank will start Par06 today!
Expected: Yes, Frank will start Par06 today!
10
Expected: 10
8
Expected: 8
Yes, Maria will start Hw03 today!
Expected: Yes, Maria will start Hw03 today!
Yes, Frank will start Hw03 today!
Expected: Yes, Frank will start Hw03 today!
37
Expected: 37
38
Expected: 38
```

Results of `StudentTester.java`