

Objectives

- Javadocs
- Inheritance
 - Final methods, fields
- Abstract Classes
- Interfaces

JAVADOCS

“Documentation is a love letter that you write to your future self.” – Damian Conway

Javadocs

- Special comments, which are used to generate HTML documentation
- Syntax:

```
/**
 * Comment
 */
```

- Put before a class, a method, or a field to describe the respective class/method/field

Sept 26, 2016

Sprenkle - CSCI209

3

Javadoc

- Can contain HTML syntax in description
- Example block tags to describe your code

`@param <paramname> <description>`

`@return <description> (include special cases)`

```
startsWith
public boolean startsWith(String prefix)
Tests if this string starts with the specified prefix.
Parameters:
prefix - the prefix.
Returns:
true if the character sequence represented by the argument is a prefix of the character sequence represented by this
string; false otherwise. Note also that true will be returned if the argument is an empty string or is equal to this
String object as determined by the equals(Object) method.
```

Sept 26, 2016

Sprenkle - CSCI209

4

Examples

```
/**
 * A simple Java class that models a Chicken. The
 * state of the chicken is its name, height, and weight
 *
 * @author Sara Sprenkle
 */
```

Tags always go last in Javadoc comment

```
/**
 * @return the height of the chicken, in centimeters
 */
```

```
/**
 * @param n the String representing the name of the
 * chicken
 */
```

Expect these types of comments on all methods from now on

Generating Javadocs

- From command-line:


```
javadoc [options] [packagenames]
[sourcefiles] [@files]
```
- Generates HTML files
 - E.g., Game's Javadocs

Summary of Inheritance

- Remove repetitive code by modeling the “is-a” hierarchy
 - Move “common denominator” code up the inheritance chain
- Don’t use inheritance unless *all* inherited methods make sense
- Use polymorphism

FINAL KEYWORD

Preventing Inheritance

- Sometimes, you do not want a class to derive from one of your classes
- A class that cannot be extended is known as a **final** class
- To make a class final, simply add the keyword **final** in front of the class definition:

```
public final class Rooster extends Chicken {
    . . .
}
```

- Example of **final** class: **System**

Sept 26, 2016

Sprenkle - CSCI209

9

Final methods

- Can make a method **final**
 - Any class derived from this class cannot override the **final** methods

```
class Chicken {
    . . .
    public final String getName() { . . . }
    . . .
}
```

- By default, **all** methods in a **final** class are **final** methods.

Why would we want to use **final**?
What are possible benefits to us, the compiler, ...?

Sept 26, 2016

ABSTRACT CLASSES

Sept 26, 2016

Sprenkle - CSCI209

11

Abstract Classes

- Some methods defined, others not defined
 - Partial implementation
- Classes in which not all methods are implemented are *abstract classes*
 - `public abstract class ZooAnimal`
- Blank methods are labeled as *abstract*
 - `public abstract void exercise(Environment env);`

Sept 26, 2016

Sprenkle - CSCI209

12

Abstract Classes

- An abstract class **cannot** be instantiated
 - i.e., can't create an object of that class
 - But can have a constructor!
- Child class of an abstract class can only be instantiated if it overrides and implements **every abstract method** of parent class
 - If child class does not override *all* abstract methods, it is **also abstract**

Sept 26, 2016

Sprenkle - CSCI209

13

Abstract Classes

- **static**, **private**, and **final** methods cannot be **abstract**
 - B/c cannot be overridden by a child class
- **final** class cannot contain abstract methods

Why?
- A class can be abstract even if it has no abstract methods
 - Use when implementation is incomplete and is meant to serve as a parent class for class(es) that complete the implementation
- Can have array of objects of abstract class
 - JVM will do dynamic dispatch for methods

Sept 26, 2016

Sprenkle - CSCI209

14

Examples of abstract classes

- Example 1:
 - `java.net.Socket`
 - `java.net.SSLSocket` (abstract)
- Example 2:
 - `java.util.Calendar` (abstract)
 - `java.util.GregorianCalendar`

Summary: Defining Abstract Classes

- Define a class as **abstract** when have *partial implementation*

INTERFACES

Sept 26, 2016

Sprenkle - CSCI209

17

Interfaces

- Pure specification, no implementation
 - A set of requirements for classes to conform to
- Classes can **implement** one or more interfaces

Sept 26, 2016

Sprenkle - CSCI209

18

Example of an Interface

- We can call `Arrays.sort(array)`
- `Arrays.sort` sorts arrays of any object class that implements the `Comparable` interface
- Classes that implement `Comparable` must provide a way to decide if one object is less than, greater than, or equal to another object

Sept 26, 2016

Sprenkle - CSCI209

19

`java.lang.Comparable`

```
public interface Comparable {  
    int compareTo(Object other);  
}
```

- Any object that is (inherits) `Comparable` must have a method named `compareTo()`
- Returns:
 - Return a negative integer if the this object is less than the object passed as a parameter
 - Return a positive integer if the this object is greater than the object passed as a parameter
 - Return a 0 if the two objects are equal

Sept 26, 2016

Sprenkle - CSCI209

20

Comparable Interface API/Javadoc

- Specifies what the `compareTo()` method should do
- Says which Java library classes implement **Comparable**

<http://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html>

Implementing an Interface

- In the class definition, specify that the class will **implement** the specific interface

```
public class Chicken implements Comparable
```

- Provide a definition for all methods specified in interface

How to determine Chicken order?

Comparable Chickens

One way: order by height

```
public class Chicken implements Comparable {
    . . .
    public int compareTo(Object otherObject) {
        Chicken other = (Chicken)otherObject;
        if (height < other.getHeight() )
            return -1;
        if (height > other.getHeight())
            return 1;
        return 0;
        // simpler: return height-other.getHeight()
    }
}
```

What if otherObject is not a Chicken?

Sept 26, 2016

Sprenkle - CSCI209

23

Interface Summary

- Contain only object (*not class*) methods
- All methods are **public**
 - Implied if not explicit
- Fields are constants that are **static** and **final**
- A class can implement multiple interfaces
 - Separated by commas in definition

Sept 26, 2016

Sprenkle - CSCI209

24

Testing for Interfaces

- Use the `instanceof` operator to see if an object implements an interface
 - e.g., to determine if an object can be compared to another object using the `Comparable` interface

```
if (obj instanceof Comparable) {
    // runs if obj is an object variable of a class
    // that implements the Comparable interface
}
else {
    // runs if it does not implement the interface
}
```

Sept 26, 2016

Sprenkle - CSCI209

25

Interface Object Variables

- Can use an object variable to refer to an object of any class that implements an interface
- Using this object variable, can *only* access the interface's methods
- For example...

```
public void aMethod(Object o) {
    ...
    if (obj instanceof Comparable) {
        Comparable comp = (Comparable) obj;
        boolean res = comp.compareTo(obj2);
    }
}
```

Sept 26, 2016

Sprenkle - CSCI209

26

Interface Definitions

```
public interface Comparable {  
    int compareTo(Object other);  
}
```

- Interface methods are **public** by default
 - Do not *need* to specify methods as **public**

Interface Definitions and Inheritance

- Can extend interfaces
 - Allows a chain of interfaces that go from general to more specific
- For example, define an interface for an object that is capable of moving:

```
public interface Movable {  
    void move(double x, double y);  
}
```

Interface Definitions and Inheritance

- A powered vehicle is also `Movable`
 - Must also have a `milesPerGallon()` method, which will return its gas mileage

```
public interface Powered extends Movable {  
    double milesPerGallon();  
}
```

Sept 26, 2016

Sprenkle - CSCI209

29

Constants in an Interface

- If a variable is specified in an interface, it is automatically a constant:
 - `public static final variable`

```
public interface Powered extends Movable {  
    double milesPerGallon();  
    double SPEED_LIMIT = 95;  
}
```

- An object that implements `Powered` interface has a constant `SPEED_LIMIT` defined

Sept 26, 2016

Sprenkle - CSCI209

30

Interface Definitions and Inheritance

- Powered interface extends `Movable` interface
- An object that implements `Powered` interface must satisfy all requirements of that interface as well as the parent interface.
 - A `Powered` object must have a `milesPerGallon()` and `move()` method

Sept 26, 2016

Sprenkle - CSCI209

31

Multiple Interfaces

- A class can implement multiple interfaces
 - Must fulfill the requirements of each interface

```
public final class String implements  
    Serializable, Comparable, CharSequence { ...
```

- But NOT possible with inheritance
 - A class can only extend (or inherit from) **one** class

Sept 26, 2016

Sprenkle - CSCI209

32

Benefits of Interfaces

- ??

Using an Interface or Abstract Class

*When should we use
an interface or an abstract class?*

Abstract Classes and Interfaces

- Important structures in Java
 - Make code easier to change
- Will return to/apply these ideas throughout the course
- Concepts are used in many languages besides Java