Lesson 6.03 — Interacting with the Object Superclass

Overview

Objectives — *Students will be able to...*

- **Replace** superclass behavior by writing overriding methods in the subclass.
- **Write** subclass methods that access superclass methods.

Assessments — *Students will...*

- **Complete** Practice questions
- **Complete** a worksheet

Homework — *Students will...*

- **Read** BJP 9.3 up to "Interpreting Inheritance Code."

Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of WS 6.3, Poster 6.3
- **Poster 6.3**
- **Empty & washed, or non-refrigerated, drink bottles**, with labels affixed (**optional**)
- Teacher access to CS Awesome **Unit 3 Lesson 07 Comparing Objects Lesson Plan** Sign up at CS Awesome AP CSA Java Curriculum
- Access to Dr. Nguyen **Compound Boolean Expressions and Comparing Objects** slide deck
- Access to CS Awesome **3.7. Comparing Objects**

Poster 6.3 is set to print a movie-sized poster of 15" x 20". If you do not want to print a poster this size, access the .pptx version of the poster, and reset the page size to legal, ledger, or whatever larger-format paper you have available to you. Note: Some fonts on this poster print incredibly small.

Arrange the drink bottles in a space where students can pick them up and look at them if need be. Encourage students to look at the bottles to get ideas for fields they can use in their *Drink* subclasses.

Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Review of the project	10min
Student practice	15min
Student practice: WS 6.3	25min

Procedure

Most of student practice today is a review and further integration of the concepts that were introduced since the beginning of this unit. The only new concept being drilled today is the equals method.

Bell-work and Attendance [5 minutes]

Review of the Project [10 minutes]

1. All classes are subclasses of the *Object* class. Whether you write extends in the header or not, all classes inherit the *Object* class. It is built into Java this way, so you never have to explicitly write *extends Object* in a class header.

- This means that all code inherits some generalized methods that come automatically with the *Object* class.

2. The AP exam covers *toString* and *equals* methods only.
   - Ask students if they remember what *toString* method does. (It gives the class name followed by a location in memory, which isn't very helpful, so we always create our own *toString* methods when we create a class.)
   - Ask students if they remember why we can't just use == to test for equality between objects.
3. The == operator tests whether two objects have the same identity, if they refer to the same object, not whether the two objects are in the same state.

```
String z = "z";
String a = z + z;
String b = "zz";

a == b;          // Evaluates to false because a and b refer to different Strings

String c = b;
c == b;          // Evaluates to true because c and b refer to the same String
```

4. The default *equals* method (that comes with your *Object* superclass) does the same thing as the == operator. (Since the *equals* method comes from the *Object* superclass, it interprets its input parameter as an object.) But for Strings, the equals method does something smarter:

```
a.equals(b);     // Evaluates to true because the content of a and b are the same "zz"
c.equals(b);     // Evaluates to true because c and b refer to the same String
```

   - To rewrite an *equals* method that compares object state (to override the *Object* version of the equals method), you need to cast the object in order to let Java know that the objects really can be compared.
   - To test if two *Drink* objects have the same name and serving size, you would write an *equals* method that looks like this:

```
public boolean equals(Object o) {
    Drink other = (Drink) o;
    return name.equals(other.name) && (ounces == other.ounces);
}
```

5. Comparing Objects with == and *!=*
   - See CS Awesome Unit 3 Lesson 07 Comparing Objects Lesson Plan
   - Slides 22-25 of Compound Boolean Expressions and Comparing Objects can be used to introduce this topic
   - Have students navigate to CS Awesome 3.7. Comparing Objects, view the Activity 1 video and complete Activities 1-7.

Student Practice: [15 minutes]

1. Have students work individually or in pairs to complete the following Practice questions:
   a. Self-Check 9.3: subclassSyntax
   b. Self-Check 9.10: inheritanceVariableSyntax
   c. Self-Check 9.8: CarTruck
   d. Self-Check 9.9: CarTruck2

Student Practice: WS 6.3 [25 minutes]

Emphasize with students...

Content - Problem decomposition - Advanced programming structures - Management of complexity
Now that you are creating superclasses and subclasses you are starting to program using quite advanced programming structures. These structures are sometimes difficult concepts for students to grasp, but once understood, they can actually facilitate programming.

The creation of superclasses and subclasses allows you to break your program down into smaller chunks (this is called decomposition). It also allows you to manage complexity by having methods and attributes exist within the class. This keeps the data separate from the main program, but of course it can be accessed or altered whenever necessary.

Once students have completed these exercises, distribute worksheet 6.3.

- Read through the questions aloud, if needed.
- If you are having the students work in Eclipse, be sure to review your procedure for submitting work electronically before students begin.
- Encourage students to explore the drink bottles to get ideas for fields they can use in their *Drink* subclasses.

Accommodation and Differentiation

At this point of the course, introducing TextExcel may be beneficial. Although the students don't have the required knowledge to complete the project (yet), TextExcel and can be useful when explaining polymorphism in the following lesson because of how cells are displayed. If you do decide to go this route, it's a good idea to give your class at least a half-day to work on TextExcel to get a better understanding of the prompt and where they get stuck in the project.

Encourage advanced students to add additional classes, fields, methods, and client code. If students still have time to spare, encourage them to increase code complexity, add additional levels to the class hierarchy, or help their peers.

If you have a few students that are struggling with the assignment, allow them to work in groups of 4, each pair helping the other with their code. If students need additional guidance, have students complete the worksheet as a series of think-pair-shares, where you return to whole group to share and discuss answers before moving on to the next step. **Teaching the class this way will roughly double the time required to complete the exercise.**

Misconceptions

Students often have confusion on the difference between overriding vs overloading methods. The following is a chart of the differences:

	OverridingMethod with same name, parameters, and return type	OverloadMethods with same name but parameters and/or return type
Method Signature	Same	Different
Class hierarchy	Subclass overrides superclass	Overloaded method can exist anywhere in class hierarchy
Behavior	Change behavior of superclass method	Multiple behavior
Execution	Run Time	Compile Time

Video

- CSE 143, *Interacting with Objects* (2:59–30:39) https://www.youtube.com/watch?v=ewsM5Iak83g&start=179

- CSE 143, *Interacting walkthrough* (31:38–40:10) https://www.youtube.com/watch?v=ewsM5Iak83g&start=1900

Forum discussion

Lesson 6.03 Interacting with the Object Superclass (TEALS Discourse account required)