TEALS Program

1 Lesson 6.01 — Inheritance Basics

1.1 Overview

1.1.1 Objectives — *Students will be able to…*

- **Correctly define** inheritance.
- **Use** proper syntax to extend a class.
- **Illustrate** is-a relationships.
- **Properly implement** constructors of derived classes using super.

1.1.2 Assessments — *Students will…*

- **Complete** a Class Hierarchy poster as indicated in WS 6.1

1.1.3 Homework — *Students will…*

- **Read** BJP 9.2 up to "DividendStock Behavior."
- **Collect** images that represent instances of the classes created for in-class poster project.

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 6.1] Start class poster, [Example 6.1]
- **Pictures** of Pokémon (http://tinyurl.com/l6mybmr) or Pokémon Cards (optional)
- **Video** about bees (https://www.youtube.com/watch?v=K30MN1a_pdg)
- **Student pair assignments**
- **Art supplies** for each group:
  - Poster paper or cut sheets of butcher paper
  - Lined paper (at least 3 sheets per group)
  - Markers
  - Glue Sticks
  - Old magazines, flyers, newspapers to cut up for collage
  - Scissors
  - Yarn, string, or embroidery floss
  - Tape, magnets, or tacks to hang finished work

Most of the supplies required for this lesson are readily available in high schools. If the school doesn't have poster paper, butcher paper works well. Other supplies may be available to borrow from the Math, Science, or Art teachers. To get an idea what a final student project should look like, check out the picture of sample student work "Example 6.1."

1.3 Pacing Guide: Day 1

Section	Total Time
Bell-work and attendance	5min
Introduction	20min
Review of the project	5min
Student work	25min

1.4 Pacing Guide: Day 2

Section	Total Time
Student work & teacher check	20min
Peer review	10min
Whole-group discussion and reteach if needed	15min
Quiz	5min

1.5 Procedure

Hook your students by prominently displaying art materials and sample work (of your own making, or saved from a previous year). To feature the most engaging student examples, look for work that has many instances of each class and uses classes/objects that are popular with your class. Invoke an air of mystery and don't offer an explanation for any of it.

1.5.1 Bell-work and Attendance `[5 minutes]`

1.5.2 Introduction `[20 minutes]`

1.5.2.1 Emphasize with students...

1.5.2.2 Big Ideas - Products can be designed for life cycle

This activity will take you through the process of designing superclasses and subclasses. As you do so, consider carefully how your superclass could be reused and repurposed, later, by another programmer to create something different.

Many of the programs you create could be altered, remixed and tweaked to create new and innovative software that is slightly different than the original. This is a great thing about Object Oriented Programming. Much of the code can be reused and repurposed to create things we haven't even thought of yet.

---

1. Have a quick class discussion about bees with the students jotting down some notes on the board.
2. Watch the video about bees and ask again what they now know adding to the notes.
3. Ask them to help you create a Bee class with attributes (ex: boolen carryingPolen) and methods (ex: fly(), gatherNectar()). This does not need to be compilable code — a UML class diagram would work well here.
4. They may come up with this on their own, but steer them in the direction of multiple types of bees — worker, queen, drone (male). Create another UML box, but don't connect them yet.
5. Ask what all bees have in common. Write everything down even if it doesn't have a perfect programming analogue (e.g. yellow stripes) or if it isn't something that all bees do (e.g. sting or gather nectar).
6. Start explaining the idea of superclasses and subclasses. Explain that subclasses are specialized versions of the superclasss. Suggest that the Bee class could be the superclass.
7. Ask students what the specialized subclasses would be. Start mapping out the heirarchy with arrows. Explain that the arrow represents an "is a" relationship. Give other examples of "is a" relationships

1.5.2.3 Examples

- An apple tree is a tree.
- A stegasoraus is a dinosaur.
- An electric Pokémon is a Pokémon.
- A computer science student is a student.
- A math teacher is a teacher.
- Soda is a drink.
- A square is a rectangle.
- A smart phone is a computer.

8. Have students give examples of other things have have this relationship and ask them to point out the superclasses and subclasses. Allow examples to have multiple subclasses.

9. Ask students to define an inheritance hierarchy in their own words. Briefly discuss why you would want to use inheritance in programming.
    - An **inheritance hierarchy** is a set of hierarchical relationships between classes of objects.
    - **Inheritance** is a programming technique that allows a derived class to extend the functionality of a base class, inheriting all of its state and behavior.)
    - **Superclass** is the parent class in an inheritance relationship.
    - **Subclass, or child class** is the derived class in an inheritance relationship.
10. Check for understanding by returning to the examples above, and asking students to give an example of some characteristics (fields) the parent class would have, and what characteristics students would add to the specialized subclasses.

    Example: Drinks could have a String *name* and boolean *carbonated*, and Soda could add a boolean *caffeinated*.
11. The class header for a subclass that extends the functionality of the parent class looks like this:

```
public class Mammal extends Animal {

public class Motorcycle extends Vehicle {

public class Churro extends Pastry {
```

- Point out that the subclass names are capitalized by convention, and that you always use the *extends* keyword. Give students a moment to think of a few hierarchical relationships in a think-pair-share, and ask several volunteers to come to the front of the room to demonstrate the correct class header.

12. For the following example, we will create subclasses that extend the *Vehicle* superclass written below. Show the slide with this code and/or create a new file with the code below. If you think your students can come up with the methods relatively quickly, you can show portions of the code and have them fill in the methods in pairs.

```
class Vehicle {
 static final int FEET_IN_MILE = 5280;
 private int xCord;
 private int yCord;

 private double fuelGallons;
 private double maxFuel;
 private double mileFuelRatio;

 public Vehicle(int xStart, int yStart,  double fuelGallons, double maxFuel, double milePerGallon){
   this.xCord = xStart;
   this.yCord = yStart;

   this.fuelGallons = fuelGallons;
   this.maxFuel = maxFuel;
   this.milePerGallon = milePerGallon;
 }

 public void move(int dx, int dy){
   // if there is enough fuel, move dx and dy feet.
   double distance = Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2));
   double fuelUsed = distance / (FEET_IN_MILE * milePerGallon);

   if (fuelGallons - fuelUsed > 0){
     this.xCord += dx;
     this.yCord += dy;
     this.fuelGallons -= fuelUsed;
   } else {
     System.out.println("Not enough fuel.");
   }
 }

 public void refuel(double fuel){
   this.fuelGallons += fuel;
   if (this.fuelGallons > this.maxFuel){
     this.fuelGallons = this.maxFuel;
   }
 }

 public int getX(){
   return xCord;
 }

 public int getY(){
   return yCord;
 }

 public double getFuel(){
   return fuelGallons;
 }

  }
```

13. Explain that this is a superclass and you wouldn't necessarily instantiate something as simply a vehicle. Ask for examples of subclasses. Write down all examples — even if someone says airplane right away — but tell the class that to keep it simple you'll start with a couple land-based examples like car and truck. Here are some examples that are in the slide. Feel free to hide that slide and make these classes with the students. Make sure you are repeating OOP vocab like inheritance, “is a”, superclass, subclass, constructors, etc.

```
class Car extends Vehicle {

  public Car(int xStart, int yStart){
```

```
    super(xStart, yStart, 15, 15, 20);
  }
}
```

``` Java class Semi extends Vehicle {

public Semi(int xStart, int yStart){ super(xStart, yStart, 130, 130, 6.5); }

}

``` Java
class Motorcycle extends Vehicle {

  public Motorcycle(int xStart, int yStart){
    super(xStart, yStart, 5, 5, 50);
  }
}
```

14. When you have a couple land-based examples down, ask students to create a vehicle subclass and constructor that calls the superclass. Good examples could be Tractor, PickupTruck, SportsCar, SUV, Tank. Advanced students can try something that doesn't have fuel like Bicycle, Skateboard, HorseDrawnCarriage.
15. Ask them to share their classes with a neighbor, and optionally share with the class.
16. Bring them back together and ask how they would start an Airplane class. Ask them what attributes and methods they will need in the Airplane class that the Vehicle class doesn't have. Here is an example that is also in the slide deck:

    ``` Java class Airplane extends Vehicle {

    private int zCord; private boolean landingGear;

    public void liftoff(int dx, int dy, int dz){ if (zCord > 0){ System.out.println("We're already flying!"); return; }

    // if there is enough fuel, move dx, dy, and dz feet. double distance = Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2) + Math.pow(dz, 2)); double fuelUsed = distance / (5280 * milePerGallon);

    if (fuelGallons - fuelUsed > 0){ xCord += dx; yCord += dy; zCord += dz; fuelGallons -= fuelUsed; landingGear = false; } else{ System.out.println("Not enough fuel."); } } } ```

    Your students may point out different attributes and methods — write down ideas as they come and allow other students to edit them to function better. Emphasize that designing classes is not an exact science and there are lots of right answers. Explain that they should expect to have to restructure their classes as they iterate through the design process.

    If you have time, ask the students to write the constructor for Airplane in pairs.

    Note: some students may notice we are accessing private attributes in Airplane (xCord and yCord), but explain that we will have some fixes for that later.

### 1.5.3 Review of the Project [5 minutes]

1. Briefly review the assignment with your students, reading the directions aloud if need be.
2. If you haven't already distributed project materials at this point, do so while your students are rearranging into partner pairs.

### 1.5.4 Student Work [25 minutes]

1. Encourage students to take 5–10 minutes on Step 1. They should review all steps of the project to ensure that their selection of classes lends itself to the project (e.g. they shouldn't pick something they don't know a lot about because they'll have trouble coming up with fields and methods).

2. Offer time checks every 10 minutes so students can stay on pace. By the end of the first day, they should have gotten to step 6 or 7. Visit each group to make sure that they haven't veered off course.
3. On **day two**, check student work and help students display their work around the room.
4. Check that the flow-of-control string (see WS 6.1 for explanation) correctly shows how a method is passed through subclasses to the superclass.
5. Remind students to take notes (Step 11 on WS 6.1) to help them remember talking points for later in the class.
6. As a whole group, ask students to volunteer what they really liked about others' projects. Solicit questions and critiques, re-teaching if needed.
7. Administer quiz 6.1 to assess student understanding.

## 1.6 Accommodation and Differentiation

Encourage advanced students to add additional classes, fields, methods, and client code. If students still have time to spare, encourage them to read on method overriding, and invite them to add that code as well. Students can attach this "extra code" using paper and tape/glue or sticky notes.

If you have a few students that are struggling with the class, give them your starter Vehicle class and some subclasses, and let them build off of your examples. You can print out your starter code and cut it into pieces and shuffle them so students have to place each line in the correct location (as with a Parson problem).

If your students need further instruction on calling a superclass' constructor, go through a few more examples using the Vehicle superclass, or classes that you created as a whole group.

## 1.7 Teacher Prior CS Knowledge

- The Object Oriented Programming (OOP) paradigm could be thought of as mimicking the real world where objects consists of data that define them and actions that can be performed on the data. As you will see, the process of learning OOP is infinitely more complex.
- The pillars of Object Oriented Programming (OOP): inheritance, encapsulation, and polymorphism. In a nutshell inheritance allows for code reuse by defining methods once in a superclass, encapsulation provides data security by hiding data implementation from the user and only allowing methods in the class to modify the data, and polymorphism offers flexibility to the designer by way of methods defined in many forms.

## 1.8 Misconceptions

Students' use of "inheritance" prior to computer science are in the context of inheritance from an ancestor and genetic inheritance of traits. However, in computer science, inheritance is used for classification where the class that inherits (extends) from a more general class (super class). Neither of the students' prior knowledge and use of inheritance is an accurate representation of Java's class structure.

## 1.9 Common Mistakes

Object oriented concepts common mistakes: http://interactivepython.org/runestone/static/javareview/oobasics/oomistakes.html

## 1.10 Video

- BJP 9-1, *Inheritance: Interacting with the Superclass*
  http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c9-1
- CSE 142, *Inheritance* (23:28–35:06)
  https://www.youtube.com/watch?v=WPdv8X291hE&start=1408
- CS Homework Bytes, *Inheritance, with Zach*
  https://www.youtube.com/watch?v=Alv2ApK_jdo

## 1.11 Forum discussion

Lesson 6.01 Inheritance Basics (TEALS Discourse account required)