LATHROP ENGINEERING

Name:

UNIT 8: INHERITANCE & POLYMORPHISM

AP Computer Science A Unit Due Date: **February 15, 2019**

Welcome to the eighth unit of *AP Computer Science*! This unit will cover two great topics in computer science: inheritance and polymorphism. These tools will give us a few new keywords in Java that make it possible to create really cool class structures and class interactions. In the end, the expectation is that you learn the following:

- What *Inheritance* is, and what it makes possible within a Java program
- What the keywords 'extends', 'super', and 'abstract' do and how to take advantage of them
- What *Polymorphism* is, and what it makes possible within a Java program
- What an interface is and what the keyword 'implements' does

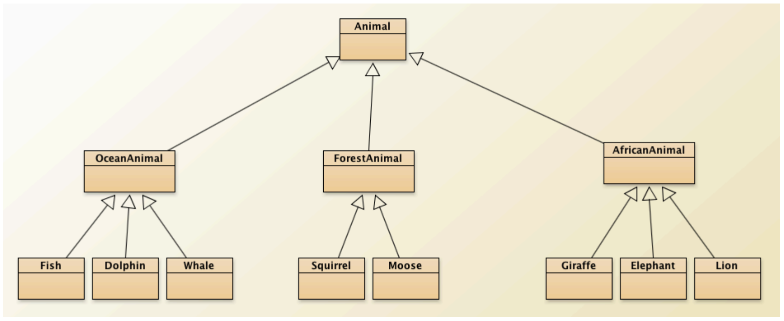
As we move through this unit, you are responsible for making adequate progress through the assignments, and for being done by the Unit Due Date (**February 15, 2019**). You are also responsible for completing each part before moving on to the next. Our unit is broken up into three main parts:

Part 1: **Inheritance** *(30 pts) Approx. 3 days*		
This unit begins with the introduction of *Inheritance*: a method by which we can make some classes receive information from parent classes. This will let us build some really cool connections between classes. In this part of the unit we'll learn how to use the 'extends', 'super', and 'abstract' key words to build higher order logic into the structure of our class organization.		☐ Inheritance Notes Notes
		⊕ Complete 6 Inheritance Tasks
		☆ Check-off from Mr. Benshoof

Part 2: **Polymorphism** *(30 pts) Approx. 3 days*		
The second part of our unit will introduce a type of class called an 'Interface'. Interfaces let us build reusable chunks of code in new ways and let us create frameworks that new classes need to fit within. This will help us learn to 'overload' methods in new ways so that we can make more flexible and adaptable parts of larger programs. Together, Inheritance and Polymorphism are the concepts that will make big, awesome		☐ Polymorphism Notes
		⊕ Complete 6 Polymorphism Tasks
		☆ Check-off from Mr. Benshoof

Part 3: **Video Game Design** *(20 pts) Approx. 3 days*		
To wrap up this unit, we finally get to start doing the big thing you've been wanting to do all year: *design your own video game*. We still won't deal with graphics (graphics are a nightmare – we'll do some next unit), but we'll get to build a complete and complex framework for a large computer game. We'll use inheritance and subclasses, interfaces, and abstract class/methods to create a framework that could turn in to the next super awesome video game!		☐ Video Game Design Brainstorm
		⊕ Complete Video Game Framework
		☐ Write Log 7: *Inheritance & Polymorphism*
		☆ Check-off from Mr. Benshoof

(30 pts) Approx. 3 days



This first part of the unit looks at a cool tool for making classes talk and interact with each other: Inheritance. In the same way that you inherit traits from your parents, 'sub' classes can inherit methods, variables, and instructions from other 'parent' classes. In the example on the right, the solid-line arrows suggest inheritance relationships. Logically, Fish (in the bottom row) are a type of "OceanAnimal", and the inheritance relationship shows that. Similarly, "OceanAnimal" is a more specific kind of "Animal". The great thing about this system, is that any method made in the "Animal" class at the top, automatically exists in every class below it… This lets us be very strategic about where methods and variables are made!

1. Take a full page of good notes on the presentations *Interface 1, Interface 2, and Inheritance Tutorial*. Take a good page of notes on these ideas and our class discussions. In particular, look for details on "extends" and "super"!
2. Now, complete the next 6 Java Tasks ***in a new project so you have a clean slate to build on!***
   a. JAVA TASK 59: Make a program that has a class called "Vehicle" that is relatively generic but includes 4 variables of different types. Then, add at least 5 subclasses and properly create their constructors to set the 4 variables in the correct way for that type of vehicle.
   b. JAVA TASK 60: Make a program that starts with a class called "2DShape" (as in geometry) with 3 variables and a constructor. Then, create at least 6 more subclasses with at least 2 levels. Each of these subclasses should have their own constructor that uses the super() constructor to set the included variables appropriately.
   c. JAVA TASK 61: Make a program that creates a parent class called "SchoolPerson" that has 3 variables. Then, create the "Student", "Teacher", "Counselor", and "Principal" classes as subclasses of SchoolPerson. Each of these 4 subclasses should have at least 1 unique variable specific to that class only. Create the proper constructors.
   d. JAVA TASK 62: Make a program that has a main parent class called "Animal" that contains 3 variables and 2 methods. One of the methods should be the toString() method. Then create 6 more classes that fit into 2 more levels of a class hierarchy, and override the toString() method in each. Also create all appropriate constructors.
   e. JAVA TASK 63: Make a program that starts with a parent class called "VideoGameCharacter" and has 4 generic variables as well as 3 generic methods – one of which should be the toString() method. Then, add 12 new classes that build a class hierarchy. These new classes should each have their own unique variable as well as an overridden toString() method.
   f. JAVA TASK 64: Make a program that demonstrates class hierarchy and inheritance with at least 8 different classes organized into at least 3 levels.

Part 1: Tasks	*10-8 points*	*7-4 points*	*3-0 points*
Inheritance Notes	+ Watch all presentations + You took a full page of notes on Inheritance topics + Your notes include details about the keywords 'extends', 'super', 'abstract'.	- Less than a full page of notes on Inheritance - Your notes are missing important parts	- Very brief or no notes in your notebook
	20-15 points	*14-8 points*	*7-0 points*
Java Tasks 59-64	+ You completed all 6 Java Tasks from this section	- You did not complete all 6 tasks	- You did not complete any tasks
Checkoff from Benshoof	+ Mr. Benshoof got to see your programs run successfully	N/A	N/A

UNIT 8: INHERITANCE & POLYMORPHISM PART2: POLYMORPHISM

(30 pts) Approx. 3 days

The next part of our unit is all about using interfaces to build in new connections between your classes. Interfaces let us setup a list of methods that must be overridden by any class that wants to use that interface. This helps classes of different types talk the same language when they try to interact with one another! The use of interfaces requires our next (and possibly last) java keyword for the year: *implements*

In order to implement an interface, we simply add to the class signature the word "implements" and then the name of any interfaces that the class implements. For example:

public class Panda extends ZooAnimal implements herbivore, cuddly, photogenic

In the above example, the "Panda" class **extends** the ZooAnimal class, inheriting relevant variables and methods. It then also **implements** the herbivore, cuddly, and photogenic interfaces. In order to do so, our Panda class will need to override every abstract method defined in each one of those interfaces. The advantage to us as the programmer is that then we'll have more guarantees about what the Panda can do and how it can interact with other herbivores, etc.

1. Watch the three videos on interfaces and take good notes
2. Now, complete the following 6 challenges, paying close attention to the creation and implementation of interfaces:
    a. JAVA TASK 65: Make a program that contains a "Student" class with at least 4 instance variables, all setters & getters, as well as constructor and a toString() method. (This class will implement the rest of the tasks)
    b. JAVA TASK 66: Many students are also great musicians! Make an interface called "Musician" and give it 3 abstract methods. Then, go back to your Student class and properly implement the Musician interface.
    c. JAVA TASK 67: Many students are also excellent athletes! Make an interface called "Athlete" and give it 3 abstract methods. Then, go back to your Student class and properly implement the Athlete interface.
    d. JAVA TASK 68: Many students are hard-working mathematicicans! Make an interface called "Mathematician" and give it 4 abstract methods. Then, go back to your Student class and properly implement the Mathematician interface.
    e. JAVA TASK 69: Many students are also very talented artists! Make an interface called "Artist" and give it 5 abstract methods. Then, go back to your Student class and properly implement the Artist interface.
    f. JAVA TASK 70: Many students also work jobs outside of school! Make an interface called "Employee" and give it ***6 abstract methods***. Then, go back to your Student class and properly implement the Employee interface.

Part 2: Tasks	*10-7 points*	*6-4 points*	*3-0 points*
Interface Notes	+ Watch the three presentations on Interfaces + Take a full page of notes on the ideas, including details on the keyword "implements"	- Less than a full page of interface notes - No brainstorming present	- Very brief or no notes in your notebook
	20-10 points	*9-5 points*	*4-0 points*
Java Tasks 65-70	+ You completed all 6 Java Tasks from this section	- You did not complete all 6 tasks	- You did not complete any tasks
Checkoff from Benshoof	+ Mr. Benshoof got to see your Java programs run successfully	N/A	N/A

(20 pts) Approx. 3 days

At this point in our AP Computer Science class, we know pretty much 95% of the content we need to do some really big, awesome things with our own computer programs! We started with control structures (if, while, for), and added data types (int, double, arrays, Strings, ArrayLists), covered object oriented design (classes, objects, methods, variables), and now have talked about the details of class structure and organization (inheritance & polymorphism).

With all those ideas in mind, it's time to build the framework for *the most awesomest video game ever!* Think about huge video games that you've played or seen: World of Warcraft, Minecraft, Halo, Skyrim, Call of Duty, Fortnite, Overwatch. Each of those video games has a massive program with a huge class hierarchy behind it. In this last part of the unit, you get to design your own video game class hierarchy and structure to illustrate your knowledge of inheritance and polymorphism

1. Use your engineering notebook to brainstorm and plan out the different games you could create, and how you want to organize your class structure once you've chosen a game! Do some brainstorming and planning before you start programming!
2. VIDEO GAME DESIGN: Make a new project in BlueJ (so you've got a nice blank work space). Create your class structure using proper inheritance and polymorphism to illustrate how everything would be related. Make sure that your design fits the following criteria:
    a. Your hierarchy should have at least 4 levels
    b. Your hierarchy should have at least 15 classes in it
    c. There should be a runner class separate from the main hierarchy
    d. You should include at least 2 interfaces
    e. You should include at least 2 abstract classes each with at least 1 abstract method
    f. Your entire project should be able to compile without error
    g. Your classes should all have at least 1 method in them that has a name that implies relevant functionality, but does not need to actually ***do*** anything

3. *Log 7: Inheritance & Polymorphism* – To wrap things up, take some time to write a full-page response to the ideas of inheritance and polymorphism. What do you think is coolest about those ideas? What's most confusing? What do the keywords ***extends***, ***super***, ***implements***, and ***abstract*** do for a program? How can you see inheritance and/or polymorphism work in other programs? What do YOU think the difference is between an abstract class and an interface?

Part 3: Tasks	*5 points*	*4-3 points*	*2-1-0 points*
Video Game Design Brainstorm	+ You made a plan in your notebook for how to organize your game structure + Your brainstorming is recorded	- You wrote less than a page - Your notes do not outline a coherent plan	- There is no plan for your program
	15-10 points	*9-5 points*	*4-0 points*
Video Game Design Class Framework	+ You completed your entire game design framework + Your framework meets all criteria listed above	- Your game design framework is mostly complete	- You framework is incomplete
	10-8 points	*7-4 points*	*3-0 points*
Log 7: *Inheritance & Polymorphism*	+ You wrote a complete page in your engineering notebook	- You wrote less than a full page	- You wrote less than half a page