Computer Science AP
Inheritance Section

Questions for Video - Introduction to Inheritance

What does inheritance allow a programmer to do?

How does the keyword 'extends' , as in public class Student *extends* Person state about a class?

Name one example in the video that demonstrates how inheritance can save the programmer a lot of time and effort.

In the InheritanceExamples project, look at the Bugs package.
How would you determine which class or classes are making use of inheritance? Which class or classes are using inheriance in this package?

In the InheritanceExamples project, look at the LCDScreen package.
Which classes build off of which classes?

How does inheritance allow you to avoid copy and pasting code that you have already written?

Questions for Videos - Inheritance Fields and Inheritance Methods

Describe the role that access modifiers like public, private, and protected play in inheritance.

In the InheritanceExamples project, look at the StudentTeacher01 package.
Assume you are coding inside the StudentTeacher01 package.
Explain why or why not the following would be allowed:

A private variable in the Person class can be used in the Student class.
A protected variable in the Person class can be used in the Student class.
A protected variable in the Person class can be used in the ExchangeStudent class.
A public variable in the Person class can be used in the Student class.
A public variable in the Person class can be used in the ExchangeStudent class.
A private variable in the Student class can be used in the Person class.

A student created the following class and it compiles without any errors

```
public class Dog extends Animal {

   public Dog() { //code not shown }

   public void showInfo() {
     System.out.println(city + ": " + tagId);
   }
}
```

How might you explain the code compiling without error when the Dog class does not appear to have a variable named tagId ?

What level of access could/would the tagId variable have in order for this to compile?

By the definition in the Java docs, a variable declared with an access level of _____ is not inherited by subclasses.

In the InheritanceExamples project, take a look at the Shapes package.

Which variable/variables from the Shape class can the Circle class not directly access?

The Circle class is able to modify the value of the x and y position because the Shape class has _____ .

The programmer of the Shape class decides to go back and make the *sides* variable have *protected* access. No other code in the project is changed. Does this cause problems with the Circle and Parallelogram class? Explain why or why not.

The programmer of the Shape class decides to go back and make the *sides* variable have *private* access. No other code in the project is changed. Does this cause problems with the Circle and Parallelogram class? Explain why or why not.

A runner class from a completely different project uses the following code:

```
public class RunnerFromADifferentMother {

  public static void main(String[] args) {
      Circle C = new Circle(10);
      C.shrink();
      ColoredCircle CC = new ColoredCircle(8, Color.BLUE);
      CC.shrink();
  }
}
```

Will the code compile without errors? Explain.

Do you think making the shrink method a private method was a good choice? Explain.

Questions for Video - Inheritance Overriding Methods

In the InheritanceExamples project, take a look at the Shapes package.
Which methods would be considered overrided in the Parallelogram class?

In the InheritanceExamples project, take a look at the Bugs package.
Which methods would be considered overrided in the Parallelogram class?

Why might you override a method from your super class (a class you extend)?

In the StudentTeacher01 package, why did the programmer decide to make the ExchangeStudent override the getAverageMark method of the Student class?

This coding task will be handed in as a file called RandomBug.java when you are done:

In the InheritanceExamples project, take a look at the Bugs package.
Read the entire task before you start!
Create a new class called RandomBug that extends the Bug class.
A random bug will pick a new direction every couple of steps.
*Don't worry if the bug wanders out the drawable area of the frame.
To accomplish this you will have to override the act() method of the Bug class in RandomBug.
The constructor for RandomBug can basically be a copy and paste of the constructor for the BoxBug (but call it RandomBug() !) class.
To test your Bug, go to the BugFrame class and find the lines of code where bugs are added to the simulation. Add in one of your RandomBugs and run the program to see if it works.

Questions for Video - Inheritance Constructors and Super

In the InheritanceExamples project, take a look at the AnB package.

When writing the constructor for ClassB, you could just write the following without even using *super()*. Why not?

```
public ClassB() {
   System.out.println("Constructor ClassB");
}
```

When writing the constructor for ClassX, you MUST make a call using *super*. Why?

Under what circumstances will a subclass have to use *super* in their constructor?

If you are going to use the word *super* in a constructor, what rule/s apply to it's use?

A student makes the following comment: "When instantiating an instance of ClassB with the line *ClassB temp = new ClassB();* the constructor of ClassA will be called whether or not you inlclude the line *super()* in the constructor of ClassB."
Is this true or false? Explain.

In the InheritanceExamples project, take a look at the Bugs package.
You are asked to start writing a new class called TurnBug that will extend class Bug.
A TurnBug will turn right by a specific angle after a set number of steps.
Start the 'top part' of this class by filling in the following with code that would be appropriate.

```
public class TurnBug extends Bug {

  ?  new fields  ?

  public TurnBug(      ? constructor parameters ?       ) {

     ? constructor code ?

  }

  public void act() {
    age++;
    if (age==stepsBetween) {
      turn(turnAngle);
      age=0;
  }

}
```

The constructor should accept parameters that state the turn angle of the bug and a number that lets us know how many steps to wait between each turn. Of course the constructor should also accept any additional parameters to allow it to be compatable with the constructor of it's super class Bug. Replace the ? sections with your own code. You can code this in Netbeans to check if you get 'red lines' indicating problems with your implementation.

Questions for Video - Inheritance Overrided Methods in Superclass

Consider the following classes and code:

```
public class Ball {
  public int speed;

  public Ball() {
    speed=0;
    startRolling();
  }

  public void startRolling() {
    speed=5;
  }

  public void speedUp() {
    speed+=5;
  }

  public void goTurbo() {
    speedUp();
    speedUp();
    speedUp();
  }

}
```

```
public class BigBall extends Ball {

  public BigBall() {
     super();
  }

  public void startRolling() {
     speed=1;
  }

  public void speedUp() {
     speed+=1;
  }


}
```

In a runner program...
Line 1: Ball ball1 = new Ball();
Line 2: BigBall ball2 = new BigBall();
Line 3: ball2.goTurbo();

Below are a few quesitons regarding overriden methods in subclasses. Before you answer them, summarize the 'rule' regarding overrided methods that you will use to help you determine your answers.

When line 1 finishes executing, what is the speed of ball1 ?

When line 2 finishes executing, what is the speed of ball2 ?

When line 3 is executed, will the goTurbo() method be using the code for speedUp() from the Ball class or the BigBall class?

When line 3 finishes executing, what will be the speed of ball2 ?