Unit 9: Inheritance

**Topic 1: Creating Superclasses and Subclasses**

|  |  |
| --- | --- |
| **Name:** | **Wei Xiang Zheng** |

|  |
| --- |
| Vehicles, Cars & Bicycles |
| 1. Start a new IntelliJ project and download the following classes from today's demo:  * [Vehicle](https://drive.google.com/file/d/1gusiJEAvT5laXbIYDAFtM6xSDarIKQMm/view?usp=share_link) * [Car](https://drive.google.com/file/d/1p7of8D-74-Cxz7q6r7h2-ysf947_uEBB/view?usp=share_link) * [VehicleRunner](https://drive.google.com/file/d/16_2Srh-kWPcwLGc_yLCP8Lur1EuJXbBD/view?usp=share_link)  1. Similar to the Car subclass, write a new Bicycle subclass that **inherits** from (extends) the Vehicle class and has the following additional behavior:  * Bicycles can ring bells (i.e. a ringBell() method) -- for this method, simply add a print statement that prints a sentence to indicate the method has been called.   And the following additional attribute:   * Bicycles have a “gear count” (i.e. a gearCount instance variable that is an int)   + *Provide this as an attribute* ***and*** *provide a getter method for it*   Don't forget to call super in the Bicycle constructor!   1. Then, add some test code to the VehicleRunner class to create a Bicycle object and call **ALL** valid methods on that object (*hint:* in addition to the constructor, there should be **seven** valid methods you can call).   **Copy/paste your code below:** |
| // Bicycle class:  public class Bicycle extends Vehicle {  int gearCount;   public Bicycle(String name, int wheels, int gearCount) {  super(name, wheels);  this.gearCount = gearCount;  }  public void ringBell(){  System.*out*.println("ring! ring! ring!");  }   public int getGearCount(){  return gearCount;  } } |
| // VehicleRunner code that you added to call **ALL** valid Bicycle methods:  Bicycle myBicycle = new Bicycle("Some Bicycle", 2, 2); System.*out*.println(myBicycle.getName()); System.*out*.println(myBicycle.getWheels()); System.*out*.println(myBicycle.getGearCount()); myBicycle.move(30);  myBicycle.turn(-90);  myBicycle.brake(0.50); |

###### [Sample Bicycle class](#_iptj863a6092)

###### [Sample runner code](#_97vpq1avnbre)

|  |  |
| --- | --- |
| The Vehicle-Car-Bicycle Hierarchy | |
| Add your Bicycle subclass to the **UML class diagram** for the inheritance hierarchy:   |  | | --- | | Vehicle | | name  wheels | | getName()  getWheels()  move(int distance)  turn(int degreesToTurn)  brake(double brakePercent) |      |  |  |  | | --- | --- | --- | | Car |  | Bicycle | | fabric |  | gearCount | | getFabric()  honk()  lockDoors() |  | getGearCount() |  [Check](#_we9usluo6g1u) | |
| **Is each of these statements true or false?**   1. A Bicycle is a Vehicle 2. A Vehicle is a Bicycle 3. Bicycle inherits from Vehicle 4. Bicycle is a subclass (child class) of Vehicle 5. Bicycle extends Vehicle 6. Vehicle is the superclass (parent class) of both Bicycle and Car | a. T  b. F  c. T  d. T  e. T  f. T [check](#_8e3yu364k49g) |

|  |  |
| --- | --- |
| **PREDICT:**  Which of these lines of code, if added to the bottom of all test code in VehicleRunner, will cause compiler errors? In other words, which are **NOT** valid method calls?  myVehicle.ringBell(); //line 1  System.out.println(myVehicle.getGearCount()); //line 2  myCar.ringBell(); //line 3  myBike.honk(); //line 4  myBike.lockDoors(); //line 5 | |
| What line(s) do you think will cause problems? | All 5 |
| **TEST your prediction!**  Copy/paste the code above into IntelliJ and see what happens. Does it let you compile? Which line(s) are actually problematic? | It does not compile, all lines are problematic |

###### [check](#_u27p2fskuls1)

|  |
| --- |
| **Design your *own* Vehicle subclass!**  Create a *third* class that extends Vehicle. Think of one that makes sense in terms of what already exists in Vehicle. Add at least one new attribute and one new method to your new subclass.  Test it by creating an object of your subclass and calling **all** valid methods in your runner class: |
| // your new subclass  public class Motorcycle extends Vehicle{  int speed;  boolean lightsOn;   public Motorcycle(String name, int wheels, int speed) {  super(name, wheels);  this.speed = speed;  lightsOn = false;  }  public int getSpeed(){  return getSpeed();  }  public boolean switchLights(){  return !lightsOn;  } } |
| // VehicleRunner code that you added to call **ALL** valid methods for your subclass  Motorcycle myMotorcycle = new Motorcycle("Some motorcycle", 2, 202); System.*out*.println(myMotorcycle.getName()); System.*out*.println(myMotorcycle.getWheels()); System.*out*.println(myMotorcycle.switchLights()); System.*out*.println(myMotorcycle.getSpeed()); myMotorcycle.move(30); myMotorcycle.turn(-90); myMotorcycle.brake(0.50); |

|  |  |
| --- | --- |
| **Compare with your partner!**  What subclass did they create?  What attributes/methods did they add to that subclass? | They created an airplane subclass with attributes number of bombs, methods dropBomb() and getter to get number of bombs. |

**Done!**

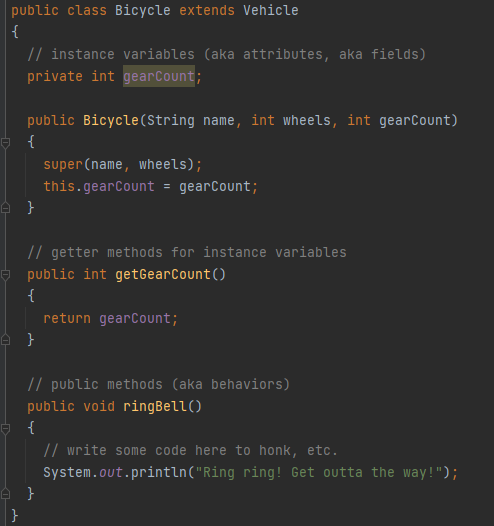
Submit in Google Classroom:

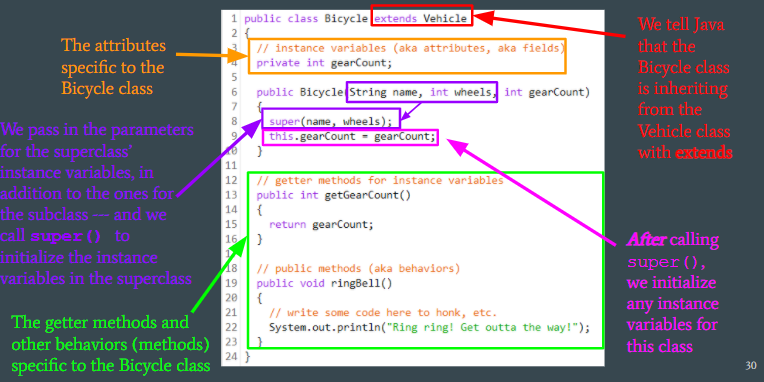


###### 

### Sample ([back](#_8erbtyz6u70k))

Don't forget the **extends** keyword in the class header!





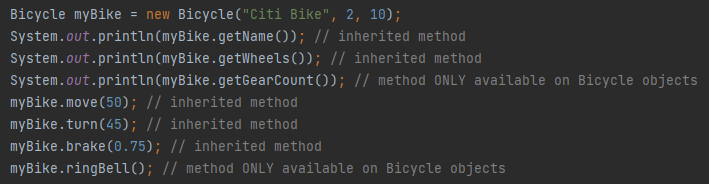
### Sample ([back](#_7crg2895x9ha))

On a Bicycle object, you can call all **five inherited**methods from the Vehicle superclass:

* getName()
* getWheels()
* move(int)
* turn(int)
* brake(double)

You can also call the **two Bicycle-only methods** defined in the Bicycle subclass:

* getGearCount()
* ringBell()

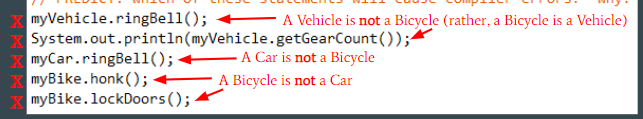


### Check ([back](#_bbhue78k2sdh))

|  |  |
| --- | --- |
| **Is each of these statements true or false?**   1. A Bicycle is a Vehicle 2. A Vehicle is a Bicycle 3. Bicycle inherits from Vehicle 4. Bicycle is a subclass (child class) of Vehicle 5. Bicycle extends Vehicle 6. Vehicle is the superclass (parent class) of both Bicycle and Car | a. True  b. **FALSE**: the "is-a" relationship only works up the chain; a subclass "is a" class of type superclass, but a superclass is not of type subclass.  c. True  d. True  e. True  f. True |

### Check ([back](#_pwa8dflltoqz))

|  |  |
| --- | --- |
| **PREDICT:**  Which of these lines of code, if added to the bottom of all test code in VehicleRunner, will cause compiler errors? In other words, which are **NOT** valid method calls?  myVehicle.ringBell(); //line 1  System.out.println(myVehicle.getGearCount()); //line 2  myCar.ringBell(); //line 3  myBike.honk(); //line 4  myBike.lockDoors(); //line 5 | |
| What line(s) do you think will cause problems? |  |
| **TEST your prediction!**  Copy/paste the code above into IntelliJ and see what happens. Does it let you compile? Which line(s) are actually problematic? | **ALL FIVE METHOD CALLS CAUSE COMPILER ERRORS BECAUSE NONE of them are valid method calls!**  You should notice that IntelliJ makes them all **RED**:    Hovering over any of them reveal the same problem:  *"Cannot resolve method"*  This means the method is not a valid method for that object type. |



### Check ([back](#_uqn65z3p6ehq))

|  |
| --- |
| The Vehicle-Car-Bicycle Hierarchy |
| Add your Bicycle subclass to the **UML class diagram** for the inheritance hierarchy:   |  | | --- | | Vehicle | | name  wheels | | getName()  getWheels()  move(int distance)  turn(int degreesToTurn)  brake(double brakePercent) |      |  |  |  | | --- | --- | --- | | Car |  | **Bicycle** | | fabric |  | **gearCount** | | getFabric()  honk()  lockDoors() |  | **getGearCount()**  **ringBell()** | |