Java Polymorphism

Lab Setup

1. Create a new Java project in Eclipse
2. Add the provided classes `MountainBike`, `Bicycle`, `Car`, `SUV`, and `Sedan`.
3. Add a main class which we can use as a driver

Polymorphism

Subclasses can define their own unique behaviors and share some of the same functionality of their parent class.

```java
// SUV.java
public class SUV extends Car {
    public boolean active4WD;

    public SUV() {
        topSpeed = 0;
        MPG = 0;
        acceleration = 0;
        active4WD = false;
    }

    public SUV( int topSpeed, int MPG, int acceleration, boolean active4WD ){
        this.topSpeed = topSpeed;
        this.MPG = MPG;
        this.acceleration = acceleration;
        this.active4WD = active4WD;
    }
}
```

Figure 1 SUV class

Graded Submission Task 1

We can use the classes `Car`, `SUV`, and `Sedan` as an example.

```java
// Sedan.java
public class Sedan extends Car {
    enum Drivetrain_2WD {
        FWD, RWD;
    }

    public Drivetrain_2WD type;

    public Sedan() {
        topSpeed = 0;
        MPG = 0;
        acceleration = 0;
        type = Drivetrain_2WD.FWD;
    }

    public Sedan( int topSpeed, int MPG, int acceleration, Drivetrain_2WD type){
        this.topSpeed = topSpeed;
        this.MPG = MPG;
        this.acceleration = acceleration;
        this.type = type;
    }

}
```

Figure 2 Sedan class

`Car` has the fields `topSpeed`, `MPG`, and `acceleration`. `Car` also defines a method `printInformation()` which prints out the fields of the class.

`SUV` and `Sedan` extend the `Car` class with the new information on its drivetrain.

For both `SUV` and `Sedan` override the `printInformation()` method so that in addition to printing the fields topSpeed, mpg, and acceleration it also prints the new field for each class.

```java
Car[] cars = new Car[3];

cars[0] = new Car();
cars[1] = new Sedan(120, 60, 5, Sedan.Drivetrain_2WD.RWD);
cars[2] = new SUV(100, 50, 4, true);

for (Car c : cars){
    c.printInformation();
}
```

Figure 3 Calling printInformation() for Car, SUV, and Sedan

Add the code shown in Figure 3 to main. You should see that when each call to the `printInformation()` method, the output is different depending on the type of the car. This is polymorphism.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

Using `super`

The keyword `super` can be used to access overriden methods from the parent class.

Inside the `Bicycle` class define a method `printFields()`, which prints out the cadence, speed, and gear.

Now in `MountainBike`, we can override the `printFields()` method to also print the `MountainBike` field `suspension` and use `super` to avoid re-implementing printing out the cadence, speed, and gear.

```java
/// MountainBike.java
@Override
public void printFields() {
    super.printFields();
    System.out.println("Suspension: " + this.suspension);
}
```

Figure 4 Using super in MountainBike method printFields()

We can also use `super` to call the constructor of the super class.

```
/// MountainBike.java
public MountainBike(){
    super();
    this.suspension = 0.0;
}

public MountainBike(int cadence, int speed, int gear, double suspension) {
    super(cadence, speed, gear);
    this.suspension = suspension;
}
```

Figure 5 Calling parent constructor using super

The constructors of `MountainBike` initialize the values of suspension, and the three inherited fields from `Bicycle`. The constructor of `Bicycle` already has that functionality, so we can use that instead of re-implementing it.

So to use the default constructor of the super class we write `super()` and to use any other constructor we write `super()` and within the parentheses we include the list of arguments for the super class's constructor.

Graded Submission Task 2

Modify `SUV` and `Sedan` from Task 1 to make use of the keyword `super`. The constructor and `printInformation` can be written using `super`.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

Drone application

We will eventually want to expand our simulation to include different types of entities. We can start developing the framework for this now. You may create a new project for this implementation or work in a previous drone project implementation.

Graded Submission Task 3

We mentioned drones are entities in the simulation. We will extend our definition of entities to include all objects in the simulation including the depots and the drone controller.

Define the base class Entity. The Entity class should have methods and fields shared by the drones, depots, and drone controller.

Define a base class for different types of depots called Depot. This class should extend Entity.

Change the class definitions of the drones, depots, and drone controller so that they derive from Entity. DroneDepot and RepairDepot will derive Entity indirectly by extending Depot.

Where possible, avoid repeated code by using `super`.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 3. You will submit this copy at the end of this lab.

Lab wrap-up

In this lab we discussed polymorphism. Subclasses can define their own unique behaviors and still share some of the same functionality of the parent class. We illustrated this using the Car base class and some classes which extend Car. All Cars have a method printInformation() the implementation of which depends on the type of Car.

The keyword super is used to access overridden methods from the parent class. It is especially useful in the constructor of a child class. You can use super to call a constructor of the base class which initializes inherited fields decreasing code repetition.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder "Lab13_firstName_lastName".
- Submit the zipped file to the respective Canvas link.

Rubrics		
Task 1 (points allotted) - Compilation (2) - SUV and Sedan override printInformation (4) - printInformation outputs the new fields for SUV and Sedan (4)	Task 2 (points allotted) - Compilation (2) - SUV and Sedan constructors use super to call Car constructor (4) - SUV and Sedan printInformation use super to call Car printInformation (4)	Task 3 (points allotted) - Compilation (2) - Entity base class defined (2) - Depot base class defined, extends Entity (2) - Drone, DroneController extend Entity; DroneDepot, RepairDepot extends Depot (4)