Java Inheritance

Objective(s)

- Learn about the concept of inheritance in object-oriented programming
- Learn about method overriding, method hiding, final methods, and final classes

Lab setup

1. Create a new Java project in Eclipse, call it "Java Inheritance"
2. Add the `Bicycle` class and `Car` class to the project
3. Add a main class to the project which we can use as a driver

Inheritance

Inheritance is a concept in object-oriented programming which allows for a class to inherit properties of another class.

A child (derived) class inherits the properties of its parent (base or super) class.

```
public class MountainBike extends Bicycle {
    double suspension;

    public MountainBike() {
        speed = 0;
        gear = 0;
        suspension = 0.0;
    }

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

Figure 1 MountainBike class

Graded Submission Task 1:

1. Let's define a child of the `Bicycle` class, `MountainBike`. We do this with the keyword `extends`. We will add one new field, `double suspension;`

`MountainBike` will inherit all the `public` and `protected` fields and methods of `Bicycle` except the constructor. Instead, we define new constructors for MountainBike which initialize both new and inherited fields.

```
public double getSuspension() {
    return suspension;
}

public void setSuspension(double suspension) {
    this.suspension = suspension;
}
```

Figure 2 Getter and Setter for suspension

2. We do not need to redefine getters and setters for the `speed` and `gear` fields as they are inherited by `MountainBike`, but we do need to define them for `suspension`.

** 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.

Method overriding

Subclasses can override any methods inherited from their base class which are not declared `final`.

```
/// Bicycle.java
 public void printType() {
     System.out.println("Bicycle");
 }
```

Figure 3 printType() method in Bicycle base class

Graded Submission Task 2:

1. Add a public method to `Bicycle` called `printType()`. For the base `Bicycle`, this method will print "Bicycle", as shown in Figure 3.

```
/// MountainBike.java
@Override
public void printType() {
    System.out.println("MountainBike");
}
```

Figure 4 printType() method overridden in MountainBike class

2. Override `printType()` in `MountainBike` so that it prints "MountainBike" instead as shown in Figure 4.

   The `@Override` is an annotation. This tells the compiler that you intend for this method to override a method in the super class. If for some reason no such method exists, the compiler will generate an error.

** 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.

Final methods and classes

The keyword `final` can be used to indicate methods which cannot be overridden by a subclass.

1. As an example, label the `Bicycle` method `printType()` to be `final`

```
/// Bicycle.java
final public void printType() {
    System.out.println("Bicycle");
}
```

Figure 5 printType() as a final method.

   The compiler should now be generating an error about the attempt to override `printType()` in `MountainBike`. You can use `final` for implementations which should not be changed and are critical to the consistent state of an object.

   Remove `final` from the `Bicycle` method `printType()`.

2. Entire classes can also be declared as `final`. This means that the class cannot be subclassed by any other classes.

   Add the `final` keyword to the `Bicycle` class as an example.

```
/// Bicycle.java
final public class Bicycle {
    //...
}
```

Figure 6 Bicycle class labeled final.

   The compiler should now be generating errors because `MountainBike` is attempting to extend `Bicycle`.

Remove `final` from the `Bicycle` class and add it to `MountainBike`.

```
/// MountainBike.java
final public class MountainBike extends Bicycle {
    //...
}
```

Figure 7 MountainBike class labeled final.

The error should be gone now. If you were to try to extend MountainBike you would again get an error.

Lab wrap-up

In this lab we discussed inheritance. Inheritance is when a child (or derived) class inherits the properties of its parent (base or super) class. Inheritance is implemented in Java using the extends keyword. When a child class extends its parent, it will inherit all public and protected methods and fields.

Child classes can override any methods it inherits from its parent. This means it provides its own definition of the method. You can use the annotation `@Override` to ensure you are overriding a method. The compiler will throw an error the method you annotate is not overriding another method.

The keyword final is used to indicate methods which cannot be overridden. Final classes cannot be subclassed/extended.

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 "Lab12_firstName_lastName"
- Submit the zipped file to the respective Canvas link.

Rubrics	
Task 1 (points allotted): - Compilation (2) - MountainBike field suspension (2) - Getter and setter for suspension (2) - Constructor initializes inherited fields and suspension (2) - MountainBike extends Bicycle (2)	Task 2 (points allotted) - Compilation (2) - Bicycle method printType (2) - @Override used (2) - printType overridden in MountainBike (4)

Appendix