Java Classes

Classes

A Java program is a set of classes. One of these classes will hold the entry point of the program called the main method. This class is called the main class.

1. Create a new Java project in Eclipse and name it “JavaClasses”. Add a new Java class to the project for the Bicycle class. You can do this from File > New > Class, the class wizard, or by right-clicking the project in the Package Explorer.

A Java class is made up of fields, constructors, and methods.

```
public class Bicycle {

    // Three fields
    public int cadence;
    public int gear;
    public int speed;

    // Constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        cadence = startCadence;
        gear = startGear;
        speed = startSpeed;
    }

    // Methods
    public void setCadence(int newValue){
        cadence = newValue;
    }

    public void setGear(int newValue){
        gear = newValue;
    }

    public void applyBreak(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }
}
```

Figure 1 Bicycle class

2. Add the code in Figure 1 to Bicycle.java. Be careful about copy-and-pasting from this document. Some of the formatted text may look similar, but they are not plain-text and will cause issues when the compiler tries to parse the file.

> A class such as this Bicycle class is a top-level class; meaning Bicycle is not derived from any other class nor is it nested within another class (we will look at examples of this in later labs).
>
> Bicycle is not a main class because it does not have a main method, and this would not be a complete Java program.

3. Check this fact by attempting to run the program. You will get an error asking you to define the main method in Bicycle.java. We won't be doing this because we do not want Bicycle.java to be the main class.

4. Add a new class to the project; call it "Driver". Driver will be our main class. When you create the class, you can choose to have Eclipse insert a method stub for the main method for you, otherwise you will need to type it out yourself.

> *You will see Driver used as the name of the main class often in these small lab examples. Drivers are small programs used to test the functionality of a class.

You can think of a class as a blueprint. Above, we defined a blueprint to create Bicycle objects. We will refer to the Bicycle class in these next sections.

Fields

Fields provide the state of the class. The state of a Bicycle object is defined by its cadence, gear, and speed. Fields are variables that hold the data. Each field should have a name and a data type (int, float, double, char, …., String, Integer, Float, Double, ..., Student, Book, ...)

Constructors

Constructors are used to initialize the state of an object. The Bicycle class provides one constructor, which is used to create a Bicycle object with startSpeed, startCadence, and startGear as initial values. A constructor which does not have any parameters is called a default constructor. It is used to initialize an object to a default state. For example, we might provide the following default constructor for Bicycle.

```
public Bicycle() {
    cadence = 0;
    gear = 0;
    speed = 0;
}
```

Figure 2 Bicycle default constructor

Methods

Methods implement the behavior of a class and its objects. Bicycle has 4 methods that define its behavior. When a Bicycle speeds up, its speed value is increased. When a Bicycle applies its break, its speed value is decreased. Consider, what might methods called **`gearUp()`** and **`gearDown()`** do?

Methods need the following information:

- Return type (void for no return type)
- Name
- Parentheses ( )
- A body between { } (must include a return statement if the return type is not void)

Between the parentheses, you can provide a parameter list.

Note: The comma-separated list provided between the ( ) in the method definition is called a parameter list, made up of parameters. When a method is called, the values/variables provided are called arguments.

Note: To call a method from outside a class you need an object of that class. Methods inside a class can comfortably call other methods within the class.

Creating and using objects from classes

We can declare an object of a class as follows:

```
Bicycle declaredBicycle;
```

We can create an object from a class definition using the `new` operator and calling a class's constructor.

```
Bicycle defaultBicycle = new Bicycle();
Bicycle bicycle = new Bicycle(2, 3, 4); /// Need a Bicycle object to ca
ll speedUp()
```

With an object of a class, we can call methods from that class and update its fields. Imagine we are using this Bicycle class in a simulation. We can create a Bicycle object and keep track of its state through its lifetime.

Graded Submission Task 1

5. Implement the Bicycle methods **`gearUp()`** and **`gearDown()`**. Above each method, comment what behavior the method represents, and how it is implemented.

6. Create two Bicycle objects in the main method. Print out the initial state of the two Bicycle objects i.e., print the fields of each.

7. For one of the Bicycle objects, call each method and print the new state of the object after each method call

** 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.

```
Initial states:
Bicycle 1
 Cadence = 0
 Gear = 0
 Speed = 0
Bicycle 2
 Cadence = 2
 Gear = 4
 Speed = 3

Calling methods for Bicycle 2
setCadence()
 New cadence value = 4
setGear()
 New gear value = 5
gearUp()
 New gear value = 6
gearDowr()
 New gear value = 5
speedUp()
 New speed value = 8
applyBreak()
 New speed value = 6
```

Figure 3 Example output for Task 1.

Drone application

We will expand on this example throughout the labs.

The Department wants to create a system for autonomous drone communication. You have been asked to develop a drone simulation before committing expensive resources toward the project. For simplicity, they suggest you develop the simulation in 2D space.

You are not sure how to immediately start developing the simulation, but you can at least start designing the classes. You decide to start with Drones and Drone Depots.

Hint: please note that, in this Drone application task we do not provide very specific requirements like the exact methods or types for fields. We wanted you to decide on your own what might be necessary or useful based on the description of the requirements/problem. Students can refer to the recorded lab-video or ask the TA during lab time to complete this Drone Task. The main objective of doing so is to give you freedom to explore and apply what you have learned. This Drone Task is a lower stakes opportunity (compared to assignments and projects) for you to try tackling a problem on your own so the TA can give feedback to you.

Drones

The main entities in this simulation are drones. Drones are identified by an ID. All drones keep a record of their current position, a point (x,y) in 2-dimensional space. They also have a top speed, effective signal range, payload limit, and message.

- Speed is measured in meters per second (m/s)
- Effective signal range is measured in meters

- Payload weights are measured in centigrams (cg)
- For our purposes, a message will be a word/collection of words

Drones shall be able to move any distance in 2-dimensional space, but they can only move on one axis at a time. (Imagine a rook from chess.) They shall also be able to carry a payload up to their limits. They shall load payloads of given weights. If the weight of a payload exceeds a drone's payload limit, it cannot be loaded.

Consider
Suppose we want to be able to load more than one item on a drone. What additional information would we need that is not specified?

Drone Depots

Drones originate from drone depots. Drone depots keep a record of their current position, a point (x,y) in 2-dimensional space.

Drone depots create drones with their ID, top speed, effective signal range, payload limit, and message then they release them into the system at the same coordinates as the depot.

Consider
What data types can be used to represent the data required for both drones and drone depots? Can they all be represented with the primitive data types? What methods should you write to meet the requirements?

Graded Submission Task 2

1. Start by creating a new Java project in Eclipse called “DroneSimulation”
2. Add the DroneDepot class to the project.
3. Add the Drone class to the project.
4. Implement the Java classes, Drone and DroneDepot
5. Write a driver (main class) to test the functionality of your implementation.
6. Specifically, your driver should:
   - Create two Drones at two different coordinates. Print their original locations, then move them around using your implemented methods and print their final locations.
   - Create one Drone Depot at any coordinate. Create a Drone using the Drone Depot and print the drone's starting location. Move the Drone around using your implemented methods, then print its final location.
   - Choose one of your created Drones and demonstrate loading payloads onto the drone using your implemented methods (including attempting to load too much weight)

** 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.

Lab wrap-up

In this lab we discussed classes in Java. We described the class definitions as "blueprints" for objects. We use fields to describe the state of a class. To instantiate and initialize an object of a class we must define a constructor for the class. There are two types of constructors. The default constructor does not take any arguments. You define the default values for the fields. Non-default constructors can take arguments whose values you can assign to the fields. Methods represent the behavior of a class and its objects. To define a method, we need the method's return type, name, a list of parameters (which can be empty) surrounded by parentheses. The body of the method is surrounded by curly braces and defines what the method does.

Then, we looked at how to use classes and objects of those classes using the Bicycle class and our Drone application example.

If you are finished with both tasks 1 and 2 you can zip them together into a zip file and submit them at this time. You can find more information about these submissions below.

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 "Lab2_firstName_lastName".
- Submit the zipped file to the respective Canvas link.

Grading Rubrics	
Task 1 (points allotted) - Compilation (2) - Create 2 Bicycle objects (2) - Print the fields of each object after instantiation (2) - Call the member variables for 1 of the objects (2) - Print the new values of the fields for the same object (2)	Task 2 (points allotted) - Compilation (4) - Appropriate choices for field data types (3) - Driver performs as expected (See 6.) (3)