Nested, Local, Anonymous

Lab Setup

1. Create a new Java project called JavaNestedClasses
2. Add the Bicycle class to the project.
3. Add a main class we can use as a driver.

Nested Class

A nested class is a class which is defined within another class. We can use nested classes to group classes which are only used in one place and increase encapsulation.

Static nested classes

Static nested classes can be accessed from outside the class using the enclosing class' name. They can also be directly imported from a package, again using the enclosing class' name. Figure 1 shows the Bicycle class with a static nested class within it.

```
public class Bicycle {

    // fields..
    static class Tyre {
        double pressure;
        double size;

        Tyre() {
            pressure = 1.0;
            size = 1.0;
        }

        void increasePressure(double increment) {
            pressure += increment;
        }

        void decreasePressure(double decrement) {
            pressure -= decrement;
        }
    }
}
```

Figure 1 Bicycle class with static Tyre class.

Tyre is nested within the Bicycle class, but a Tyre object can exist without a Bicycle object existing.

```
public static void main(String[] args) {
    Bicycle.Tyre tyre = new Bicycle.Tyre();

    System.out.println("Pressure: " + tyre.pressure);
}
```

Figure 2 Creating a Tyre object from static nested class.

Graded Submission Task 1:

1. Add the static nested class in Figure 1 to the Bicycle class.
2. Create a Tyre object in the driver then print the initial state of the Tyre object as shown in Figure 2.
3. Call the Tyre methods and print the state of the object after each 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.

Inner Classes

Non-static nested classes are called inner classes. They can only exist within an instance of the outer class.

1. Modify Tyre to be non-static.

```
public class Bicycle {

    //...
    class Tyre {
        double pressure;
        double size;

        Tyre() {
            pressure = 1.0;
            size = 1.0;
        }

        void increasePressure(double increment){
            pressure += increment;
        }

        void decreasePressure(double decrement){
            pressure -= decrement;
        }
    }
}
```

Figure 3 Bicycle class with non-static Tyre class.

You will notice that the instantiation line is now displaying an error.

```
No enclosing instance of type Bicycle is accessible. Must qualify the
allocation with an enclosing instance of type Bicycle (e.g. x.new A() where x
is an instance of Bicycle).
```

Let's fix that. Now we need a Bicycle object to create a Tyre object (shown in Figure 4).

The rest of the driver code should work as previously written.

```
/// No longer works
/// Bicycle.Tyre tyre = new Bicycle.Tyre();

// Create an object of the outer class
Bicycle bikeObject = new Bicycle();

// Use object to create an object of the inner class
Bicycle.Tyre tyre = bikeObject.new Tyre();
```

Figure 4 Creating a Tyre object from non-static inner class.

Under the umbrella of inner classes are two special kinds of inner classes: local classes and anonymous classes.

Local Class

Local classes are classes which are defined in a block. A block is a group of statements between balanced braces. For example, the body of a method is a block.

In Figure 5, we have the `Invoice` class. An invoice has an ID and an associated phone number. An invoice is instantiated without a phone number. The phone number is added later and is validated using the local class `PhoneNumber.`

```
class Invoice {

    private int ID;
    private String phoneNo;

    Invoice(int ID){
        this.ID = ID;
        this.phoneNo = null;
    }
    // Example from Oracle Java tutorial
    // https://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html
    private void validatePhoneNo(String phoneNo) throws Exception{

        class PhoneNumber{

            String formattedPhoneNumber = null;

            PhoneNumber(String phoneNumber){
                // numberLength = 7;
                String currentNumber = phoneNumber.replaceAll(
                  regularExpression, "");
                if (currentNumber.length() == numberLength)
                    formattedPhoneNumber = currentNumber;
                else
                    formattedPhoneNumber = null;
            }

            public String getNumber() {
                return formattedPhoneNumber;
            }
        }

        PhoneNumber number = new PhoneNumber(phoneNo);

        if (number.getNumber() == null) {
            throw new NumberFormatException();
        }

        this.phoneNo = number.getNumber();
}
```

Figure 5 Invoice class using PhoneNumber local class.

Anonymous Class

Instead of definitions, anonymous classes are expressions. An example we have already seen is `Iterator<T>`, which is used in classes which implement `Iterable`.

An anonymous class expression consists of:

- `new` operator
- Name of interface to implement or class to extend
- Parentheses for the arguments of a constructor.
- A class declaration body.

If we wanted to rewrite this to use an anonymous class, we could; as shown in Figure 6.

```
class Invoice {

    private int ID;
    private String phoneNo;

    interface PhoneNumber {
        public void formatNumber();
        public String getNumber();
    }

    Invoice(int ID){
        this.ID = ID;
        this.phoneNo = null;
    }

    // Example adapted to anonymous class from Oracle Java tutorial
    // https://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html
    private void validatePhoneNo(String phoneNo) throws Exception{

        PhoneNumber number = new PhoneNumber() {
                String formattedPhoneNumber = null;

                public void formatNumber() {
                // numberLength = 7;
                    String currentNumber = phoneNo.replaceAll(regularExpression, "");
                    if (currentNumber.length() == numberLength)
                        formattedPhoneNumber = currentNumber;
                    else
                        formattedPhoneNumber = null;
                }
                public String getNumber() {
                    return formattedPhoneNumber;
                }
        };

        number.formatNumber();
        if (number.getNumber() == null) {
            throw new NumberFormatException
        }
        this.phoneNo = number.getNumber();
}
```

Figure 6 Invoice class using PhoneNumber anonymous class.

Drone Application

We will apply nested classes to the Drone application by introducing repair depots and repair drones.

Graded Submission Task 2:

Define a new class, `RepairDepot`. Within the `RepairDepot` class, nest a `RepairDrone` class. Repair drones cannot exist within the system without an existing repair depot.

`RepairDepot` has an ID and the same (x,y) position information as `DroneDepot`. It should have the following methods:

- **`Repair(Drone d)`**
  Triple the `topSpeed` of Drone d
- **`DeployDrone()`**
  Release a `RepairDrone` into the system.

`RepairDrone` has an ID and (x,y) position. It should have the following methods:

- **`Repair(Drone d)`**
  Double the `topSpeed` of d
- **`moveX(int dx)` and `moveY(int dy)`**
  increment/decrement x or y by the delta

Write a driver to test the functionality of the `RepairDepot` and `RepairDrone`. This driver should create a `Drone` then subject it to the repairs of the `RepairDepot` and `RepairDrone`. Test the deployment of a repair drone, and the repair drone's movement as well.

** 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 nested classes. Classes can be nested within other classes to increase encapsulation. We discussed static nested classes first. These are nested classes which are labeled static. Like static methods and fields, static nested classes do not need an object of the enclosing class. They can exist (be instantiated) without an instance of the enclosing class.

Nested classes which are not labeled static are called inner classes. Inner classes require an instance of the enclosing class to be instantiated. The syntax for instantiating is shown in Figure 4.

There are two special types of inner classes: local classes and anonymous classes.

Local classes are those defined within a block, (a pair of curly braces). We gave the example of a local class defined within the body of a method.

Anonymous classes are not defined, instead they are expressions. The instantiation of an anonymous class includes a definition of the class in the expression.

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 “Lab8_firstName_lastName”.
- Submit the zipped file to the respective Canvas link.

Rubrics	
Task 1 (allotted points) - Compilation (2) - Complete static nested class Tyre (4) - Tyre object created (2) - Call each Tyre method (2)	Task 2 (allotted points) - Compilation (2) - RepairDepot class and methods (3) - RepairDrone inner class and methods (3) - Driver tests RepairDepot and RepairDrone (2)