Java Packages

Lab setup

- Create a new Java project called "JavaPackages"
- Add the provided `Rideable` and `Driveable` interfaces to the project
- Add the provided `Bicycle`, `Motorcycle`, `Car`, `Plane`, and `Helicopter` classes to the project
- Create a main class and declare a variable of each provided class

Packages

A package is a grouping of related types providing access protection and name space management.

Creating a package is done using a `package` statement. Every source file which contains a type which should be included in that package will have the `package` statement as the first line in the file. Files which do not include a `package` statement are instead part of an unnamed, default package. This is fine for small applications, but in larger ones, classes and interfaces should always be in a named package.

For example, we would write the following to add an abstract class `Bicycle` to a `bicycles` package.

```
/// Bicycle.java
package bicycles;

public abstract class Bicycle {
    //...
}
```

Figure 1 Bicycle abstract class within bicycles package.

Graded Submission Task 1

1. Add all the classes and interfaces in the project, except the main class, to a package called `vehicles` using `package` statements.

2. Eclipse will be showing an error now, "declared package does not match expected package "" ". The empty string is the default package.

   Accept Eclipse's suggestion to fix this error for each class and interface, "Move *.java to package 'vehicles'".

In the package manager, you should notice that a package `vehicles` has been added.

3. The compiler will also be showing some errors in the main class caused by the visibility of the classes. There are two ways to fix this. The first is to add `import` statements for each class before the main class.

   You may notice this means the beginning of the file would have a lot of import statements if you use many classes from different packages. We can instead exchange all the `import` statements with a single `import` statement.

   ```
   import vehicles.*;
   ```

   This means you import all classes and interfaces within the `vehicles` package. This is fine for this small example. For larger projects, you may need to be careful to watch what you are importing. If you happen to write two different `import` statements which import two classes with the same name from different packages there will be conflicts. The cause of the error being the `import` statements may not be immediately apparent to you or future maintainers. So explicitly importing each class you intend to use will clear any confusion.

** 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.

Package naming conventions

There are some naming conventions for packages used to avoid conflicts. Package names are written in all lowercase. Companies use their reverse Internet domain for the beginning of their packages.

You could name a package you develop for this course beginning with `edu.odu.cs.cs251`.

Packages in the file system

Graded Submission Task 2

Packages are directories in a file system. We can see this using Eclipse.

1. Right-click on the `vehicles` package and choose "Show In -> System Explorer".

   This will open the directory containing the `vehicles` package in your system's file manager. Open the vehicles directory and you will find all of the files which are part of the package.

2. Add each of the files in the `vehicles` package to the root package `edu.odu.cs.cs251`

   The Package Explorer will show the package as `edu.odu.cs.cs251.vehicles`. Right-click on this package and choose "Show In -> System Explorer".

This will again take you to the directory containing the `vehicles` package, but you will notice the file path is different. Each successive part of the package name is actually a directory.

Move up the directories to reach the main class in the default package.

3. Fix any errors in the main class caused by the changes made in this task while keeping the `edu.odu.cs.cs251` root package.

4. Submit the Eclipse project and a text file containing the relative file paths for each class in the `vehicles` package

   E.g. The relative file path for a text file in some directories "path/to/file/file.txt"

** 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.

Role-playing game

You have been given an Eclipse project containing work-in-progress code for a role-playing game called "Dungeon Crawler". You may run the code once to see its output. For now, it will print out the names and IDs of entities in an Instance.

The code has developed to the point that there are many files and organization has become a concern. You are tasked with creating packages and dividing the code into these packages.

The root package shall be `edu.odu.cs.cs251.dungeoncrawler`. The main class will be the only class in the root package. From the root package, you should create packages to split the rest of the source code into.

You should also fix any errors that arise because of the new structure.

** 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 packages. A package is a grouping of related types which provides access protection and namespace management. Classes are apportioned to packages using a package statement. The package statement should be the first line in the source file and name the package that the class/interface in the file belongs to.

The common naming conventions used for packages are reverse internet domains. A package for this course could be edu.odu.cs.cs251. Packages correspond to directories in a file system. The package edu.odu.cs.cs251 is the directory relative to the project root `./edu/odu/cs/c251`. Source files belonging to this package would be in that last directory.

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 "Lab16_firstName_lastName".
- Submit the zipped file to the respective Canvas link.

Rubrics		
Task 1 (allotted points) - Compilation (2) - Vehicles package (4) - Appropriate import statements used (4)	Task 2 (allotted points) - Compilation (2) - Text file matches package structure (8)	Task 3 (allotted points) - Compilation (2) - Root package only has main class (2) - Appropriate packages to split classes (6)