CS102A Introduction to Computer Programming
Fall 2020 Lab 10

Objective

1. Learn to create packages to organize classes.
2. Learn to specify `classpath`.
3. Learn to use `enum` types.

1 Before Exercise

1.1 Part 1: Package and `classpath`

Given the `Circle.java` file you wrote in Lab 9 (we added a `main` method for this lab), add the following package declaration statement at the beginning of the `.java` file.

```
1 package sustech.cs102a.lab9;
```

Now go to the directory where the `Circle.java` file resides and run the following command to compile the `Circle.java` file. Observe what would happen.

```
> javac Circle.java
```

You will find a `Circle.class` file appearing in your working directory. If you run the directory listing command, you can see both the `.java` and `.class` files.

Now suppose you want to run the `Circle` class. You might wish to run this command:

```
> java Circle
```

Unfortunately, you will get the following error message:

```
Exception in thread "Main" java.lang.NoClassDefFoundError: Circle (wrong
name: sustech/cs102a/lab9/Circle)
        at java.lang.ClassLoader.defineClass1(Native Method)
```

```
        ...
Could not find the main class: Circle.
```

This is because by adding a package declaration, the fully qualified name of the `Circle` class becomes `sustech.cs102a.lab9.Circle`. We cannot simply use the simple name to run the class. What if we pass the fully qualified name to the `java` command?

```
> java sustech.cs102a.lab9.Circle
Exception in thread "Main" java.lang.NoClassDefFoundError: sustech/cs102a/
lab9/Circle
Caused by: java.lang.ClassNotFoundException: sustech.cs102a.lab9.Circle
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        ...
Could not find the main class: sustech.cs102a.lab9.Circle.
```

Again, no luck. The JVM cannot find the `Circle` class even when we use the fully qualified name. To understand why, we need to know how JVM locates a class.

JVM uses a class loader that searches the classes in the SDK first and then searches the directories specified by the environment variable `classpath` (the default value is `.`, meaning the current directory). With the Java package mechanism, when JVM searches under the current directory for the `Circle` class, it would expect that the `Circle.class` file resides in the directory `./sustech/cs102a/lab9/`. Unfortunately, the parent directory of our `Circle.class` is the current directory.

In order to make things right, you need to use the `-d` option to set the destination directory for the class files that you would like the Java compiler to generate. Now, try the following command.

```
> javac -d . Circle.java
```

With this command, the Java compiler will generate the `Circle.class` file and put it under the `sustech/cs102a/lab9/` directory (the directory will be automatically created if it does not exist). Then, with the following command, you will be able to successfully run the `Circle` class.

```
> java sustech.cs102a.lab9.Circle
main method in sustech.cs102a.lab9.Circle
```

Continue to create a `CircleTest.java` file with the following code:

```
1 package sustech.cs102a.lab10;
2 import sustech.cs102a.lab9.Circle;
3 public class CircleTest {
4     public static void main(String[] args) {
5         Circle c = new Circle(1.0, 0.0, 0.0);
6         c.position();
```

```
    }
}
```

In the code, we need to import the `sustech.cs102a.lab9.Circle` class because it is declared in another package. Compile the `CircleTest.java` with the following command:

```
> javac -d . CircleTest.java
```

You will find that the `CircleTest.class` will be put under `sustech/cs102a/lab10` directory. After compilation, you are able to run the `CircleTest` class by the command:

```
> java sustech.cs102a.lab10.CircleTest
Position of the circle is (0.0, 0.0)
```

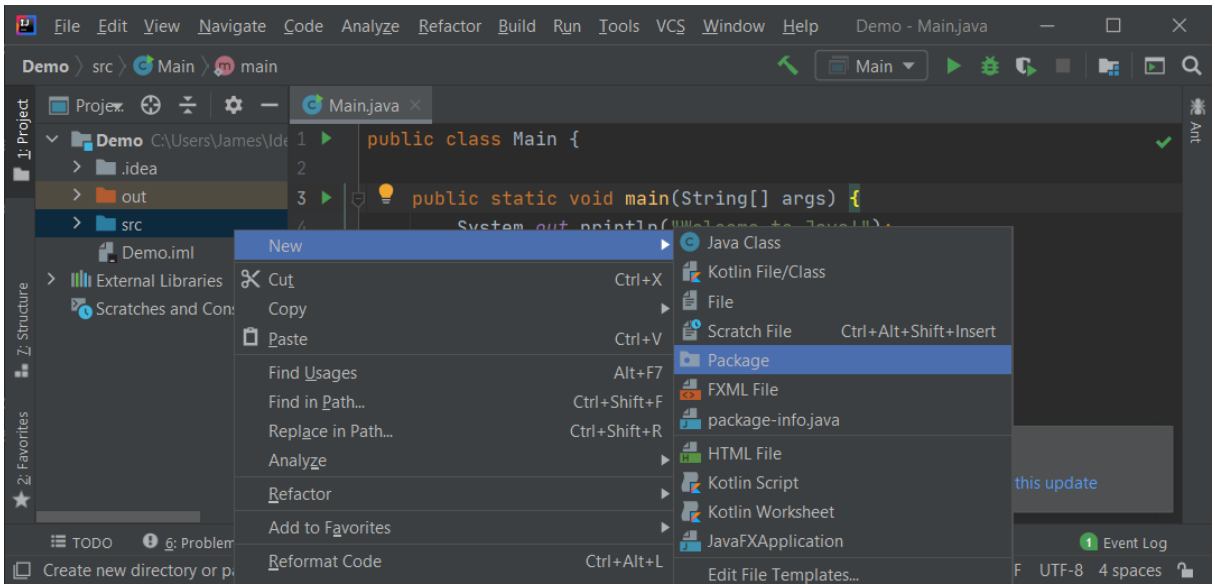
Note that for all the above steps, we assume that we don't change our working directory during the whole process. If we change to another directory and wish to run the `CircleTest` class from there, we need to specify the `classpath` as follows:

```
> java -cp parent-dir-of-sustech sustech.cs102a.lab10.CircleTest
```

If the `classpath` contains several directories, we should use directory separators to separate them. Semicolon (`;`) is the directory separator on Windows. On Unix/Linux/Mac, you should use colon (`:`). Below is an example:

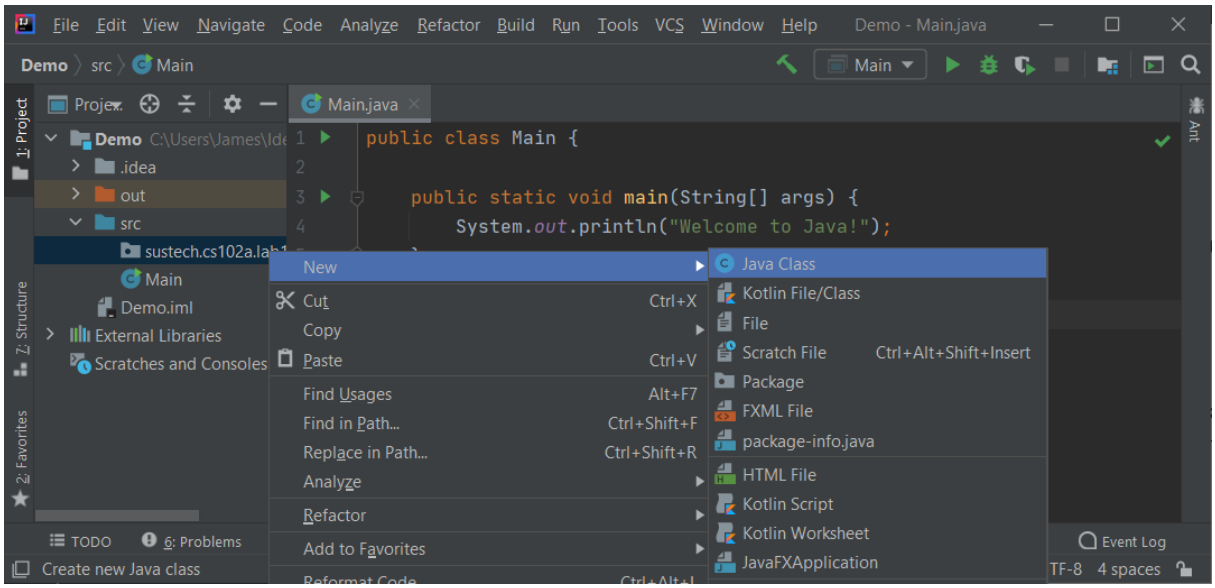
```
> java -cp dir1;dir2;dir3 ClassToRun
```

The above tutorial explains package and classpath at a low level. In an IDE, creating packages and declaring classes in them is as easy as pie. We provide the steps below for IntelliJ IDEA.



In a project, right click on `src`, then click `New` and `Package`. Enter the package name in the dialog box, click *OK* and you will see the package created.

Right click on the package, then click `New` and `Java Class`. Enter the class name in the dialog



box, click *OK* and you will see the skeleton code of the newly created class. You will find that the package declaration statement is added automatically.

Now compile the class and go to the directory where the project is stored. You will see that project directory contains two sub-directories: `src` stores the `.java` source files and `out` stores the `.class` files after compilation. If you check the `src` and `out` directories, you will find that the `TestPackage.java` and `TestPackage.class` files reside in the following directories, respectively:

```
src/sustech/cs102a/lab10
out/production/Tutorial/sustech/cs102a/lab10
```

IDEA helps manage all the stuff automatically. The way how source files and class files are organized may be different for other IDEs (for example, Eclipse), but the `TestPackage.class` file is always put under `sustech/cs102a/lab10` after compilation.

1.2 Enumerations

An enum type is a special data type that enables a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. For example, a week has seven days (`MONDAY` to `SUNDAY`).

A enum type is declared using the enum keyword, not class. Let's create a new enum type Direction with four constants named `NORTH`, `SOUTH`, `EAST`, and `WEST`, respectively. In IDEA, creating a

new enum type is similar to creating a new class. The only difference is to select `Enum` in the dropdown list.

```
package sustech.cs102a.lab10;

public enum Direction {
    NORTH, SOUTH, EAST, WEST // semicolon unnecessary
}
```

Variables of this enum type `Direction` can only receive the values of the four enum constants. For example, the following code creates an object of this enum type.

```
package sustech.cs102a.lab10;

public class DirectionTest {
    public static void main(String[] args) {
        Direction d = Direction.EAST;
        System.out.println(d);
    }
}
```

The above code prints `EAST`. The last statement in the main method is equivalent to `System.out.println(d.toSt` The `toString()` method returns the name of the enum constant `EAST`.

In the code, we cannot create an object of the enum type using the `new` operator with a constructor call. If you compile the following code, you will receive the error message `Enum types cannot be instantiated`. This is because under the hood, every enum type is internally implemented using class (the compiler will create a private constructor that cannot be called outside the enum type).

```
public final class Direction extends Enum {
    public static final Direction NORTH = new Direction();
    public static final Direction SOUTH = new Direction();
    public static final Direction EAST = new Direction();
    public static final Direction WEST = new Direction();
} // simplified for illustration
```

From this internal view, we can see that `NORTH`, `SOUTH`, `EAST`, `WEST` are no more than four class variables pointing to four `Direction` objects. The final modifier makes them constants.

An enum type variable can be passed as an argument to a `switch` statement.

```
package sustech.cs102a.lab10;

public class DirectionTest {
```

```
4
5       private Direction d;
6
7       public DirectionTest(Direction d) {
8           this.d = d;
9       }
10
11      public Direction getDirection() {
12          return d;
13      }
14
15    public static void main(String[] args) {
16      DirectionTest test = new DirectionTest(Direction.EAST);
17      switch(test.getDirection()) {
18          case EAST: // must be unqualified name of the enum constant
19              System.out.println("Countries in the east: Japan, Korea");
20              break;
21          case WEST:
22              System.out.println("Countries in the west: US, Germany");
23              break;
24          case SOUTH:
25              System.out.println("Countries in the south: Australia, New
                    Zealand");
26              break;
27          case NORTH:
28              System.out.println("Countries in the north: Russia, Mongolia")
                    ;
29              break;
30      }
31    }
32  }
```

When declaring an enum type, besides the enum constants, we can also declare other members such as constructors, fields and methods. A enum type constructor can specify any number of parameters and can be overloaded, but it cannot have the access modifier public (must be private or no-modifier, meaning package private).

```
1  package sustech.cs102a.lab10;
2
```

```
public enum Book {
    JHTP("Java: How to Program", "2012"),
    CHTP("C: How to Program"),
    CPPHTP("C++: How to Program", "2012"),
    VBHTP("Visual Basic: How to Program", "2011"),
    CSHARPHTP("Visual C#: How to Program");

    private final String title;
    private final String year;

    private Book(String title, String year) {
        this.title = title;
        this.year = year;
    }

    private Book(String title) {
        this.title = title;
        this.year = "no info";
    }

    public String getTitle() {
        return title;
    }

    public String getYear() {
        return year;
    }
}
```

In the enum type `Book`, there are two fields: `title` and `year`. They are declared to be constants since enum type objects only receive predefined constant values (enum constants). There are two getter methods. There are two overloaded constructors. The two constructors are used in the declarations of the enum constants. For example, when declaring the enum constant `CHTP`, the one-argument constructor is used.

We can further write the following program to test the enum type.

```
package sustech.cs102a.lab10;
import java.util.EnumSet;

```

```java
public class BookTest {
    public static void main(String[] args) {
        System.out.println("All books:");

        for (Book book : Book.values()) {
            System.out.printf("%-10s", book);
            System.out.printf("%-30s", book.getTitle());
            System.out.printf("%s\n", book.getYear());
        }

        System.out.println("\nDisplaying a range of enum constants:");

        for(Book book : EnumSet.range(Book.JHTP, Book.CPPHTP)) {
            System.out.printf("%-10s", book);
            System.out.printf("%-30s", book.getTitle());
            System.out.printf("%s\n", book.getYear());
        }
    }
}
```

In the above example, only five Book objects will be created. The constants such as `Book.JHTP` stores the references to the objects.

The `values()` method is a static method that is automatically generated by the compiler to return an array of the enum constants (an array of references to the objects of the enum type).

The generic class EnumSet's static method `range()` returns a collection of the enum constants in the range specified by two endpoints. In the above code, `range()` takes two enum constants as arguments. The first constant should be declared before the second (the `ordinal()` method of a enum constant can return the position of the constant in all declared constants). If this constraint is violated (for example, when `EnumSet.range(Book.CPPHTP, Book.JHTP)` is used in the code), a `java.lang.IllegalArgumentException` will be thrown.

2 Exercise

1. Create an enum type PhoneModel in the package `sustech.cs102a.lab10`, which contains the following constants: `IPHONE`, `HUAWEI`, `PIXEL`, `SAMSUNG`, `LG`.
2. Create a field named `price` (`int` type). Write a getter method for this field.

3. Create a one-argument constructor `PhoneModel(int price)` that can be used to create the enum constants. The prices for the five models are: 9999, 8888, 6666, 9399, 5588.

4. Write a test program `Lab10E1.java`. The class is also in the package `sustech.cs102a.lab10`. It contains a main method that recommends possible phones for a user based on the user's budget.

Three sample runs:

```
Your budget: 4000
You do not have sufficient money
```

```
Your budget: 8888
HUAWEI      price: 8888
PIXEL       price: 6666
LG          price: 5588
```

```
Your budget: 10000
IPHONE      price: 9999
HUAWEI      price: 8888
PIXEL       price: 6666
SAMSUNG     price: 9399
LG          price: 5588
```