CS102A Introduction to Computer Programming

Fall 2020

Lab 10

Objectives

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

1 Prework

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 that a `Circle.class` file has appeared 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 try to run this command:

```
> java Circle
```

Unfortunately, you would 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.
```

That is because, by adding a package declaration, the full name of the `Circle` class becomes `sustech.cs102a.lab9.Circle`; we cannot just use `Circle` to run the class. What if we pass the full 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. JVM cannot find the `Circle` class even when we use its full name. To understand why, we need to look at how JVM locates a class.

JVM uses a class loader that first searches for the classes in the SDK and then searches within 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 rectify this issue, 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

already 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
```

Next, 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();
7     }
8 }
```

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

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

You will find that the `CircleTest.class` will be put under the `sustech/cs102a/lab10` directory. After compilation, you may run the `CircleTest` class via the following command:

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

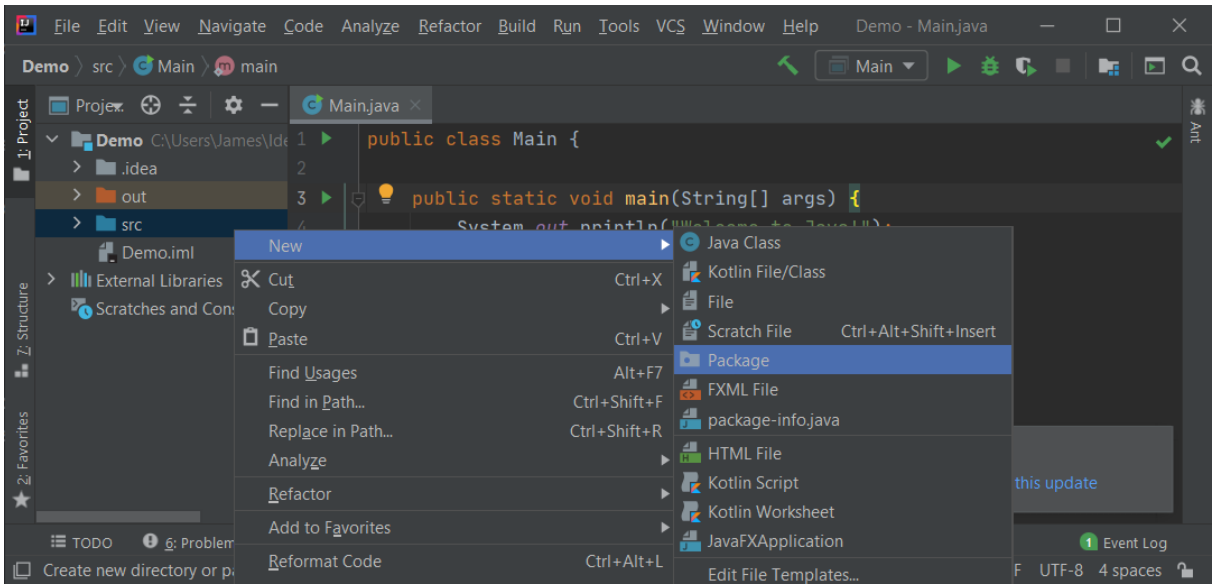
Note that, for all the above steps, we assume that we do not change our working directory during the whole process. If we switch to another directory and wish to run the `CircleTest` class from there, we need to specify the `classpath` as follows:

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

If the `classpath` contains several directories, we must use directory separators to separate them. On Windows, the semicolon (`;`) is used as the directory separator. On Unix/Linux/Mac, you may use the colon (`:`). For example:

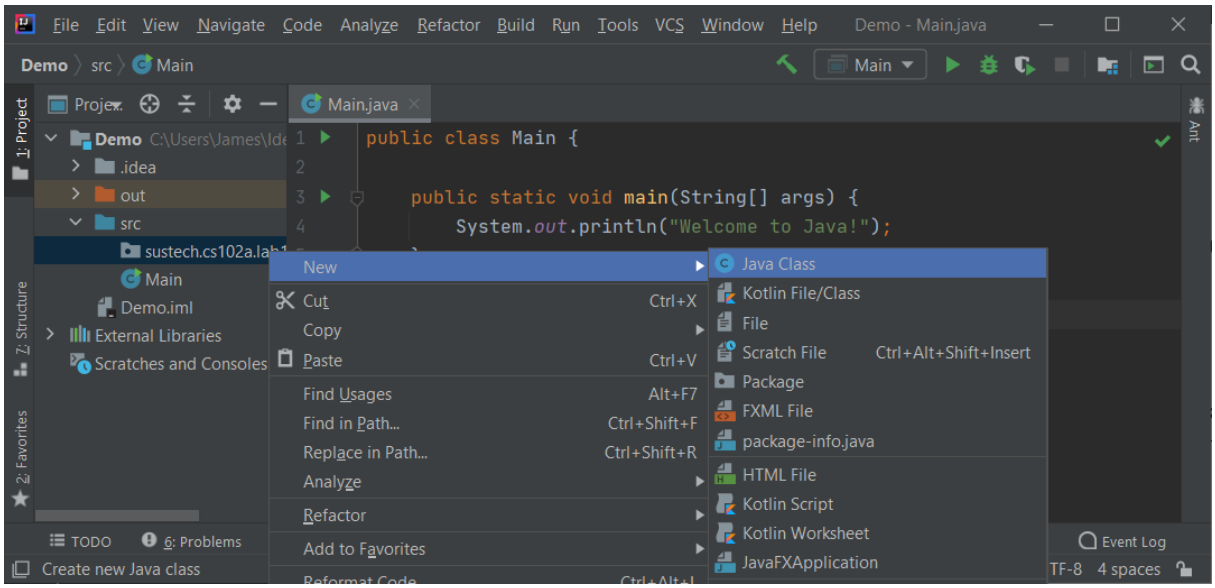
```
> java -cp dir1;dir2;dir3 ClassToRun
```

The above tutorial explains `package` and `classpath` at a low level. Creating packages and declaring classes in an IDE is fairly straightforward. We provide the necessary steps for IntelliJ IDEA below.



In a project, right click on *src*, then click *New* and *Package*. Enter the package name in the dialog box and click *OK*. You will see that the package has been 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 has been added automatically.

Next, compile the class and go to the directory where the project is stored. You will see that the project's directory contains two sub-directories: `src` stores the `.java` source files and `out` stores the `.class` files after compilation. If you browse 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 everything automatically. The way that source and class files are organized may be different from other IDEs (for example, Eclipse), but the `TestPackage.class` file is always put under `sustech/cs102a/lab10` after compilation.

1.2 Enumerations

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

An `enum` type is declared using the `enum` keyword. Let us 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 that one must select *Enum* from the dropdown list.

```
1 package sustech.cs102a.lab10;
2
3 public enum Direction {
4     NORTH, SOUTH, EAST, WEST // semicolon unnecessary
5 }
```

Any `Direction` variables can only take the values of the above four `enum` constants. For example, the following code creates an object of this `enum` type:

```
1 package sustech.cs102a.lab10;
2
3 public class DirectionTest {
4     public static void main(String[] args) {
5         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.toString())`. The `toString()` method returns the name of the `enum` constant `EAST`.

In our 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 in fact 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` variable can be passed as an argument to a `switch` statement:

```
package sustech.cs102a.lab10;

public class DirectionTest {

    private Direction d;

    public DirectionTest(Direction d) {
        this.d = d;
    }

    public Direction getDirection() {
```

```
        return d;
    }

  public static void main(String[] args) {
    DirectionTest test = new DirectionTest(Direction.EAST);
    switch(test.getDirection()) {
        case EAST: // must be unqualified name of the enum
           constant
            System.out.println("Countries in the east: Japan,
               Korea");
            break;
        case WEST:
            System.out.println("Countries in the west: US,
               Germany");
            break;
        case SOUTH:
            System.out.println("Countries in the south: Australia
               , New Zealand");
            break;
        case NORTH:
            System.out.println("Countries in the north: Russia,
               Mongolia");
            break;
    }
  }
}
```

When declaring an `enum` type, apart from the `enum` constants, we can also declare other members such as constructors, fields, and methods. An `enum` type constructor can specify any number of parameters and can be overloaded, but it cannot have a `public` access modifier (must either be `private` or have no modifier at all, implying a private package).

```
package sustech.cs102a.lab10;

```

```
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 and 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;

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`, store 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` has a static method `range()` that 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 an `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 `sustech.cs102a.lab10` package. It contains a `main` method that recommends possible phones for a user based on the user's budget.

Here are 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
```