Tutorial of Enum and package

> Based on the tutorial of "2020S-Java-A" designed by teaching group in SUSTech
>
> Modified (only change to markdown file) by ZHU Yueming in 2021. April. 19th

Objective

- Learn to create packages to organize classes
- Learn to use `enum` types

Part 1: Package and classpath

1. By command line

Introduce

Given the `Circle.java` file you wrote previously, add the following package declaration statement at the beginning of the `.java` file.

```
package sustech.cs102a.lab8;
```

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 or check in the directory window, 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:

```
错误: 找不到或无法加载主类 Circle
```

or

```
D:\Dropbox\teaching\CS102A-Spring-2019\Labs\Lab10\code>java Circle
Exception in thread "main" java.lang.NoClassDefFoundError: Circle (wrong name: sustech/cs102a/lab9/Circle)
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClassCond(ClassLoader.java:631)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:615)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:141)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:283)
        at java.net.URLClassLoader.access$000(URLClassLoader.java:58)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:197)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
Could not find the main class: Circle.  Program will exit.
```

This is because by adding a package declaration, the fully qualified name of the Circle class becomes “sustech.cs102a.lab8.Circle”. We cannot simply use the simple name to run the class.

Try

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

It should be run the following command in the directory “sustech/cs102a/lab8” included.

```
java sustech.cs102a.lab8.Circle
```

Tips: the “javac” command may run in 2 ways:

- In directory “lab8”: `javac Circle`
- In upper directory of “sustech”: `javac sustech\cs102a\lab8`

In both ways, the `Circle.class` will be in the directory “sustech\cs102s\lab8”.

Now, continue to create a `CircleTest.java` file with the following code in lab9:

```
package sustech.cs102a.lab9;
import sustech.cs102a.lab8.Circle;

public class CircleTest {
    public static void main(String[] args) {
        Circle c = new Circle(1.0, 0.0, 0.0);
        c.position();
    }
}
```

In the code, we need to import the `sustech.cs102a.lab8.Circle` class because it is declared in another package.

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
```

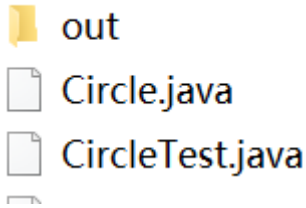
If the classpath contains several directories, we should use directory separators to separate them. Semicolon(;) is the directory separator on Windows. On Unix/Linux/Mac, you should use colon (:). Below is an example:

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

Exercise

Try following command:

- check files in current path. The out folder is an empty folder



- compile `Circle.java` into out folder.

  ```
  javac -d out Circle.java
  ```

  A `Circle.class` file would be created in the path of `./out/sustech/cs102a/lab8`

- compile `CircleTest.java` in current folder and use the `Circle.class` in `./out/sustech/cs102a/lab8`

  ```
  javac -cp out -d . CircleTest.java
  ```

  A `CircleTest.class` file would be created in the path of `./sustech/cs102a/lab9`

- Run `CircleTest`

  ```
  java -cp out; sustech.cs102a.lab9.CircleTest
  ```

  or

  ```
  java -cp out: sustech.cs102a.lab9.CircleTest
  ```

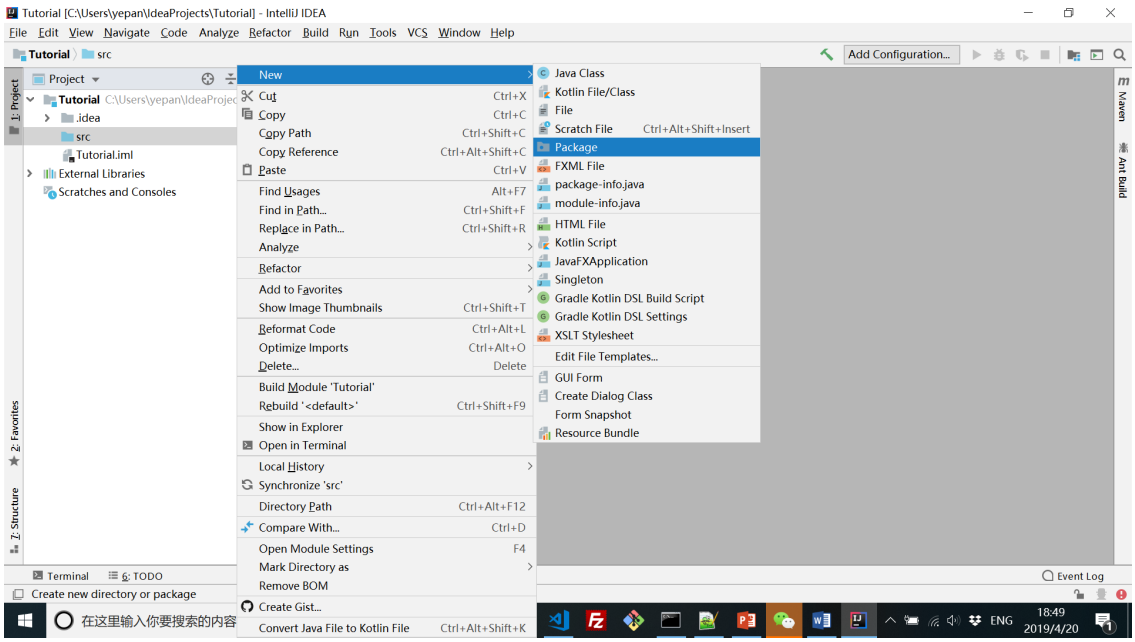
  result:

  ```
  Position of the circle is (0.0, 0.0)
  ```

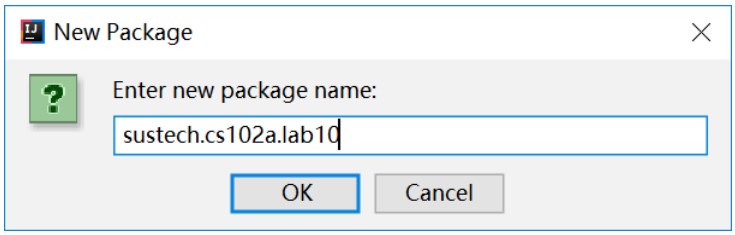
2. By Intellij IDEA

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.

- Step 1: In a project, right click on the src, then click New->Package

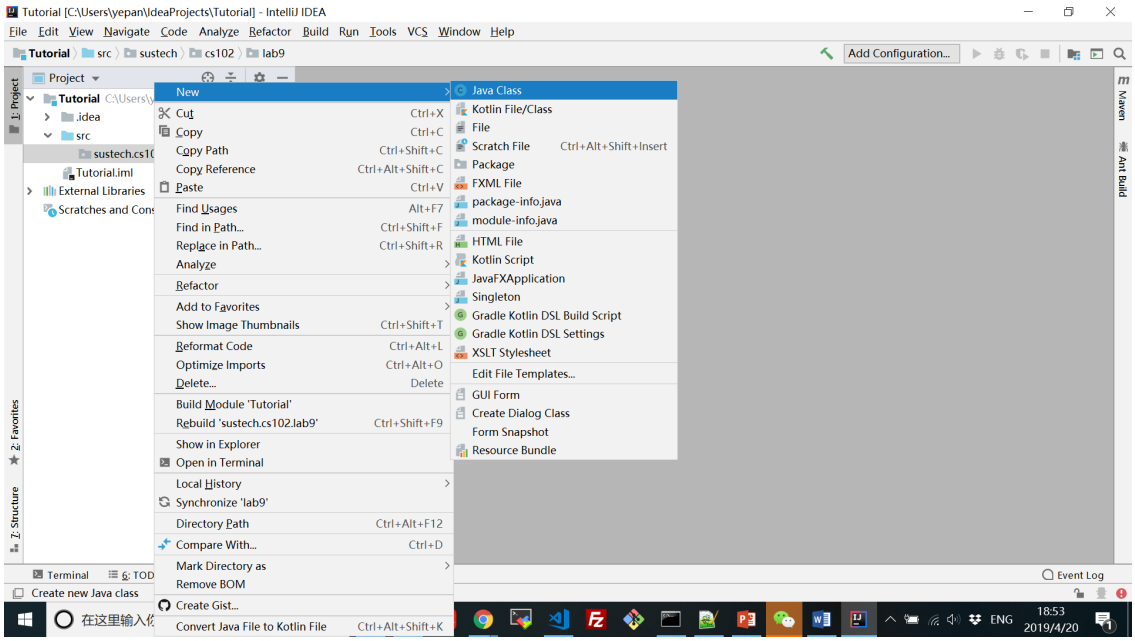


- Enter the package name in the dialog box, click ok and you will see the package created.

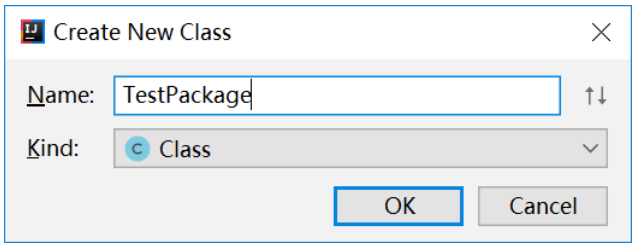


(correction: please type “sustech.cs102a.lab9” here)

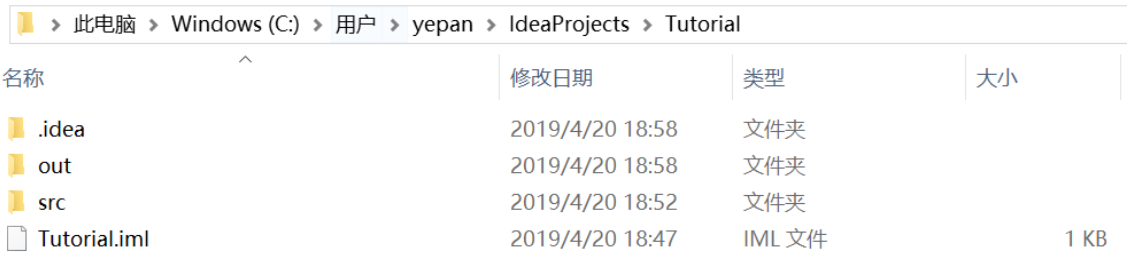
- Right click on the package, then click New->Java Class



- 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:

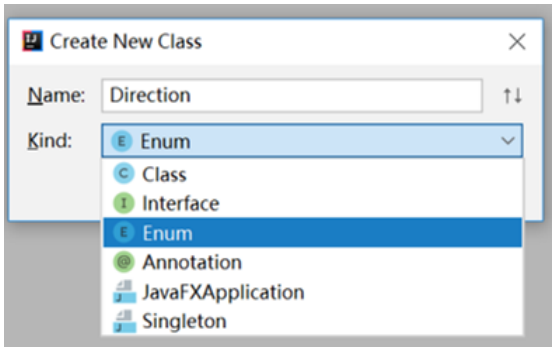
```
src/sustech/cs102a/lab9
out/production/[Project Name]/sustech/cs102a/lab9
```

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/lab9` after compilation.

Part 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.




```java
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.toString())`. 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".

```
public Direction d = new Direction();
```

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 {

  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, 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).

```
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. 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.

```java
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());
        }
    }
}
```

The code prints:

```
All books:
JHTP      Java: How to Program          2012
CHTP      C: How to Program             no info
CPPHTP    C++: How to Program           2012
VBHTP     Visual Basic: How to Program  2011
CSHARPHTP Visual C#: How to Program     no info

Displaying a range of enum constants:
JHTP      Java: How to Program          2012
CHTP      C: How to Program             no info
CPPHTP    C++: How to Program           2012
```

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` is 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), an `java.lang.IllegalArgumentException` will be thrown.

Exercise

1. Create an `enum` type `PhoneModel`, 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 .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 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
```