Objective

Package in Java
How to create and organize package

Package

Packages are used in Java in order to prevent naming conflicts. It is also used to organize related data types (classes, interfaces, enumerations, ...)

Some of the packages of Java API:

- java.util
- java.lang

Java package

To use a class, we import it into our source code using import Java keyword

```
import java.util.Scanner;
```

that imports Scanner class into our source code.

Or we can use symbol, *, to import all of the classes of a package:

```
import java.util.*;
```

that imports all component of java.util package into to our source code.

Question 1

Why should we be cautious using java symbol, *, to import all component of a package?

package/folder

Package and folders are the same in java terminology. packages are related to the folder names in the file system.

Lets see it through an example: I have four java files

interface Drive.java

classes Car.java and Sedan.java

Test.java with a main method as a test harness.

Refer to example1

Example 1

```
public interface Drive {
   public void move();
   public void stop();
}
//----
public class Car implements Drive {
   public void move(){
      System.out.println("Car Driving");
   }
   public void stop(){
      System.out.println("Car Stopped");
   }
}
```

Example 1 (continued)

```
public class Sedan extends Car{
  public String getType(){
    return "sedan";
  }
}
//----
public class Test{
  public static void main(String[] args){
    Sedan myCar = new Sedan();
    myCar.move();
    myCar.stop();
    System.out.println(myCar.getType());
  }
}
```

Organize in package

In example1, we did not used package to organize our files, so the compiler assume that all classes are in the default package which is our current working directory.

Assume that we want to organize our classes into following packages.

File Name	Package Name
`Sedan.java`	`sedan`
`Drive.java`	`sedan.vehicle`
`Car.java`	`sedan.vehicle`

Example 2

and this is what we want to have:

`sedan.Sedan.java`

`sedan.vehicle.Drive.java`

`sedan.vehicle.Car.java`

Create folders

Think of the packages as a folder in a file system.

Step 1: Create sedan and sedan/vehicle folders in our current working directory.

Notes:

1. The folder names matches exactly the package name, and sedan is our top folder.

2. In this example sedan is our top package and vehicle in a nested package.

Moving files into folders

Step 2: Move Sedan.java, Drive.java, and Car.java into the folders that corresponds to the package names:

note that dot (‘.’) stands for current folder in the file system.

- *// current working folder*

Test.java

sedan *// folder*

Sedan.java

vehicle *// folder*

Car.java

Drive.java

Filenames are shown in green and folders are shown in brown just for clarification purpose.

Add package names

Step 3: Add the package names to top of each source file that matches the folder they are saved. Package name should be the first Java statement in the source file.

```
// File Drive.java

package sedan.vehicle;

Public interface Drive {

   ...

}

// File Car.java

package sedan.vehicle;

public class Car implements Drive {

  ...

}
```

Both classes are located in folder sedan/vehicle

Add package names

what about Sedan? Note that Sedan.java is in folder sedan, so we need to add the package name to the top of the source file.

```
// File Sedan.java

package sedan;

public class Sedan extends Car{

  ...

}
```

Import the needed classes

However, class Sedan also uses class Car as a super class. So we also need to import class Car.

```
// File Sedan.java

package sedan;

Import sedan.vehicle.Car;

public class Sedan extends Car{

  ...

}
```

Important: Unlike the file systems, we cannot use relative addressing in importing/naming packages.

Everything starts from the top package that is package sedan in this example.

Test.java

Everything is done. Now we have organized our classes with package/folders, but what about Test.java?

Test.java is located in the current working directory, and it is not part of the package in this example. Compiler assumes default package for it.

However, it uses class Sedan in package sedan. So we import it too. Here Test.java is modified to:

```
// File Sedan.java

import sedan.Sedan;

public class Test{

   ...

}
```

Refer to example2

Question 2

To use class Car in Sedan, we need to import it.

```
// File Sedan.java

package sedan;

Import sedan.vehicle.Car;

public class Sedan extends Car{

  ...

}
```

However, we also used interface Drive in class Car, but we did not import it.

Why?

Question 3

Can I have two or more package names on top of a source file?

Something like:

package sedan.vehicle;

package cycle.motorcycle;

Why?

Question 4

Assume I have another class Engine in package sedan.vehicle;

```
// file Engine.java

package sedan.vehicle;

// import ???

class Engine{

  private Sedan myCar;

}
```

and, I am using class Sedan. How should I import it?