211lab10_S11

Lab 10 Java Packages, Generics;

Java Packages

A Java package is a grouping of classes similar to the notion of a directory is a grouping of files. Packages are used to help avoid name clashes and to hide particular pieces of code from the clients. A package has a name, such as `utility` or `java.lang.util` . In general, a package name is a series of strings of alphanumeric characters (starting with an alphabetic character) and separated by periods. To make a java class part of a particular package, say `funList` , you must add the declaration package `funList` ; to the very top of the class source file.

Also, you will need to put the file in the directory structure that mirrors the package name. For example, the java classes that belong to the package `funList` should be in a directory also named funList. If you don't do this, it will still compile, but it won't run correctly.

NOTE: DrJavalanguage levels do not support packaging. We must use Full Java to work with packages.

Exercises:

- 1. Create a subdirectory called lab10 for all the files of this lab. Create and save a separate file for each of the following classes and interface.

```
/**
 * Abstract list structure.
 */
public abstract class IntList {
    public abstract Object accept(IntListVisitor v);
    public ConsIntList cons(int n) {
        return new ConsIntList(n, this);
    }
}

/**
 * Concrete empty list structure containing nothing.
 */
public class EmptyIntList extends IntList {
    public static final EmptyIntList ONLY = new EmptyIntList();
    private EmptyIntList() {
    }

    public Object accept(IntListVisitor v) {
        return v.forEmptyIntList(this);
    }
}

/**
 * Concrete non-empty list structure containing an int, called first,
 * and a rest, which is a list structure.
 */
public class ConsIntList extends IntList {
    private int first;
    private IntList rest;

/* NOTE: Programmer must write constructor code and gettors code in full Java.
 */
    public ConsIntList(int f, IntList r) {
        first = f;
        rest = r;
    }

    public int first() {
        return first;
    }

    public IntList rest() {
        return rest;
    }

    public Object accept(IntListVisitor v) {
        return v.forConsIntList(this);
    }
}

/**
 * Abstract operation on IntList.
 */
public interface IntListVisitor {
    public Object forEmptyIntList(EmptyIntList host);
    public Object forConsIntList(ConsIntList host);
}
```

- 2. Add the declaration package `funList` ; to the top of `IntList.java` . Compile it using DrJavaTools/Compile Current document. You should get an error message saying that you are in the wrong package. Close the file for now.

You need to create a subdirectory called `funList` and move `IntList.java` into it. The full class name for `IntList` is now `funlist.IntList`.

Reopen the file in the `funList` subdirectory. Now compile again. You should get error messages saying it can't find `class IntListVisitor` and `class ConsIntList` this time. You will need to package all the other classes/intefaces and move them into appropriate subdirectories.

- 3. Add the package `funList` ; declaration to the top of `EmptyIntList.java` , `ConstIntList.java` , and `IntListVisitor.java` , and move them into the `funList` subdirectory. You should be able to compile each file individually. Try it.

Note: if you use the command window to compile with the command javac, you should always compile from your project's main directory. If you compile from within a package subdirectory, it doesn't find all the supporting definitions.

We can't run anything yet, because that's just a piece of the whole program.

- 4. Create a JUnit test class called `TestEmptyIntList` . Do not make `TestEmptyIntList.java` part of the package. `TestEmptyIntList.java` does not have a package name, and is thus said to be in the no-name (or default) package. Save `TestEmptyList.java` in lab10 subdirectory (right above `funlist`).

Add code to test the `accept` method of `EmptyIntList` . What can we do here?

If you try to compile `TestEmptyIntList.java` now, you will get an error message. Try it to see what happens.

You need to add the statement `import funList.*` ; to the top of `TestEmptyIntList.java` to indicate to the compiler that you are using all the public classes in that package. Try to compile it again. Is everything OK?

Now, remove the `public` access from the `EmptyIntList` class. By default, a class is "package-private", i.e., it is known within the package, but not from outside. Try to compile again. You should see a few error messages saying that you can't use `EmptyIntList.java` because it is not public. This is because the `TestEmptyIntList` class is not part of the `funList` package. One way to resolve this problem by making `TestEmptyIntList` part of the `funList` package. A class of a package can access all the classes (public or "package-private") in the package. However this is not a good solution in general because a client may be using many classes from different packages, but no class can be part of more than one package. For now, just make `EmptyIntList.java` public again, and recompile `TestEmptyIntList.java` . You should get no error. Try to run `Test_List.java` now by click the Test button in DrJava.

Java Generics

Please refer to the Lecture 30 notes. Also, download and open the contained DrJava project in lec29_code.zip .

Practice writing some visitors using generics:

1. Forward accumulation sum of a list of integers.
2. Concatenate, with spaces in between, all the elements of a list of Strings.
3. A single ToString visitor class that could be used to create instances that could then be used on any given type of list.

DrJava Projects

Here's some more information on using DrJava's project capabiltiies:
DrJavaprovides a utility called Project as a way to manage large Java programs with many files and many packages. We will illustrate the use of DrJavaProject by writing a few visitors for the `funlist` package.

In general, a project has a bunch of java files (the source code) and class files. It is a good idea to separate the java files from the compiled class files by creating a subdirectory called `bin` for the class files and `src` for the java files.

- 1. Inside of lab10 subdirectory, create subdirectory called bin and a subdirectory called src. Move the `funlist` subdirectory inside of src.

- 2. In !DrJava, use the menu Project/New to create a DrJavaproject, save it as `ListVisProj` inside of `lab10` . A dialog window will popup asking for the Project Root, Build Directory and Working Directory, etc.

Set the project root directory lab10, the build directory to lab10/ `bin` and working directory to lab10/ `src` . Click OK and now we have an empty project file called {{ListVisProj.drjava }} saved in xml format. Take a look at the subdirectory lab10 to see what's there.

- 3. To add the whole `funlist` package to the project, use the File/Open Folder and select the folder `src` (and be sure the check the Open folder recursively checkbox). All the java files in `funlist` should be displayed. Save the project. Compile the project by clicking on the Compile Project button. Everything should compile. Check the subdirectory bin to see all the class files generated by the compiler.

Now let's write a list visitor to compute the length of a list and add it to the project.

- 4. Here is the list visitor code. It is not quite compilable on purpose.

```
/**
 * Computes the length of the host list using a tail-recursive
 * helper visitor.
 */
public class GetLength implements IntListVisitor {
    static GetLength ONLY = new GetLength();
    private GetLength() {
    }

    /**
     * @return an Integer
     */
```

```
    Object forEmptyIntList(EmptyIntList host) {
        return 0;
    }

  /**
     * @return an Integer
     */
    Object forConsIntList(ConsIntList host) {
        return host.rest().accept(new GetLengthHelp(1));
    }
}

class GetLengthHelp implements IntListVisitor {
    int acc;

    // need constructor

    Object forEmptyIntList(EmptyIntList host) {
        return acc ;
    }

   Object forConsIntList(ConsIntList host) {
        return host.rest().accept(new GetLengthHelp(acc + 1));
    }
}
```

Save it in a subdirectory of `funlist` called visitor. The full class name for `GetLength` and `GetLengthHelp` are now `funlist.visitor.GetLength` and `funlist.visitor.GetLengthHelp`, respectively. Note that `GetLength` is `public` while `GetLengthHelp` is `package private`. Now try to compile it! You will see a few error messages. Fix the errors until everything compiles.

- 5. Write a JUnit test class called `TestGetLength` to test the `GetLength` visitor written in the above. Save this test class in a subdirectory of visitor called test. You will need to create a new folder when you try to save the `test` class.
    1. Add appropriate packaging declaration so that it compiles.
    2. Incrementally add test code and incrementally compile the test file.
    3. Incrementally run the test code by clicking the Test Project button (or the menu Project/Test Project).

Be sure to save the project code periodically.

- 6. Write a visitor called `ToString` to compute a Scheme-like String representation of the list. (See ListString visitor done in lab 09). Package it in `funlist.visitor` . Write stub code only and make sure that everything compiles.
    1. Now write a JUnit test class called `TestToString` to test `ToString` . Package it in the `test` subdirectory as done in the above step 5. Make sure it compiles.
    2. Incrementally write ToString and incrementally test it until all the test code pass.
