Prepared by Linan Qiu <lq2137@columbia.edu>

This note is prepared with extensive lifting from Java Lessons

Packages

A World Without Packages

Let's say we have the following classes:

```java
// Pokemon.java
public class Pokemon {
  // ...
}

// Trainer.java
public class Trainer {
  // ...
}

// GymBoss.java
public class GymBoss extends Trainer {
  // ...
}

// Rectangle.java
public class Rectangle {
  // ...
}
```

They'd exist in 3 separate files. In this class, this would be perfect. You really don't need to do anything else.

However, in big (usually enterprise / team) projects, they'd get messy. Imagine lots of classes like `Tunnel.java`, `Item.java`, `Pokeball.java` etc. You'd want to sort them in some way or another right? Or if you're the more adventurous type, you'd have tried to put them in folders. Unfortunately, that doesn't really work during compilation right? You'd also want to distinguish them from the Digimon / Lord of the Rings project you're working on.

Packages solves this problem. In fact, it solves a lot more. You should use packages in the real world for these reasons:

- You and other programmers can easily determine that these types (a class is a **type**) are related.
- You and other programmers know where to find types that can provide related functions (such as Pokemon, or even within Pokemons, trainers).

- The names of your types won't conflict with the type names in other packages because the package creates a new namespace. (Having `Trainer` in Pokemon won't conflict wiht `Trainer` in Digimon)
- You can allow types within the package to have unrestricted access to one another yet still restrict access for types outside the package.

Creating Packages

To create a package, simply do what we told you not to do the entire semester – add that `package whatever;` line at the start of your source file. For example, in our code above, we'd do:

```java
// Pokemon.java
package pokemon;
public class Pokemon {
  // ...
}

// Trainer.java
package pokemon;
public class Trainer {
  // ...
}

// GymBoss.java
package pokemon;
public class GymBoss extends Trainer {
  // ...
}

// Rectangle.java
package pokemon;
public class Rectangle {
  // ...
}
```

The `package whatever;` must be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

If you do not use a package statement, your type ends up in an unnamed package. (i.e. the **default package**. This is what we have been asking you to do the entire class. This makes for easy compilation and debugging.) Generally speaking, an unnamed package is only for small or temporary applications or when you are just beginning the development process. Otherwise, classes and interfaces belong in named packages.

Naming Convention

With programmers worldwide writing classes and interfaces using the Java programming language, it is likely that many programmers will use the same name for different types. In fact, the previous example does just that: It defines a `Rectangle` class when there is already a `Rectangle` class in the `java.awt` package. Still, the compiler allows both classes to have the same name if they are in different packages. The fully qualified name of each `Rectangle` class includes the package name. That is, the fully qualified name of the `Rectangle` class in the `pokemon` package is `pokemon.Rectangle`, and the fully qualified name of the Rectangle class in the `java.awt` package is `java.awt.Rectangle`.

Package names are written in all lower case to avoid conflict with the names of classes or interfaces.

Companies use their reversed Internet domain name to begin their package names—for example, `com.example.mypackage` for a package named mypackage created by a programmer at `example.com`.

Packages in the Java language itself begin with `java.` or `javax.`

Using Package Members

The types that comprise a package are known as the **package members**.

To use a public package member from outside its package, you must do one of the following:

- Refer to the member by its fully qualified name
- Import the package member
- Import the member's entire package

Using Fully Qualified Name

Let's use our Pokemon example earlier. Let's say my directory is organized as such:

```
code/pokemon/Pokemon.java
code/pokemon/Trainer.java
code/pokemon/GymBoss.java
code/pokemon/Rectangle.java
code/Test.java
```

Notice that my `Test.java` is outside the package and is in the default package.

```
// Test.java
public class Test {
  public static void main(String[] args) {
```

```java
    // stuff goes here
  }
}
```

I can use `Trainer` like this:

```java
// Test.java
public class Test {
  public static void main(String[] args) {
    pokemon.Trainer ash = new pokemon.Trainer();
  }
}
```

Import Package Member

However, typing `pokemon.Trainer` each time is annoying. So I can import `pokemon.Trainer`. This would tell Java that every time it sees `Trainer` in my source code, I mean `pokemon.Trainer`.

```java
// Test.java
import pokemon.Trainer;

public class Test {
  public static void main(String[] args) {
    Trainer ash = new Trainer();
  }
}
```

Importing Entire Package

If I'm even lazier, I can import the entire package using the asterisk character.

```java
// Test.java
import pokemon.*;

public class Test {
  public static void main(String[] args) {
    Trainer ash = new Trainer();
    Pokemon pikachu = new Pokemon();
    // notice I can use Pokemon without doing pokemon.Pokemon since
    // I already imported all members of pokemon
  }
}
```

Applying this to Java's `LinkedList`

You've probably already done this before (without knowing that you did it!) for classes like `ArrayList` or `LinkedList`.

In fact, any of these statements are valid:

```java
// TestAnother.java
public class TestAnother {
  public static void main(String[] args) {
    java.util.LinkedList<String> list = new java.util.LinkedList<>();
  }
}
```

```java
// TestAnother.java
import java.util.LinkedList;

public class TestAnother {
  public static void main(String[] args) {
    LinkedList<String> list = new LinkedList<>();
  }
}
```

```java
// TestAnother.java
import java.util.*;

public class TestAnother {
  public static void main(String[] args) {
    LinkedList<String> list = new LinkedList<>();
    ArrayList<String> anotherList = new ArrayList<>();
  }
}
```

This also means that if you want to use both Java's `LinkedList` and your own linked list (say `LinkedList` that is in the package `myhomework`), you can do this:

```java
// TestYetAnother.java
public class TestYetAnother {
  public static void main(String[] args) {
    java.util.LinkedList<String> javaList = new java.util.LinkedList<>();
    myhomework.LinkedList<String> myList = new myhomework.LinkedList<>();
  }
}
```

or if you have already imported java's `LinkedList`, you can still use `myhomework.LinkedList` by fully specifying its name:

```java
// TestYetAnother.java
import java.util.LinkedList;
```

```
public class TestYetAnother {
  public static void main(String[] args) {
    LinkedList<String> javaList = new LinkedList<>();
    myhomework.LinkedList<String> myList = new myhomework.LinkedList<>();
  }
}
```

Compiling

To compile your code with packages (say `Trainer.java`), simply do

```
$ javac pokemon/Trainer.java
```

This will create a file `Trainer.class` in the folder `pokemon`

And to run it (if you have a `main` method inside `Trainer`) , type

```
$ java pokemon.Trainer
```