HBO Graduaat Informatica Optie Programmeren

Java Basics

Packages

Introduction

- Objectives
  - To explain the need for structure and grouping
  - To introduce the related Java syntax
- Chapter content
  - Package structure – rationale and Java syntax
  - Related compiler options
  - Related JVM syntax and options
- Practical content
  - Familiarization with packages
- Summary

Packages

- One can put groups of related types into *packages*
- Advantages:
  - classes are easier to find
  - avoid name conflicts
  - control access
  - easily determine that these grouped types are related

Packages

Definition:

A package is a grouping of related types providing access protection and name space management

Packages

- Java platform classes are members of various packages

operator	description
java.lang	Fundamental classes (String, Math, System, …) Classes of this package must not be imported
java.io	Input/output (File, InputStream, Serializable, EOFException, ...)
java.util	Utilities (List, Iterator, Random, Date, ...)
java.sql	Accessing and processing data stored in a data source (Connection, Statement, ResultSet, …)

- You can put your types in packages too

Creating a Package

1. Choose a name for the package
2. Add a package declaration as first statement in source file

```
package be.leerstad;

public interface CarDAO {
  List findAll();
  Car findById(int id);
  void create(CarDTO car);
  void update(CarDTO car);
  void delete(String[] ids);
  ...
}
```

- if you declare more than one class in a source file, only one class can be `public`. Others cannot have a visibility modifier, but can be declared `static` and/or `final`.
- So, all other classes have at most *package* visibility.

Packages

- Are packages mandatory?
  - Without a package statement, your type ends up in an unnamed package.
  - Unnamed package is only for small or temporary applications

Packages

- Naming conventions (recommendation)
  - Package names are written in all lowercase
  - Companies use the reverse Internet domain name (hyphens or reserved words are prohibited)
  - Packages in the Java language itself begin with `java.` or `javax.`
  - Examples: `com.ibm`, `org.apache`, ...

- The NATO organization's domain name is nato.int, but `nato.int` is a forbidden package identifier because of the `int` part; so, `_int.nato` seems a good trade-off instead.
- There are exceptions: Oracle packages do not contain the *com* prefix: `oracle.xml.sql`, `oracle.jdbc` ...

Packages

Referring to a class declared in a package

```
package be.leerstad;

public class Car {
  private Engine engine;
  public void setEngine(String engine) {
    this.engine = engine;
  }
  ...
}
```

```
package be.leerstad;

public class Engine {
  private int power;
  public Engine (int p) {
    this.power = p;
  }
  ...
}
```

We can refer to a class by its FQCN (Fully Qualified Class Name)

```
package org.test;

public class Demo {
  public static void main(String[] args) {
    be.leerstad.Engine e = new be.leerstad.Engine();
    be.leerstad.Car c = new be.leerstad.Car();
    c.setEngine(e);
  }
}
```

Packages

Referring to a class declared in a package

```java
package be.leerstad;

public class Car {
  private Engine engine;
  public void setEngine(String engine) {
    this.engine = engine;
  }
  ...
}
```

```java
package be.leerstad;

public class Engine {
  private int power;
  public Engine (int p) {
    this.power = p;
  }
  ...
}
```

- We can refer to a class by its name, with an `import` declaration
- You can use a wildcard: `import be.leerstad.*;`

```java
package org.test;
import be.leerstad.Car;
import be.leerstad.Engine;
public class Demo {
  public static void main(String[] args) {
    Engine e = new Engine();
    Car c = new Car();
    c.setEngine(e);
  }
}
```

Packages

- Apparent Hierarchies of Packages
  - Packages appear to be hierarchical, but they are not, only the names are.
  - The classes (and interfaces) in the `be.leerstad.util` package are not in the `be.leerstad` package!
  - The following clause

```
import be.leerstad.*;
```

does not import types from `be.leerstad.util` package, only those from `be.leerstad`

Packages

- Static imports
  - For static members, use a static import:

```
public class Demo {
  public static void main(String[] args) {
    ...
    surface = 2 * radius * Math.PI;
    volume = Math.pow(radius, 3.0) * Math.PI * 4 / 3;
    ....
  }
}
```

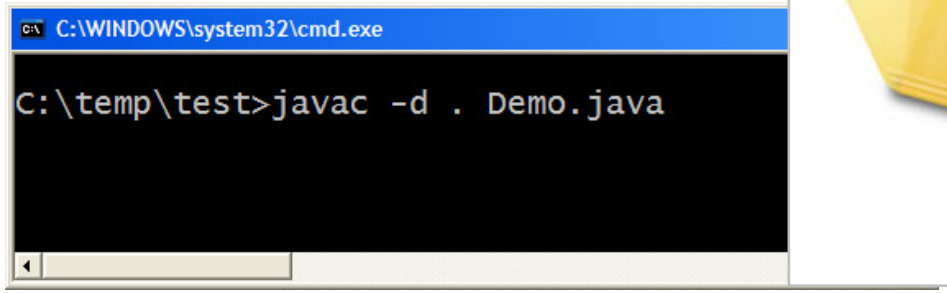
becomes

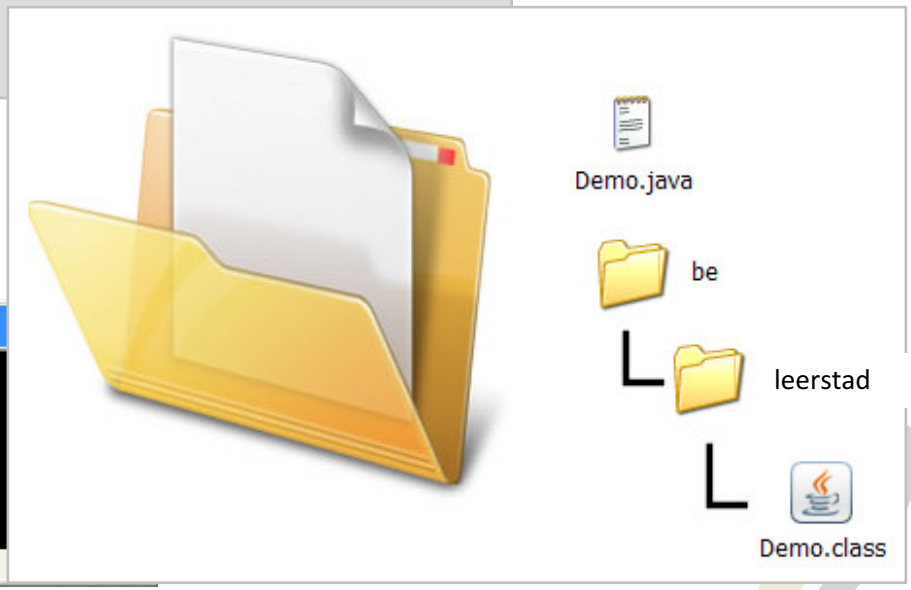
```
import static java.lang.Math.*;

public class Demo {
  public static void main(String[] args) {
    ...
    surface = 2 * radius * PI;
    volume = pow(radius, 3.0) * PI * 4 / 3;
    ....
  }
}
```

Packages

- Managing Source and Class Files

```
package be.leerstad;

public class Demo {
  public static void main(String[] args) {
    System.out.println("Java is good for you!");
  }
}
```





Packages

- Running a Demo class from the be.leerstad package? Use the FQCN – always!

```
C:\Windows\system32\cmd.exe
E:\Java\courses\1. Java Basics\code>java be.hardeel.Demo
Java is good for you!

E:\Java\courses\1. Java Basics\code>
```

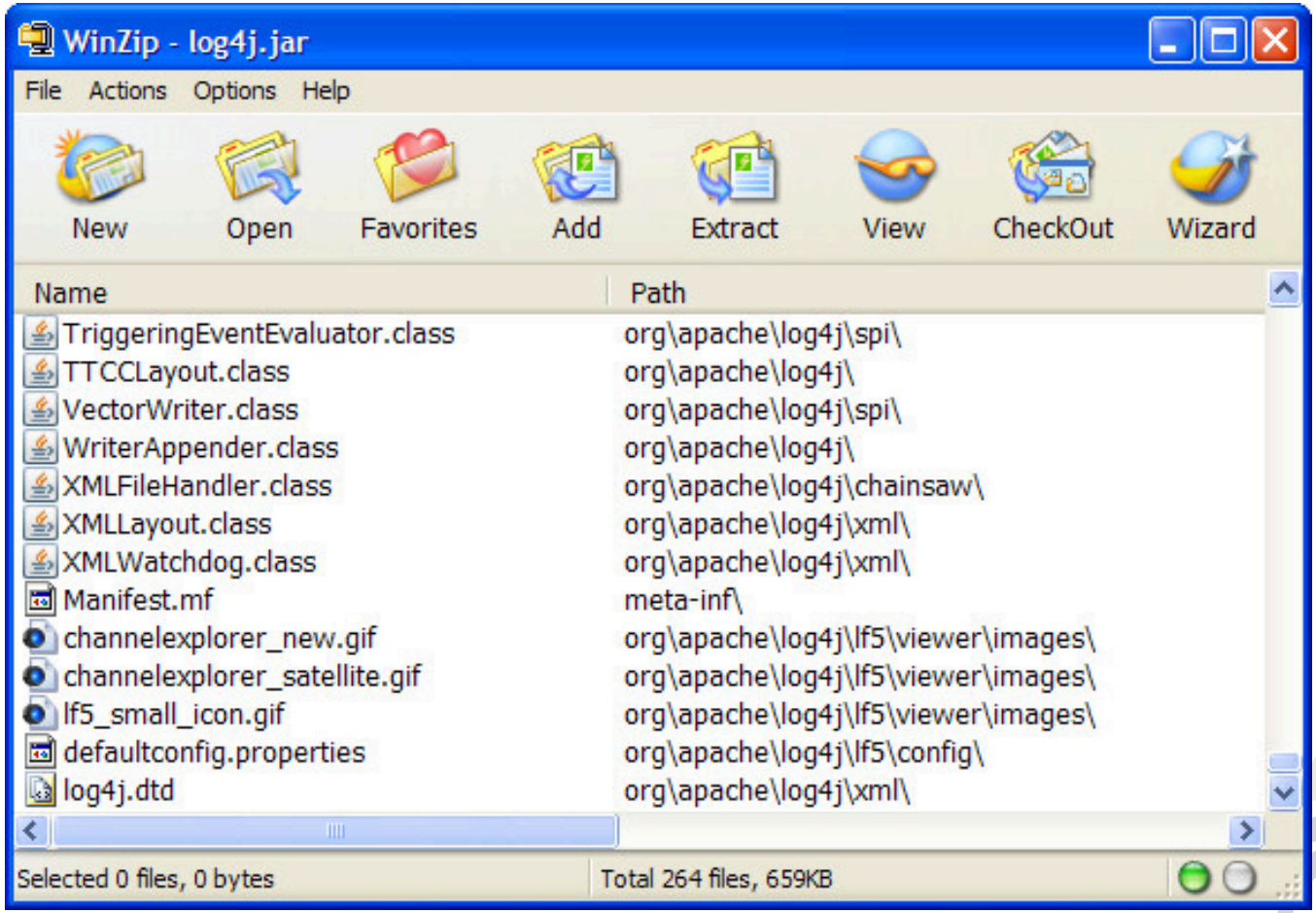
- You always must invoke from the root of the hierarchy, or by using the `-classpath` or `-cp` JVM options
- Or by setting the OS `CLASSPATH` variable

Grouping Files Logically

- Packages allow to logically group Java types
- Typical file extension: *.jar*
- Java Archives allow to group any kind of file required by an application:
  - class files (.class)
  - properties files and i18n resources (.properties)
  - images, sounds, ...

Grouping Files Logically

- Example: log4j.jar (logging library)



Grouping Files Logically

- Types of archives
  - classical : .jar
  - web archives: .war
  - enterprise: .ear
  - ...

Archives and JVM Parameters

- Frequently used option: `-cp`
  - Starting a program that uses a class from a third-party library archived in log4j.jar:

```
C:\>java -cp C:\Peter\courses\Java\IFA\code;
->           C:\java\lib\logging-log4j-1.2.14\dist\lib\log4j.jar
->  ClassPathDemo

15:35:44,000  INFO ClassPathDemo:6 - *** Starting a demo ***
```

- We start the application from C:\
- We indicate the JVM:
  - where to find the main class
  - where to find a dependent library (a .jar file)

Exercise

Compiling and using classes in packages

Summary

- Java provides you a construct to logically group classes together.
- There are some naming conventions and restrictions.
- Distributing libraries and/or project occurs via archives: physical group of types and related resources (kind of ZIP file)

Questions ??