JAVA PROGRAMMING BASICS

Module 2: Java Object-oriented Programming

Training program

Module contents

- Packages
  - The Package
  - Package import
  - Adding class to Package
  - Static import
  - Package organization
  - The jar utility
  - Executable jars

Module contents
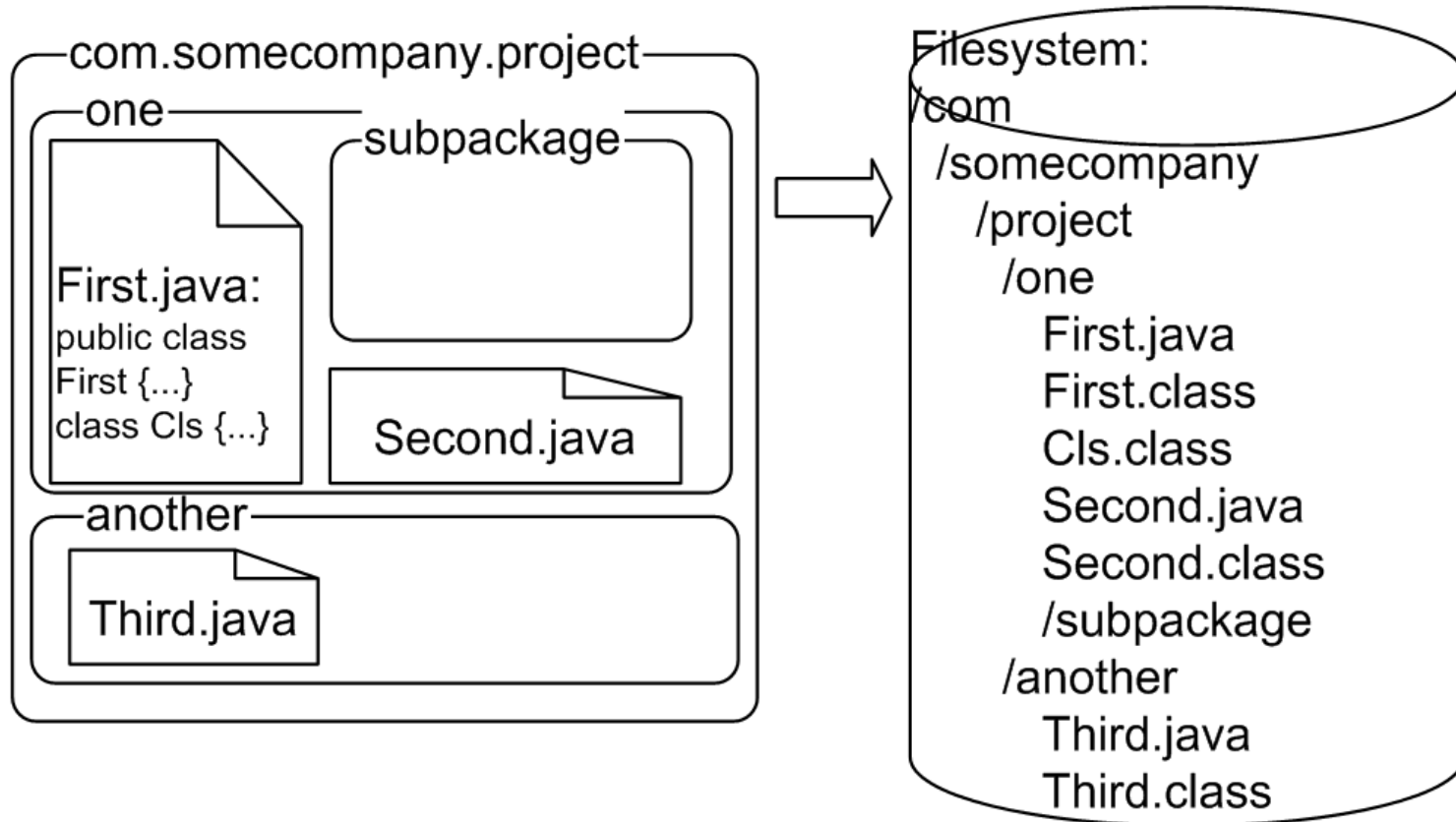
- Packages
  - The Package
  - Package import
  - Adding class to Package
  - Static import
  - Package organization
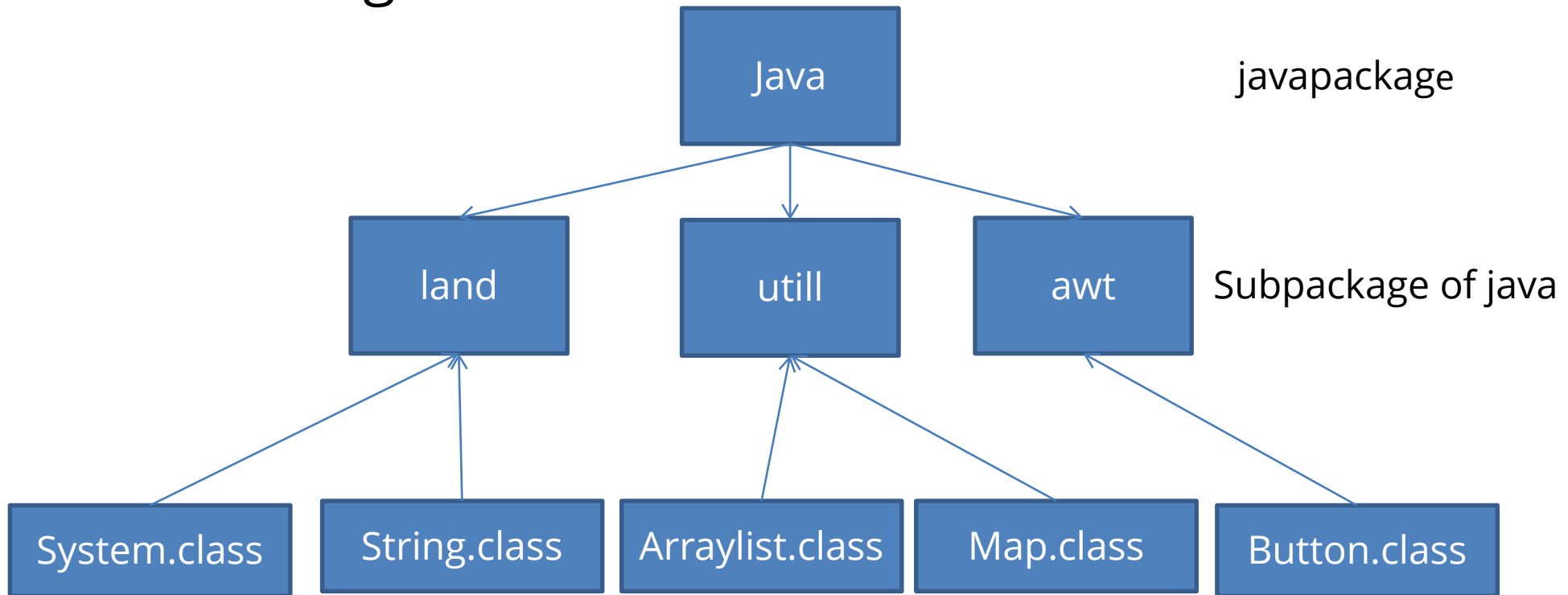 - The jar utility
  - Executable jars

The Package 1/3

- A package is a collection of related classes and interfaces providing namespace management
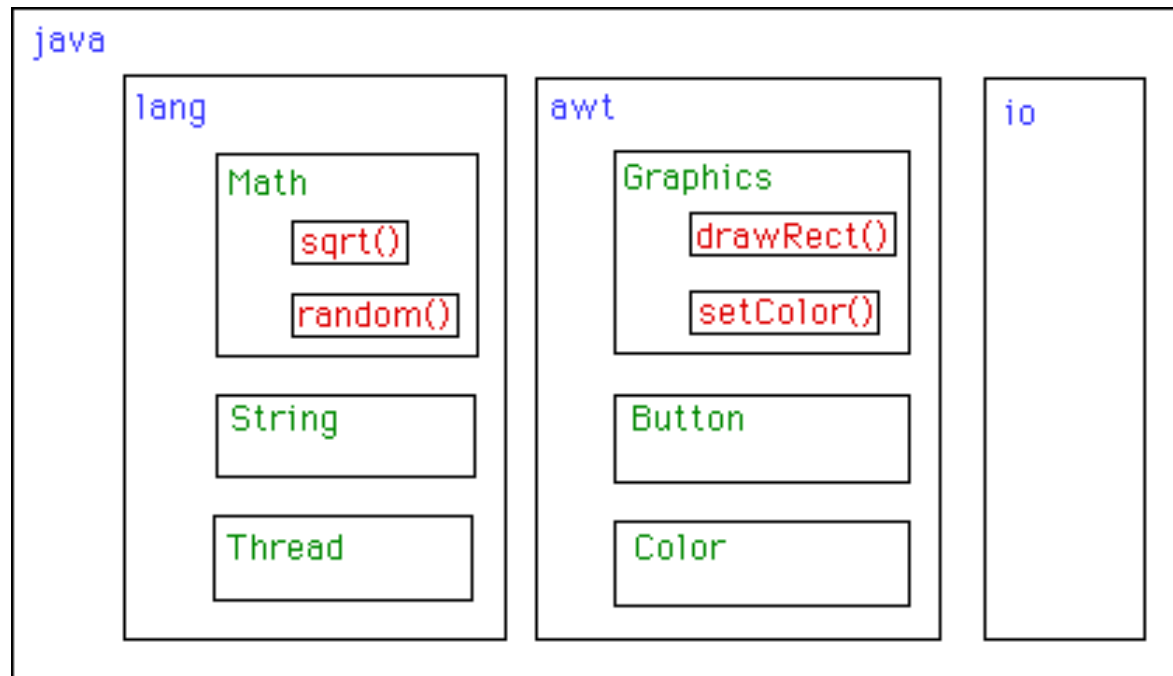
The Package 2/3

- Packages support hierarchical organization, and are used to organize large programs into logical and manageable units

```
                        ┌──────────┐
                        │   Java   │           javapackage
                        └──────────┘
              ┌──────────────┼──────────────┐
         ┌─────────┐    ┌─────────┐    ┌─────────┐
         │  land   │    │  utill  │    │   awt   │    Subpackage of java
         └─────────┘    └─────────┘    └─────────┘
```

| System.class | String.class | Arraylist.class | Map.class | Button.class |

The Package 3/3

- Java packages are namespaces. They allow programmers to create small private areas in which to declare classes.

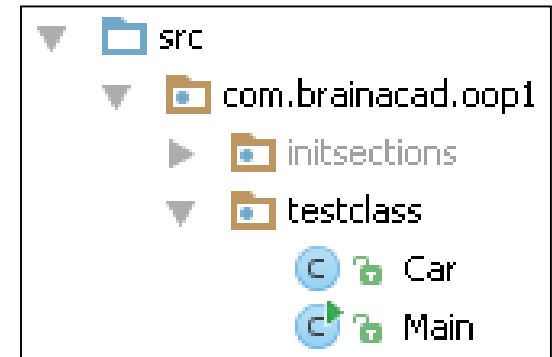Subroutines nested in classes nested in two layers of packages.
The full name of sqrt() is java.lang.Math.sqrt()

Module contents

- Packages
  - The Package
  - **Package import**
  - Adding class to Package
  - Static import
  - Package organization
  - The jar utility
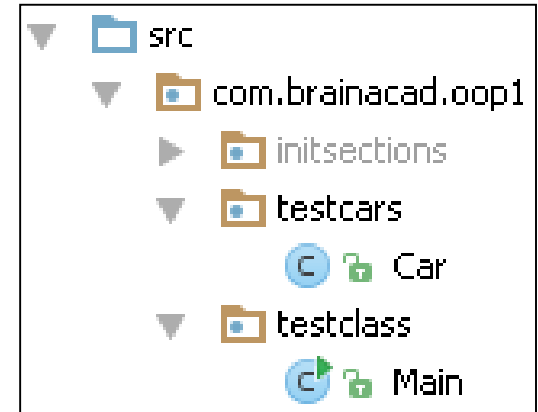 - Executable jars

Package import 1/4

- If class Main and Car are located in the same Java package:

1. **package** com.brainacad.oop1.testclass;
2. **public class** Main {
3.   **public static void** main(String[] arg) {
4.   Car myCar1 = **new** Car();
5.   }
6. }

Package import 2/4

- If class Main and Car are located in the same Java package:

```
1.  package com.brainacad.oop1.testclass;
2.  import com.brainacad.oop1.testcars.Car;
3.  public class Main {
4.      public static void main(String[] arg) {
5.          Car myCar1 = new Car();
6.      }
7.  }
```

Import
Car class

Package import 3/4

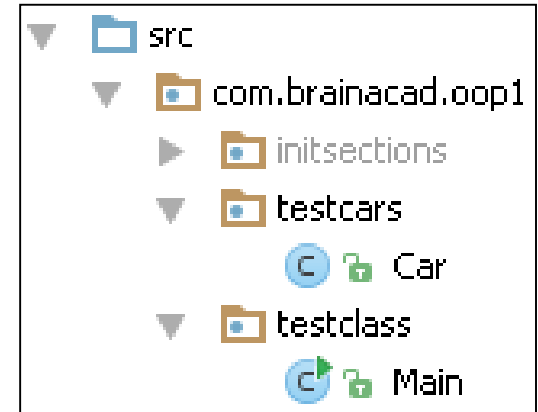- If class Main and Car are located
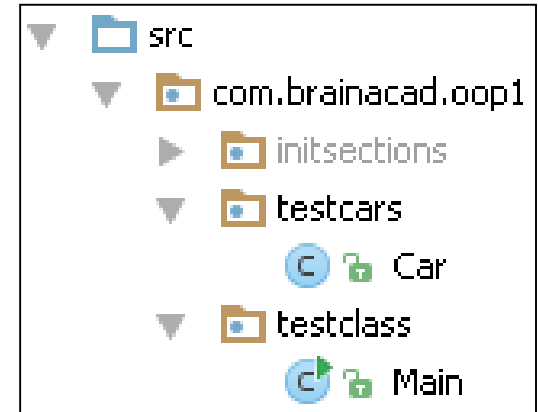 in the same Java package:

```
1.  package com.brainacad.oop1.testclass;
2.  import com.brainacad.oop1.testcars.*;
3.  public class Main {
4.      public static void main(String[] arg) {
5.          Car myCar1 = new Car();
6.      }
7.  }
```

Import
all classes

Package import 4/4

- If class Main and Car are located in the different Java packages:

1. **package** com.brainacad.oop1.testclass;
2. **public class** Main {
3.   **public static void** main(String[] arg) {
4.   com.brainacad.oop1.testcars.Car myCar1 =
5.   **new** com.brainacad.oop1.testcars.Car();
6.   }
7. }

Fully Qualified Class Name

Module contents

- Packages
  - The Package
  - Package import
  - Adding class to Package
  - Static import
  - Package organization
  - The jar utility
  - Executable jars
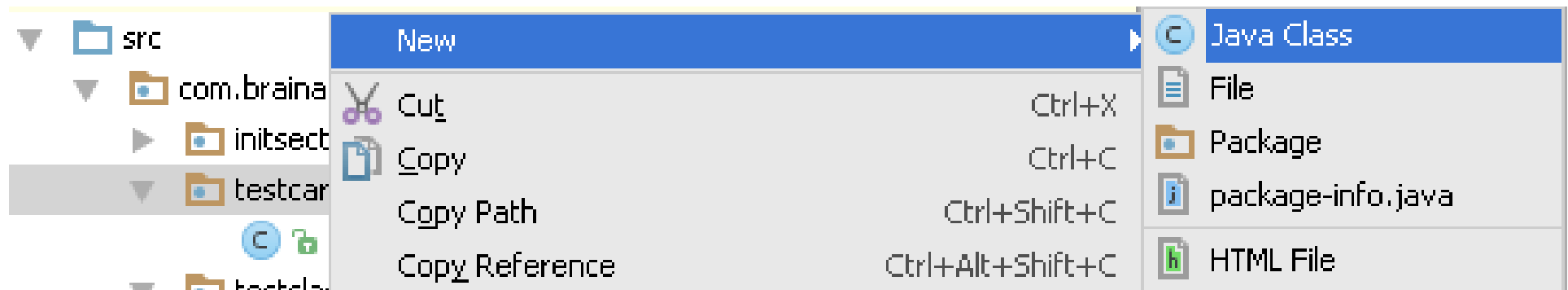
Adding class to Package

- To create a new class in package

Module contents

- Packages
  - The Package
  - Package import
  - Adding class to Package
  - Static import
  - Package organization
  - The jar utility
  - Executable jars

Static import 1/3

```java
1. package com.brainacad.oop1.testclass;
2. public class Main {
3.    public static void main(String[] arg) {
4.       double theta = 1;
5.       double r = Math.cos(Math.PI * theta);
6.    }
7. }
```

Static import 2/3

- Static imports allow the static items of one class to be referenced in another without qualification.

```java
1.  package com.brainacad.oop1.testclass;
2.  import static java.lang.Math.PI;
3.  import static java.lang.Math.cos;
4.  public class Main {
5.      public static void main(String[] arg) {
6.          double theta = 1;
7.          double r = cos(PI * theta);
8.      }
9.  }
```

Static import 3/3

- Static imports allow the static items of one class to be referenced in another without qualification.

```
1.  package com.brainacad.oop1.testclass;
2.  import static java.lang.Math.*;
3.  public class Main {
4.      public static void main(String[] arg) {
5.          double theta = 1;
6.          double r = cos(PI * theta);
7.      }
8.  }
```

Module contents

- Packages
  - The Package
  - Package import
  - Adding class to Package
  - Static import
  - **Package organization**
  - The jar utility
  - Executable jars
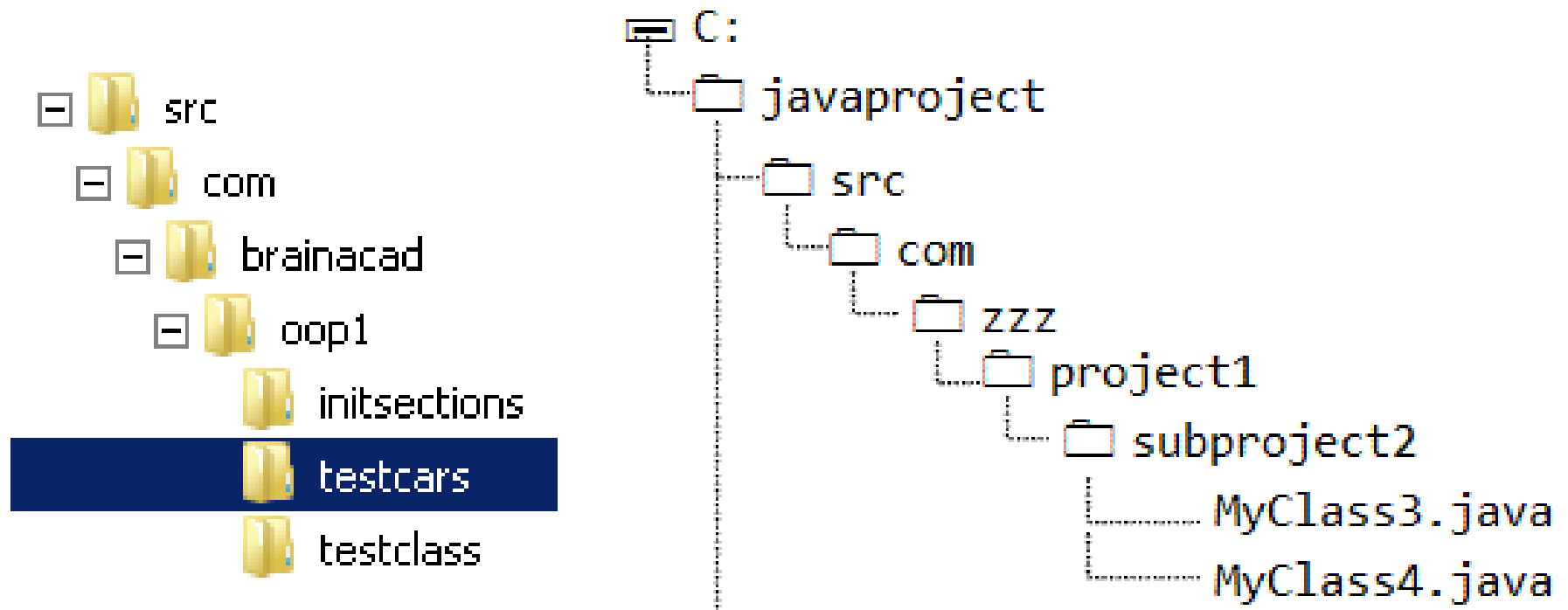
Package organization 1/5

- **Package Naming Conventions**
- To prevent package name collisions, the convention is for organizations to use their reversed Internet domain names to begin their package names.
- For example, *com.brainacad*

Package organization 2/5

- **Package Naming Conventions**
- If the Internet domain name contains an invalid character, such as a hyphen, the convention is to replace the invalid character with an underscore.
- If a domain name component starts with a digit or consists of a reserved Java keyword, the convention is to add an underscore to the component name.
- For example, *com.brainacad._1java*

Package organization 3/5

- The sub-directory structure corresponding to the package name for the classes will be created automatically if it does not already exist.

Package organization 4/5

- **Package By Feature**

- *Package-by-feature* uses packages to reflect the feature set. It tries to place all items related to a single feature (and *only* that feature) into a single directory/package. This results in packages with high cohesion and high modularity, and with minimal coupling between packages.

- *com.mycompany.report*

- *com. mycompany.security*

- *com. mycompany.util*

Package organization 5/5

- **Package By Layer**
- The competing *package-by-layer* style is different. In package-by-layer, the highest level packages reflect the various application "layers", instead of features, as in:
- *com. mycompany.action*
- *com.mycompany.model*
- *com. mycompany.dao*

Module contents

- Packages
  - The Package
  - Package import
  - Adding class to Package
  - Static import
  - Package organization
  - **The jar utility**
  - Executable jars

The jar utility 1/3

- jar-The Java Archive Tool

- jar combines multiple files into a single JAR archive file.

- Java Archive (JAR) is a platform-independent file format that allow you to compress and bundle multiple files associated with a Java application, into a single file. JAR is based on the popular ZIP algorithm, and mimic the Unix's tar file format

The jar utility 2/3

- **Creating JAR File**

1. Via the command-line jar tool: JDK provides a command-line tool called "jar.exe"

2. Via the "export" option of IDE such as IntellijIDEA or Eclipse, in practice.

The jar utility 3/3

- **Create JAR File using Command-Line "jar" Tool**
- To create a JAR file using the jar tool, issue the jar command (on CMD shell) with 'c' option:

- > jar cvf *jarFile inputFileDir1 inputFileDir2 ...*

- For example, create a JAR file named MyJar.jar by entering the following command:
- >jar cvf MyJar.jar MyPackage/*.class

Module contents

- Packages
  - The Package
  - Package import
  - Adding class to Package
  - Static import
  - Package organization
  - The jar utility
  - Executable jars

Executable jars 1/4

- JAR functions, such as main-class specification, are supported though a file called *manifest*.
- The manifest is a special file, called "MANIFEST.MF" under the "META-INF" sub-directory, that contains information about the files contained in a JAR file

Executable jars 2/4

- In the jar "META-INF\MANIFEST.MF" file can specifies the main-class that contains the entry main() method for launching the application:

1. Manifest-Version: 1.0
2. Created-By: 1.7.0_06 (Oracle Corporation)
3. Main-Class: MyPackage.Main
4.

Executable jars 3/4

- To run the application directly from JAR file, invoke the JRE with option "-jar":

- > **java -jar Hello.jar**

Executable jars 4/4

- **Setting an Entry Point with the JAR Tool**

- >jar cfe app.jar MyApp Main.class