UEF4.3 Object Oriented Programming

Mrs Sadeg and Mrs Bousbia
s_sadeg@esi.dz, n_bousbia@esi.dz
Ecole Nationale Supérieure d'Informatique
(ESI

Packages

Packages

- A package is a group of classes under a common name, facilitating code organization and modularity.
- This concept is similar to namespaces in other programming languages and helps prevent name conflicts between classes.
- By structuring code into packages, the development of large-scale applications becomes easier to manage and maintain.

Packages

Example

The Shape, Circle, Rectangle, Square, Point, ColPoint classes, and the Colorable interface can be bundled into a package named graphics. Doing so provides the following benefits:

- It clearly establishes that these types are related
- It defines a specific location for types that provide graphics-related functionalities.
- It prevents naming conflicts with types from other packages.
- It allows types within the package to have unrestricted access to one another while restricting access for types outside the package.

Creating a Package

To create a package, we put a package statement with that name at the top of every source file that contains the types that belong to the same package.

```
package graphics; //in the Colorable.java file
public interface Colorable {. . .}

package graphics; //in the Shape.java file
public abstract class Shape {. . .}

package graphics; //in the Circle.java file
public class Circle extends Shape  implements Colorable {
. . .}
```

Note:
If we do not use a package statement, your type ends up in an unnamed package

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, we must do one of the following:

1. Refer to the member by its fully qualified name
2. Import the package member
3. Import the member's entire package

Using package members

1. Referring to the member by its fully qualified name

- if we try to use a member from a different package and that package has not been imported, we must use the member's fully qualified name, which includes the package name.
- The fully qualified name of Circle class declared in the graphics package in the previous example is `graphics.Circle`
- The qualified name can be used name to create an instance

```
graphics.Circle c = new graphics.Circle();
```

- When a name is used repeatedly, typing it multiple times becomes tedious and reduces code readability. Instead, we use the second option.

Using package members

2. Importing the package member

- The import statement allows to import the name of one or more classes. The name must be fully qualified.

```
import graphics.Circle;
```

- Now, the Circle class can be used without mentioning the package name.

```
Circle c= new Circle();
```

- If multiple classes from the same package are involved, the third option is preferable.

Using package members

3. Importing the member's entire package

We can import all the types contained in a particular package, by use the import statement with the asterisk (*) wildcard character.

```
import graphics.*;
```

Now we can refer to any class or interface in the graphics package by its simple name.

```
Circle c = new Circle();
Rectangle r = new Rectangle();
```

Using package members

Remarks

- If two packages containing classes with the same name are imported, it results in an error.
- Referencing a package without using all its classes does not increase compilation time or the size of the generated .class files.
- The purpose of filtering is to limit ambiguities in case of similar class names across different imported packages.
- Most environments impose constraints on the location of files corresponding to a package.A package named X.Y.Z is always entirely located in a subdirectory called X/Y/Z. However, the package X.Y.U, which is in the subdirectory X/Y/U, may be in a different directory from X.Y.Z.With the SDK (or JDK), package lookup is performed in the directories declared in the CLASSPATH environment variable.

Standard packages in java

- The standard classes supplied with java are structured in packages such as java.awt, java.awt.event,javax.swing…
- There's a special package called **java.lang** which is automatically imported by the compiler and allows you to use standard classes such as Math, System, Float or Integer, Double, Object…

Packages and access rights

1. Access Rights for Classes

- Each class has an access right (or access modifier) that determines which other classes can use it. The access right is specified by the presence or absence of the public keyword:
  - With the public keyword, the class is accessible to all other classes (potentially using an import statement).
  - Without the public keyword, the class is only accessible to classes within the same package.
- As long as we are working with the default package, the public keyword is optional—except for the class containing the main method, which must be public for the JVM to access it.

Packages and access rights

2. Access rights for class members

We have already seen that a member (field or method) can have one of the following access modifiers:

- **public**: The member is accessible from outside the class.
- **private**: The member is only accessible within the class's methods.
- **protected**: The member is accessible by derived classes.
- There is also a fourth possibility: the absence of an access modifier. In this case, access is restricted to classes within the same package. This access level is known as **package-private** (or **default access**).

```
package P1;
public class A // accessible everywhere
{…
void f1() {….}
public void f2(); {…}
}
```

```
package P2;
Import P1.A;
class  B// accessible only in p2
{public void g(){ A a = new A();
a.f1();     // Error
a.f2();    // OK }
}
```

Access modifiers for members and inner classes

Modifier	Meaning for a Class or an Interface
public	Always accessible
(no modifier) (package-private)	Accessible only from classes within the same package

Access modifiers for members and inner classes

Modifier	Meaning for a Class Member (Field or Method) or an Inner Class
public	Accessible everywhere the class itself is accessible
(no modifier) (package-private)	Accessible from all classes in the same package (regardless of the access level of the class, as long as it has at least package-level access)
protected	Accessible from all classes in the same package or from derived classes
private	Accessible only within the class where the member or inner class is declared