Java training

Packages, imports, access modifiers

Session overview

- Packages
- Imports
- Access modifiers

Packages

- **Package** - namespace that organizes a set of related entities
  - Classes, interfaces and enums
- Conceptually - similar to the folders on a computer
  - Keeping documents in one folder, photos in another one, music in another one
- Java projects can be composed of hundreds / thousands of classes
  - → Separation of Concerns
- JDK built-in packages → comprehensive suite of predefined packages:
  - `java.lang` - contains the core classes: `Object`, `Integer`, `String`, etc
  - `java.util` - contains utilities classes; among them - the collections hierarchy
  - ...

Creating and using packages

- Creating a package → done via the IDE: ‘Create package’
  - A package = a file-system folder
- After creating a package, when creating entities in that package, the first line will be the package name:

```
package com.example.product;
```

- The rest of the file content:
  - After the package - an empty line
  - The created entity - class, interface or enum (further presented)

Packages - naming conventions

- Written in lowercase → avoid conflict with classes or interfaces names
- Companies use their reversed domain name for their package names:
  - `com.example.service` → a package named `service` created by someone from the example.com company
- Name collisions in a company → handled by conventions (within the company)
  - Example: including the region or the project name after the company name:
    - `com.example.region.service`

Packages - naming conventions

Adding _ (underscore) - if the internet domain name is not a valid package name:

- Contains a hyphen or other special characters
- The package name begins with a digit or other illegal character
- The package name contains a reserved Java keyword, such as "int"

Examples:

after-sunset.example.org	org.example.after_sunset
example.int	int_.example
123forms.com	com._123forms

Importing (using) classes from a package

- Using a class from an existing package - needs to be imported

- The import statements *must* be declared immediately after the package name
- **Hands-on** → creating a class in a `com.example` package, importing a class

Importing an entire package

- If a class uses a lot of classes from a package, the imports list can become too long → the `import <package-name>.*` can be used

```
import java.util.stream.*
```

- Usually not recommended - some editors automatically import all the classes when the import number exceeds a threshold
- Apparent hierarchy - packages are *not* hierarchical - when a package is imported, the sub-packages are *not* imported

Access modifiers

- There are four types of Java access modifiers - in ascending access order:
  - `private`
  - `default`
  - `protected`
  - `public`
- `private` - the members are accessible only within the class:

```
public class Store {
    private Set<Section> sections;
}
```

→ the sections field is accessible only in the current class

Using private constructors

- Using a `private` constructor:

```
public class Tablet {
    private Tablet {}; → the class cannot be instantiated
}
```

- Usefulness:
  - Instantiating objects only inside the class → factory methods (further presented)
  - If a class needs to be made non instantiable → singletons (further presented)

Hands-on example →

`default` access modifier

- If no access modifier is specified → `default` is implied
- Result: the class will be accessible only within it's package

```
package com.example.domain;

class Tablet {          → the class will be accessible only within the package
    private String name;
}
```

Hands-on example →

`protected` access modifier

- `protected` → the entity (class / variable) is accessible through inheritance:
  - Within the package
  - Outside the package - from extending classes / interfaces
- The `protected` access modifier can be applied to:
  - Members / variables
  - Methods & constructors
- Cannot be applied on the class

```
class Tablet {
    protected String name;
}
```

→ accessible within and outside the package, via inheritance

`public` access modifier

- Accessible from everywhere → the widest access scope

```
public class Tablet {  → accessible & usable from everywhere
    private String name;
}
```

- Most JDK API classes have `public` scope

Access modifiers wrap-up

Access modifier	Within the class	Within the package	Outside the package by subclass	Outside the package
`private`	✓	×	×	×
`default`	✓	✓	×	×
`protected`	✓	✓	✓	×
`public`	✓	✓	✓	✓

Other types of modifiers

- The others (non-access) modifiers in Java:
  - `static` (already presented)
  - `final` (already presented)
  - `abstract` (will be presented soon)
  - `synchronized` (will be presented in the Threads topic)
  - `native`
  - `volatile`
  - `transient` (will be presented in the serialization topics)

Q & A session

1. You ask, I answer
2. I ask, you answer
    a. What are packages?
    b. How can we use classes from other classes?
    c. What are the Java access modifiers?