3. Software components

Slides:

- *07 Software components*
- *08 JavaBeans*
- *09 Reflection in Java*
- *10 Microsoft software components*
- *11 Frameworks Inversion of Control*
- *12 Designing Software Frameworks*

Why components?

Components allows to re-use software, with the following benefits.

- Reduce the costs of development.
- It's cheaper to buy a framework and implement a software overriding its default behaviour.
- It's easier create a software by composing code.
- Produce more reliable software: components are widely widespread and therefore more exempt to errors.
- Components are necessary to build systems composed of independent parts, such as microservices architectures.

Desiderata for sw components

1. Modular (packaged code)
    - compatible (simple interface)
    - reusable (its main aspect)
    - extendible (through inheritance and overriding)
2. Reliable
    - correctness (respect specification)
    - robustness (a.k.a. fault-tolerance: able to work in abnormal situations)
3. Efficient
4. Portable to different platforms
5. Timely: released when or before needed

Basic concepts

- Component interface: describe the implemented operations that other components can use.
- Composition mechanism: how to compose different component to implement some task.
- Component platform: platform for development and execution of components.

Modules vs Components

Modules are available only for that specific programming language that you are using, and doesn't allow to cooperate with code written in different programming languages.

Modules are usually compiled separately, and the implementation is not known to the module user.

Components has several concept already present in modules, but instead of being part of a program, as for the module, are part of a system.

Components can be anything and contain anything.

Component specification

A *Component specification* describes, through a set of *component interfaces*, the behaviour of a set of *component objects*.

The component specification is realized as a *component implementation*, which is independently deployable as *installed component*. An instance of installed components is called *component object*.

A *component composition* is the process of assembling components to form an application or a larger component.

Java Beans

A software component model for Java.
A class to be considered a Java Bean must support:

- Properties, e.g. using getter and/or setter. If setted at design time we have customization, at runtime we have application logic.
- Events, allowing other classes to register a callback (listener) for property change (bound properties: `PropertyChangeListener`; constrained properties: `PropertyChangeEvent`, `VetoableChangeListener`) following the publish-subscribe pattern.
- Customization, using protected properties and facilitating inheritance.
- Persistence, using serialization.
- Introspection, based on reflection, naming conventions and design patterns.

Furthermore, must:

- Have a public constructor with 0 arguments.
- Be in a JAR file containing a manifest file.

Reflection in Java

The ability of a program to manipulate itself as data.

- Introspection: can observe its internal state (read-only)
- Intercession: can modify itself

This abilities requires reification, a mechanism to encode the execution state as data.

- Structural reflection: complete reification of both the program in execution and its abstract data types.
- Behavioral reflection: complete reification of semantic and implementation of the program and of the runtime system.

Reflection purpose

- Class browser: enumerate the members of classes
- Visual Developer Environment (IDE): help developers to write correct code
- Debugger
- Test tools

Drawbacks of reflection

- Performance overhead: is not possible to apply the JVM optimization to dynamically loaded classes.
- Security restriction: is not always possible to run program using reflection since is considered not secure.
- Exposure of Internals: with reflection is possible to access to private fields, breaking the abstraction principles.

Java reflection

For every type the JVM keeps an object of class `java.lang.Class` that is the entry point for reflection. This object can be obtained using `Object.getClass()` or `Class.forName(String className)`.

It contains:

- Class name and modifiers
- Superclass & Interfaces implemented
- Methods, fields, constructors, etc.

It is usually used to:

- Creating new objects of a type that is not known at compile type

- Accessing members that are not know at compile time

Example:

- A is a superclass;
- classes B and C extends A implements the same method `doSomething` with different parameter type;
- a method m accept objects of type A, that actually are object of type B or C;
- using reflection m can invoke `doSomething` on both B and C objects with the correct parameter type.

Some operations are forbidden by privacy rules

- Changing a final field
- Reading or writing a private field
- Invoking a private method

Microsoft Software Components

Distributed Component Technologies

The goal is the integration and interoperability of services for applications on various platform.

- Sun: JavaBeans, Enterprise JavaBeans, J2EE
- SOAP (using XML)
- Microsoft:
  - COM (Component Object Model): simply components.
  - DDE (Dynamic Data Exchange): lets different Windows programs share data through links. Limitation: data must be updated in the source and not via link, links would break data is moved around in the source file.
  - OLE (Object Linking and Embedding): the same ad DDE with embedded data (snapshot copy).
  - ActiveX: the Microsoft equivalents to Java applets. It's affected by security issues because ActiveX controls have full file access.

.NET Framework

Consists of:

- Common Language Specification (CLS): guidelines that languages should follow to interoperate with .NET languages. It does also type-checking.
- Framework Base Class Libraries (BCL): OO prepackaged functionalities and libraries for .NET programs. Includes CLI and GUI libraries and ASP.NET tools (Web Forms and XML Web services)
- Common Language Runtime (CLR): language-neutral development and execution environment. Can run all .NET languages (36 atm) and uses a *Common Type System*. It is similar to the JVM: compiles in *Common*

Intermediate Language (CIL) executed by the *Common Language Interpreter* (CLI). It also manages the memory allocation, includes a Garbage Collector and perform JIT compilation.

- Common Type System: Defines a rich set of data types, based on an object-oriented model. Defines rules for interoperability between languages, scoping, visibility and inheritance.

.NET Assembly vs Modules

An Assembly is a .NET library. Has a .dll (local, not executable) or .exe (distributed, executable) extension and can be loaded dynamically.
Consists in up to 4 parts: manifest, metadata, CIL code and resources.
A Module has the same format of an Assembly, but doesn't include a manifest and is not dynamically loadable.

Delegates

Delegate is a type that represent a reference to a method: the method can be invoked by a delegate instance. Using delegates is possible to set up Event Handlers exploiting Observer or Publish/subscribe design pattern for events as for Java.

Inversion of Control

With Inversion of Control, the program flows is not dictated by the caller but by the framework.
The framework provides a default behavior that can be changed by extensibility: subclassing and selective overriding, implementing interfaces, registering for events.
Example: in GUI-based interaction, the GUI loop decide when to call the methods.
Hollywood Principle: “Don't call us, we'll call you”.

Loosely coupled systems

Loosely coupled systems provides many advantages, and in particular improve the extensibility, testability and reusability of the software.

Service Locator and Dependency Injection

Service Locator helps in avoiding strong coupling exploiting object factoring and generic interfaces.
Dependency Injection allows avoiding hard-coded dependencies (strong coupling) injecting them at runtime.
It is very convenient for mock testing and for fast replacing of dependency.
IoC Containers create objects, ensure that all dependencies are satisfied and then provide a lifecycle support.

Template methods

- concrete operations can be declared private ensuring that they are only called by the template method
- operations that must be overridden are declared protected abstract
- operations that may be overridden are declared protected