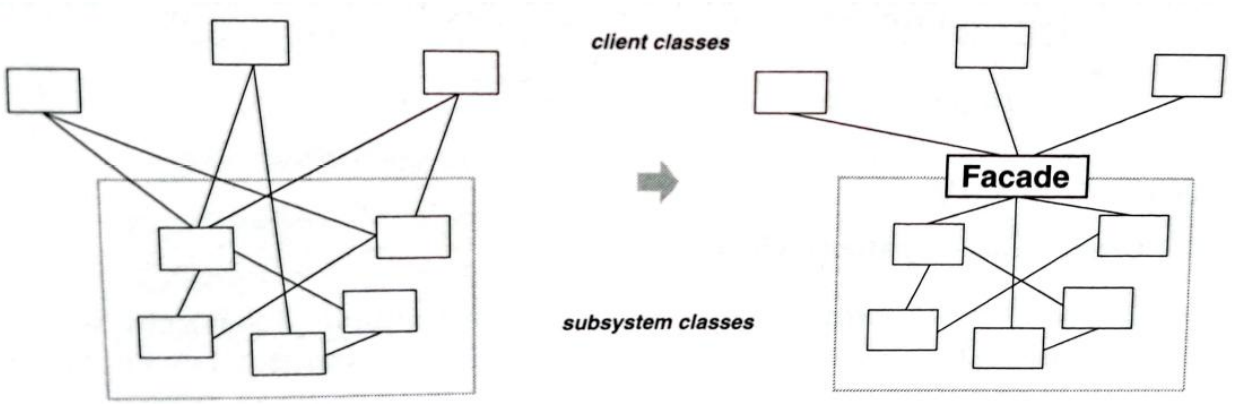
Facade

An object structural pattern

Learning goals

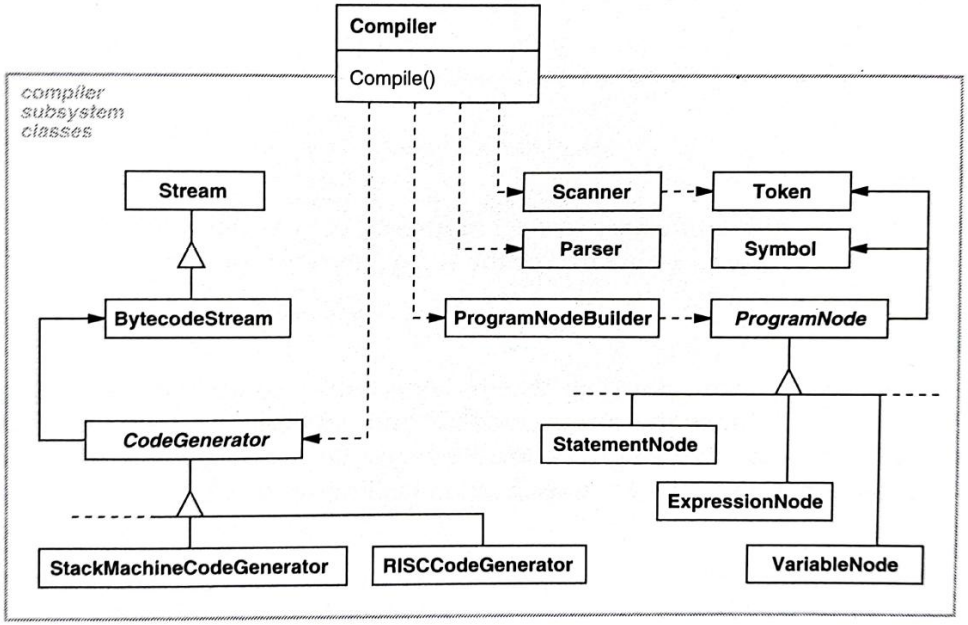
1. Learn the idea, structure, and Java implementation of the Facade design pattern.
2. Learn to apply the Facade DP in your own programming.

Idea of Facade



- The Facade DP provides **a simple interface** to complex subsystems.
- Goals:
  1. **Simplification**: Offers an easy and clear interface to the functionalities of complex systems.
  2. **Separation**: Decouples the system's interface from its backend operations
  3. **Safety**: Reduces direct interactions between subsystems, which minimizes the potential for errors.
  4. **Maintainability**: Provides a centralized control point from which many different parts can be managed.

Example: Compiler



- The compiler subsystem contains may classes that are involved in the compilation process.
- In most use cases, the client just needs to compile a given source code.
- Having to deal with several subsystem objects adds complexity.
- Adding a facade serves these use cases efficiently.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 257

General structure

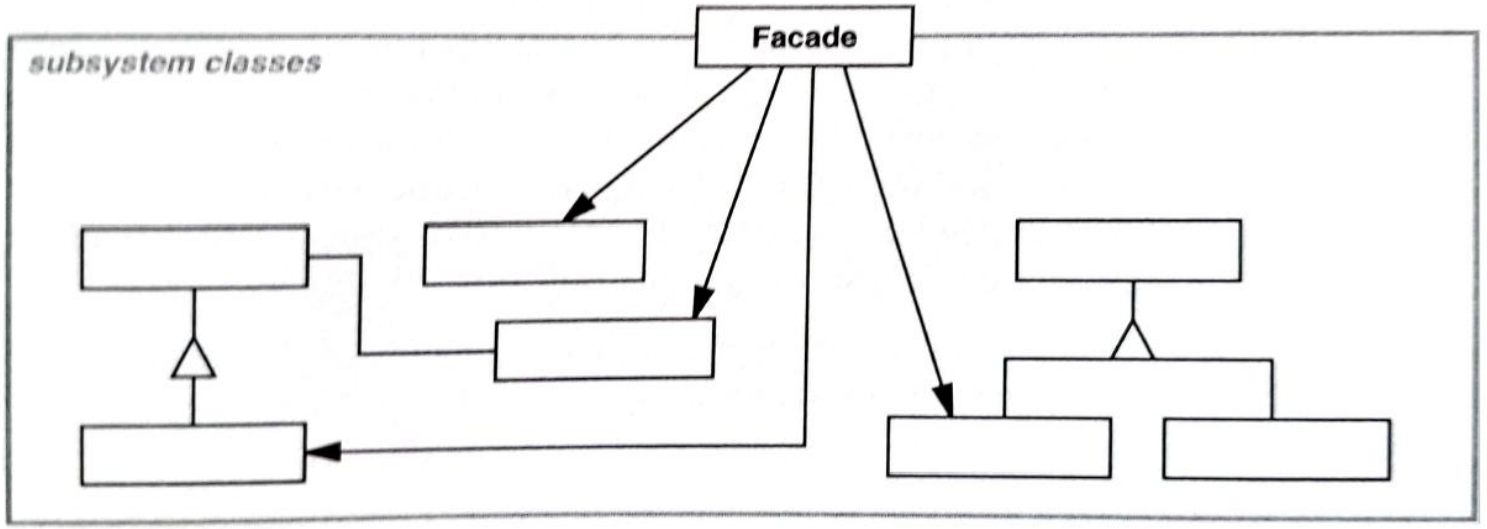


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 259

Roles

- **Facade** declares a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
- **Subsystem Classes** implement subsystem functionality.
  - They perform the actual work and are utilized by the Facade to fulfill client requests.
  - Subsystem classes are not aware of the facade; they operate within the system and work directly with the data.
- **Client** uses the Facade to access the subsystem.
  - The Client is not, however, restricted to the functionality provided by the facade.

Practical issues

- The Facade pattern centralizes complex interactions into a single interface.
  - Clients do not need to understand the inner workings of the subsystems.
- This poses certain risks:
  - The Facade becomes a critical part of the system architecture. If not implemented correctly, it can become a **single point of failure**.
  - Adding an additional layer with the Facade can introduce a slight **performance overhead**.
  - Clients might **underuse the subsystems** as they only interact with them through the Facade
  - **Unit testing** of the Facade may be complicated, at is highly dependent of many other objects (may require mocking, i.e. creating mock objects with a dedicated framework).