Object-Oriented Programming

Design Pattern Facade

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

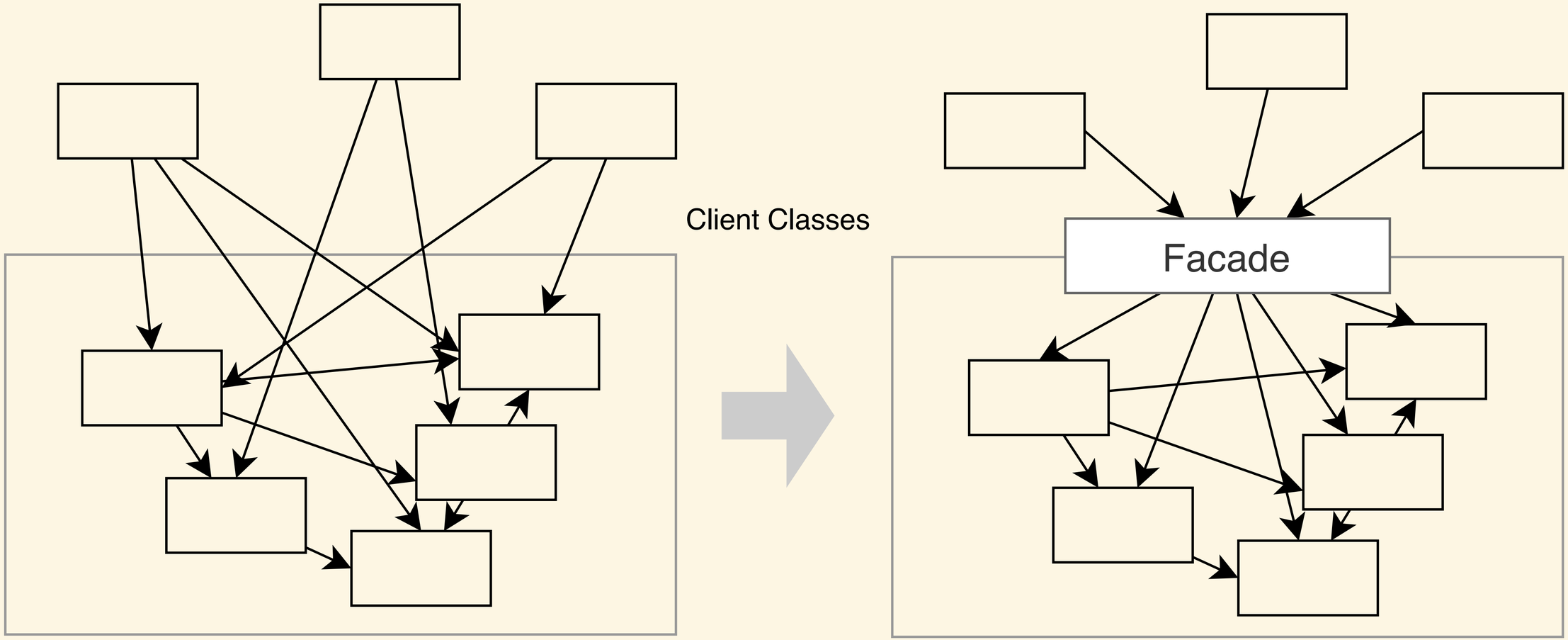
Facade

Provide a unified interface to a set of interfaces in a subsystem

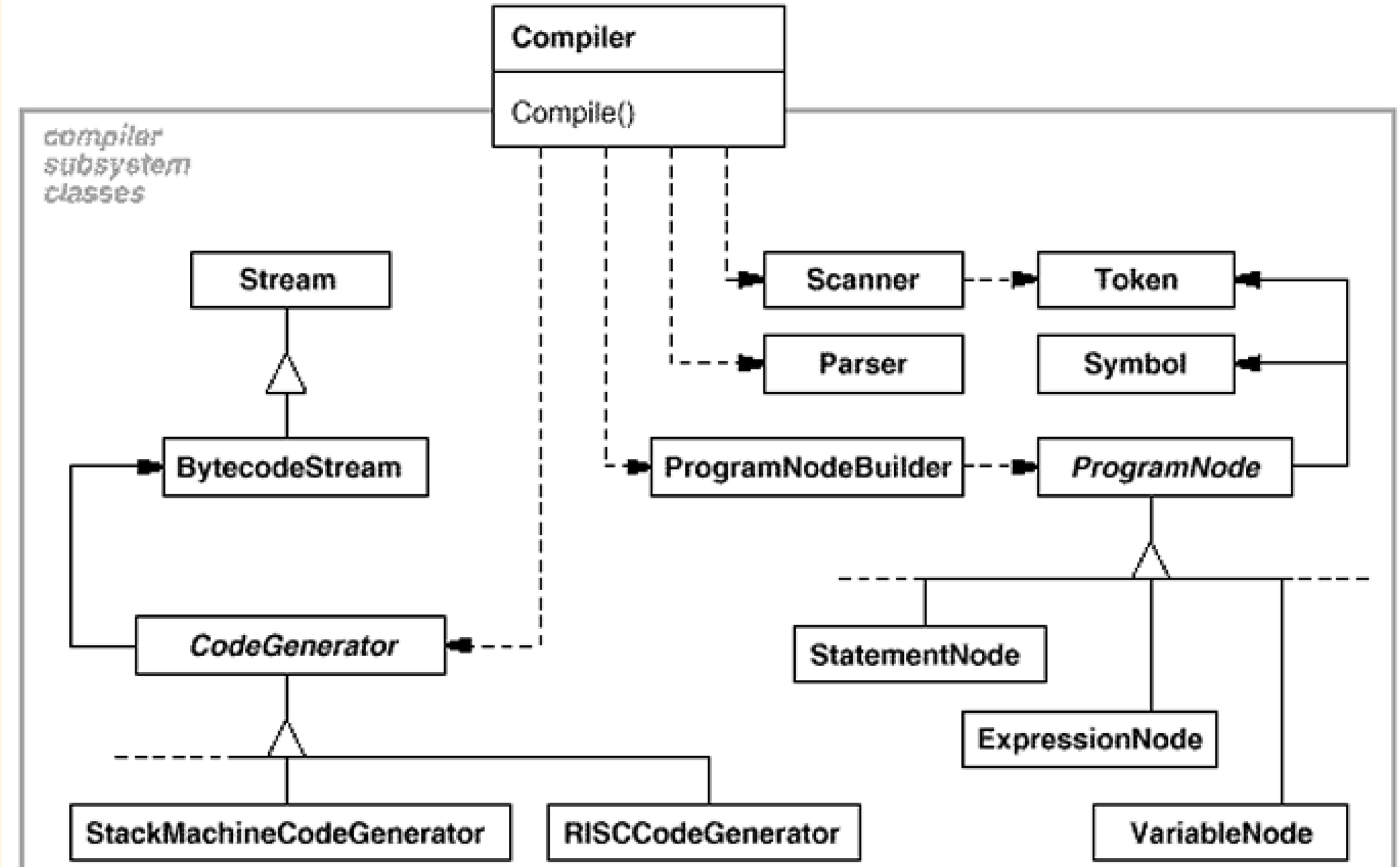
Facade defines a higher-level interface that makes the subsystem easier to use

- Structural Pattern

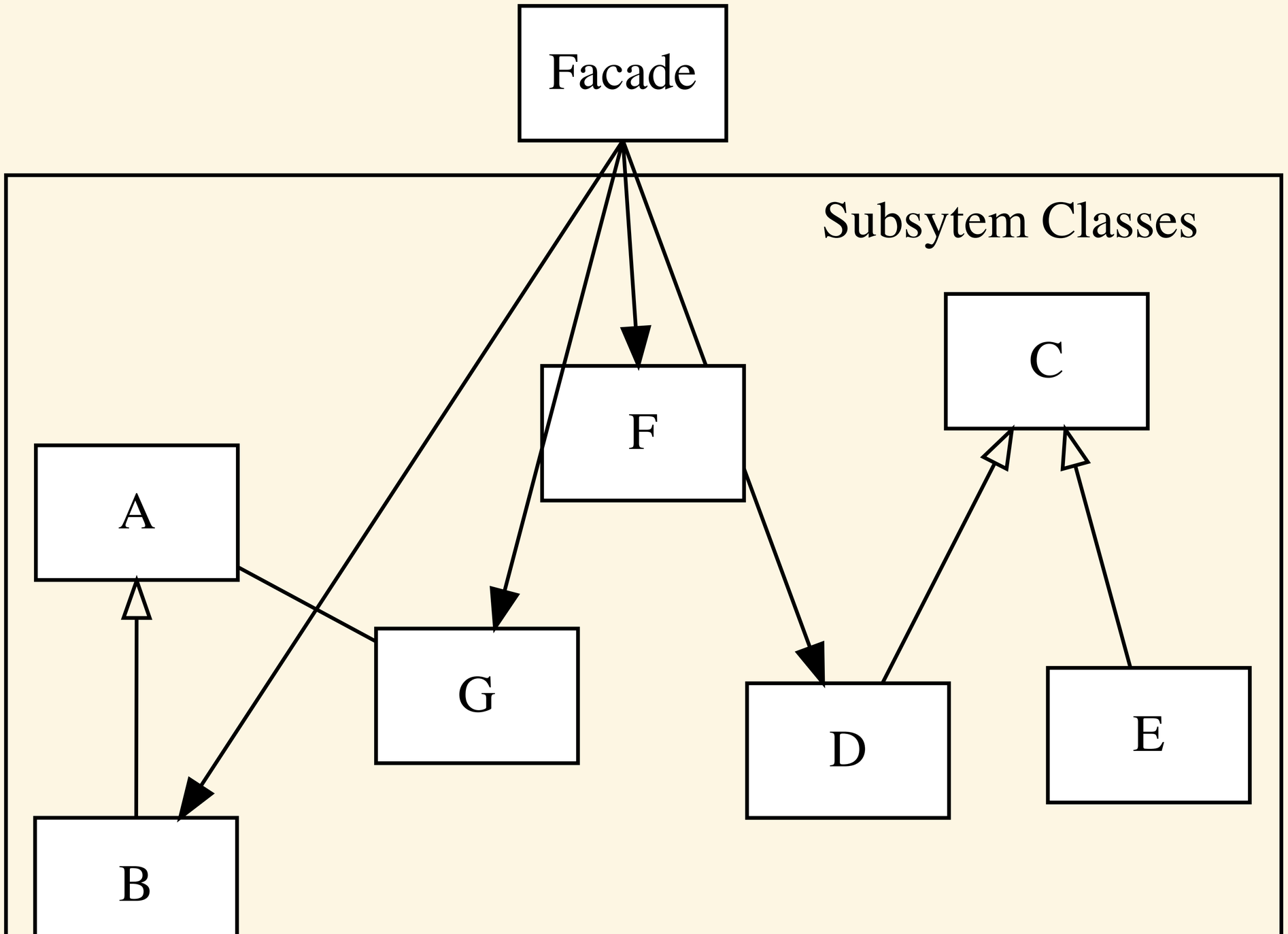
Facade: Motivation



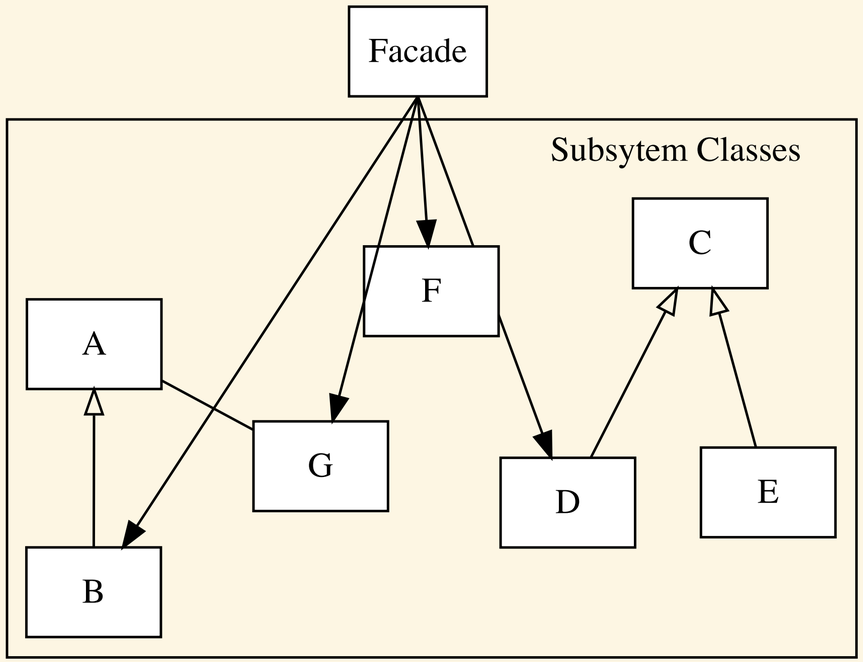
Facade: Motivation



Facade: Structure



Facade



- Converts client requests into target class requests

- Allows a current class to be used by client code expecting a different interface

- Often, the first step towards replacement/major changes to a class with the "wrong" interface

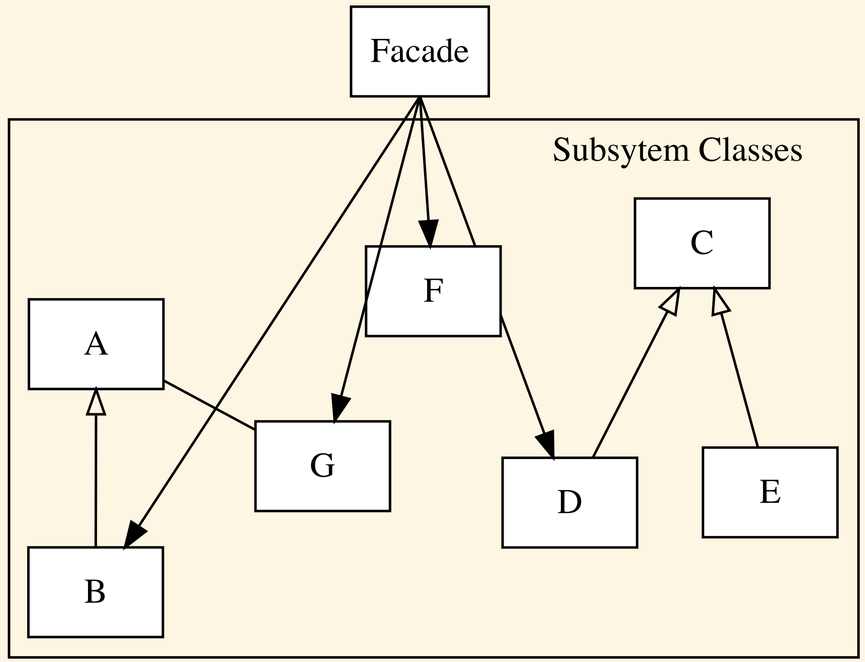
- May add functionality missing in the target class

Facade: Applicability I

Use the Facade pattern when you want to provide a simple interface to a complex subsystem

- Subsystems often get more complicated as they evolve
- Most patterns result in more and smaller classes
- Subsystems are more reusable and easier to customize but also harder to use for clients that don’t need customization
- Facade provides a simple default view of the subsystem that is good enough for most clients
- Only clients needing more customizability look beyond the facade for more functionality

Facade: Applicability II



- There are many dependencies between clients and the implementation classes of an abstraction

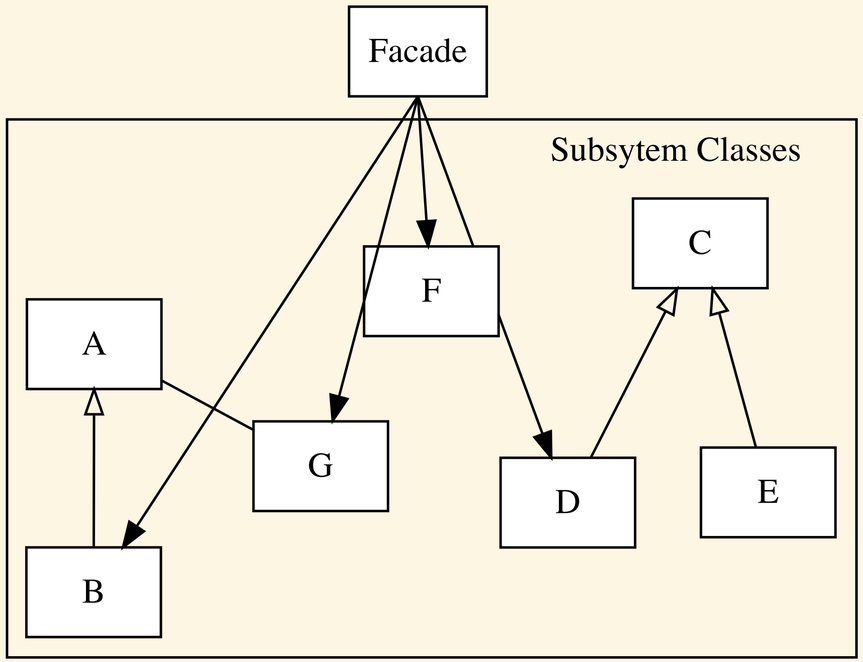
  Decouple the subsystem from clients and other subsystems, thereby promoting subsystem independence and portability

- You want to layer your subsystems

  Define a Facade entry point to each subsystem level

  Simplify dependencies between subsystems by communicating solely through facades

Facade: Participants



- Facade (e.g., Compiler)

  Knows which subsystem classes are responsible for a request and delegates client requests to appropriate subsystem objects

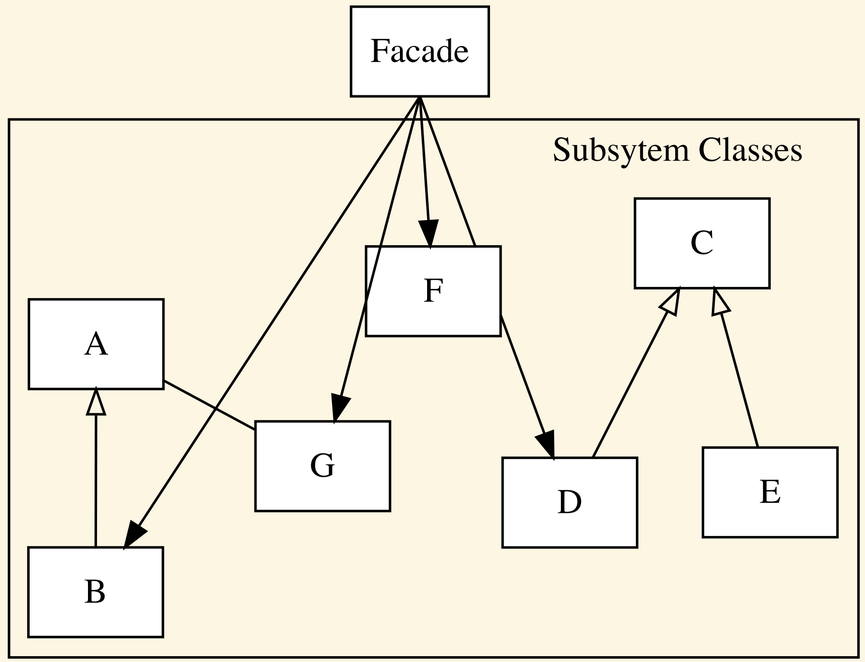
- *subsystem classes* (e.g., Scanner, Parser, ProgramNode, etc.)

  Handle work assigned by the Facade object

  Do not know about the Facade, and keep no references or pointers to the Facade

  Are not marked as part of the pattern

Facade: Collaborations

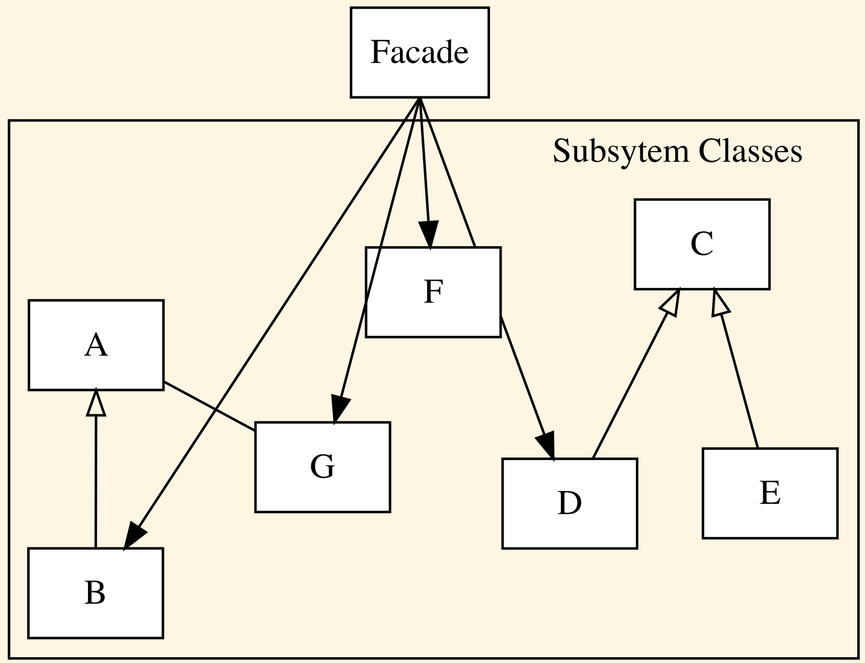


- Clients communicate with the subsystem by sending requests to Facade, which forwards them to the appropriate subsystem objects

- Subsystem objects perform the actual work; the facade may have to work to translate the Facade interface to subsystem interfaces

- Clients that use the Facade don’t have to access its subsystem objects directly but can if needed

Facade: Consequences (Benefits)

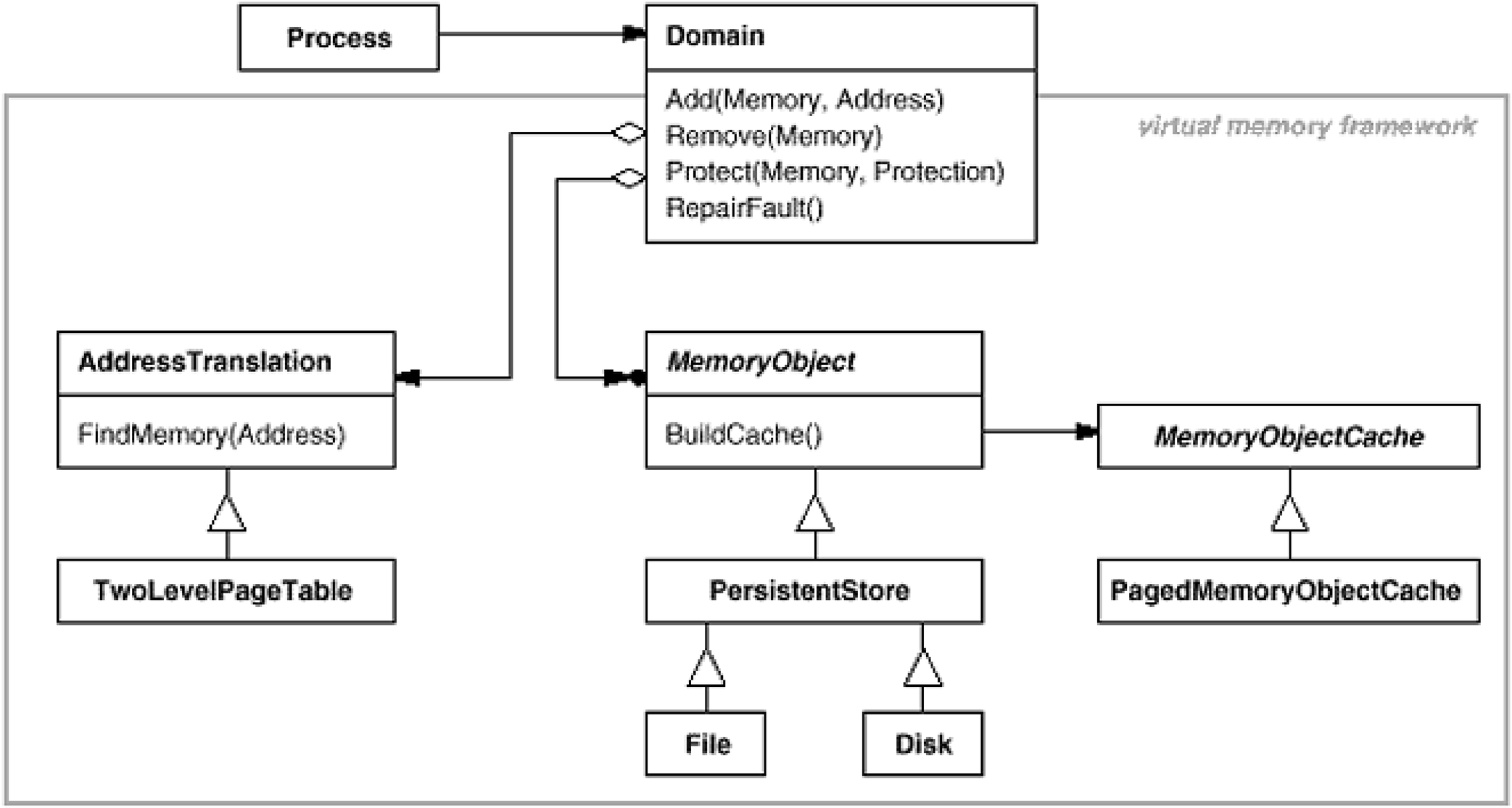


- Shields clients from subsystem components, reducing the number of objects that clients deal with and making the subsystem easier to use
- Promotes weak coupling between the client and the subsystems
- Reduces the need for recompilation
- Simplifies porting to other platforms
- Allows a choice between ease of use and generality, as clients can use the subsystems directly

Implementation

```
class Compiler {
public:
    /*
     * Compiles the characters of the input into Bytecode
     *
     * @param input Character input
     * @param output Bytecode output
     */
    virtual void compile(std::istream&, BytecodeStream&);
};

/*
 * Compiles the characters of the input into Bytecode
 *
 * @param input Character input
 * @param output Bytecode output
 */
void Compiler::compile (std::istream& input, BytecodeStream& output) {

    // setup scanner and builder
    Scanner scanner(input);
    ProgramNodeBuilder builder;

    // generate a parse tree of the scanner using the builder
    Parser parser;
    parser.Parse(scanner, builder);

    // traverse the parse tree for RISC code generation
    RISCCodeGenerator generator(output);
    ProgramNode* parseTree = builder.GetRootNode();
    parseTree->Traverse(generator);
}
```

Known Uses: Choices Operating System



Related Patterns

- *Abstract Factory*

  Used with Facade to provide an interface for creating subsystem objects

  An alternative to Facade to hide platform-specific classes

- *Mediator*

  A facade merely abstracts the interface to subsystem objects to make them easier to use

- *Singleton*

  Usually, only one Facade object is required, and Facade objects are often Singletons