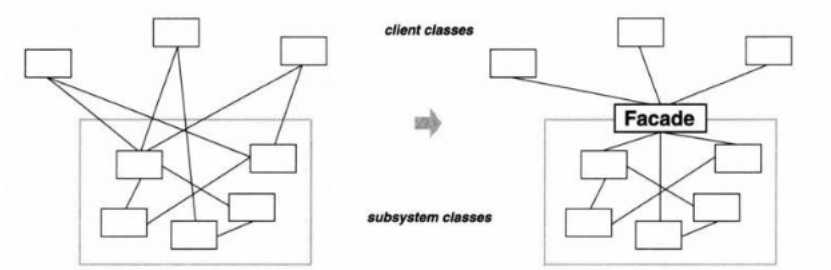
Façade

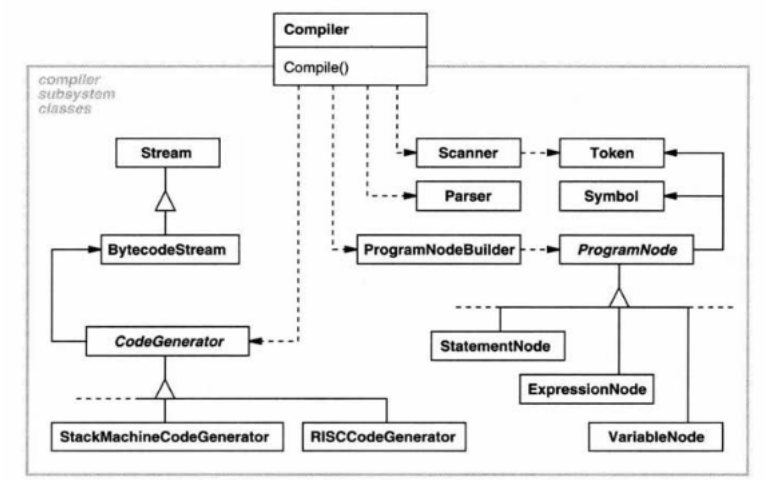
Baris Aktemur
CS 534| Ozyegin University

Contents are from “Design Patterns” by Gamma, Helm, Johnson, Vlissides

Façade

- Intent
  - Provide a unified interface to a set of interfaces in a subsystem.
  - Define a higher-level interface that makes the subsystem easier to use.





- The compiler facade makes life easier for most programmers without hiding the lower-level functionality from the few that need it.

Implementation

```
class Compiler {
public:
  Compiler();
  virtual void Compile(istream&, BytecodeStream&);
};
void Compiler::Compile(istream& input, BytecodeStream&
   output) {
  Scanner scanner(input);
  ProgramNodeBuilder builder;
  Parser parser;
  parser.Parse(scanner, builder);
  RISCCodeGenerator generator(output);
  ProgramNode* parseTree = builder.GetRootNode();
  parseTree->Traverse(generator);
}
```

Applicability

- Use the Facade pattern when
  - you want to provide a simple interface to a complex subsystem. Only clients needing more customizability will need to look beyond the façade.
  - there are many dependencies between clients and the implementation classes of an abstraction. Introduce a facade to decouple the subsystem from clients and other subsystems
  - you want to layer your subsystems. Use a facade to define an entry point to each subsystem level. If subsystems are dependent, make them communicate with each other solely through their facades.

Collaborations

- Clients communicate with the subsystem by sending requests to Facade, which forwards them to the appropriate subsystem object(s).
- Although the subsystem objects perform the actual work, the facade may have to do work of its own to translate its interface to subsystem interfaces.
- Clients that use the facade don't have to access its subsystem objects directly.