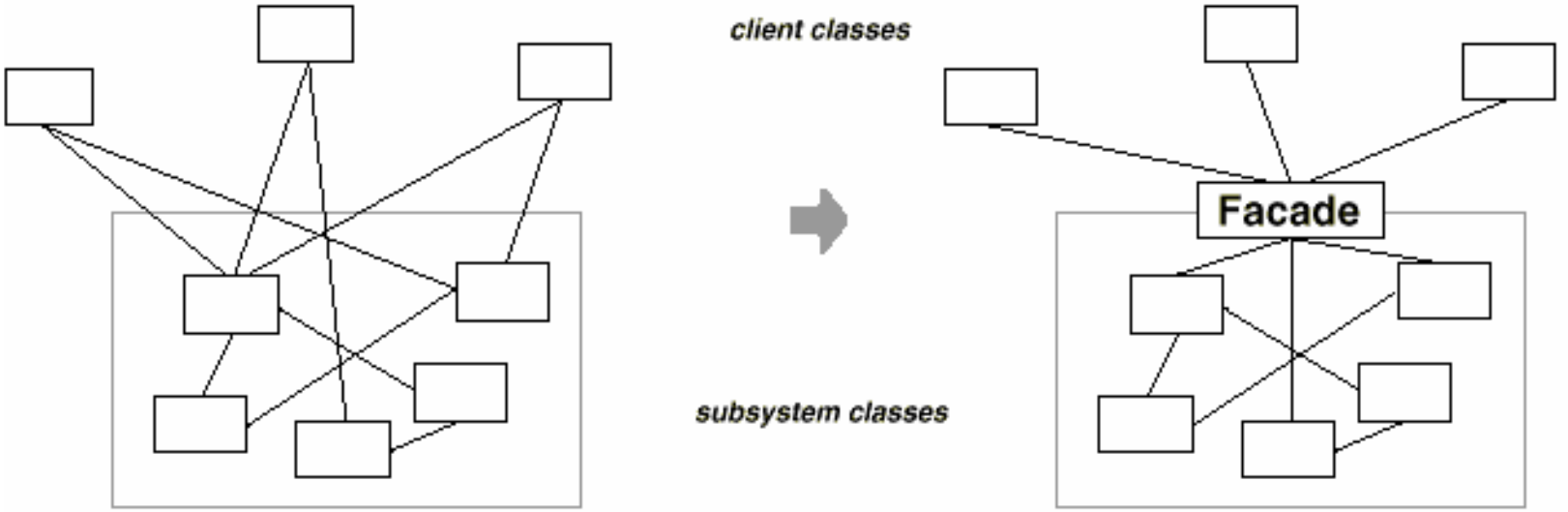
Facade

Facade

- **Známý jako**
  - Facade, Fasáda
- **Účel**
  - sjednocené high-level rozhraní pro subsystém
  - zjednodušuje použití subsystému
  - zapouzdření – skrytí návrhu před uživateli
- **Motivace**
  - zjednodušit komunikaci mezi subsystémy
  - zredukovat závislosti mezi subsystémy
  - neznemožnit používání low-level interfaců
    - pro použití „na míru“

Facade - motivace



Facade – motivace

Puštění filmu:

- zatáhnutí žaluzií
- zapnutí projektoru
- zapnutí DVD přehrávače
- zapnutí ozvučení

Ukončení přehrávání:

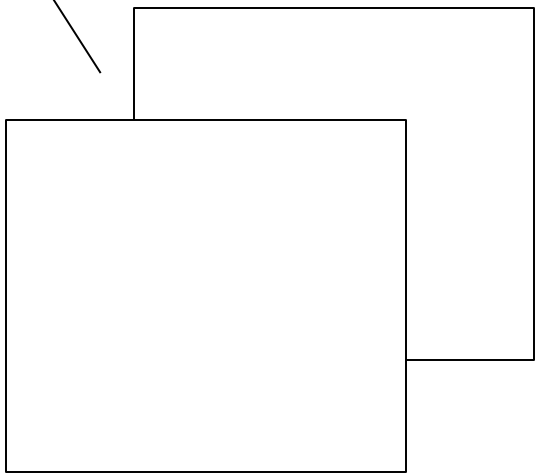
- vypnout DVD přehrávač
- vypnout ozvučení
- vypnout projektor
- vytáhnout žaluzie

Facade – motivace

Řešení:

Univerzální ovladač s funkcemi:

- Přehrát film
- Ukončit film
- Přehrát hudbu
- Vypnout hudbu
- …

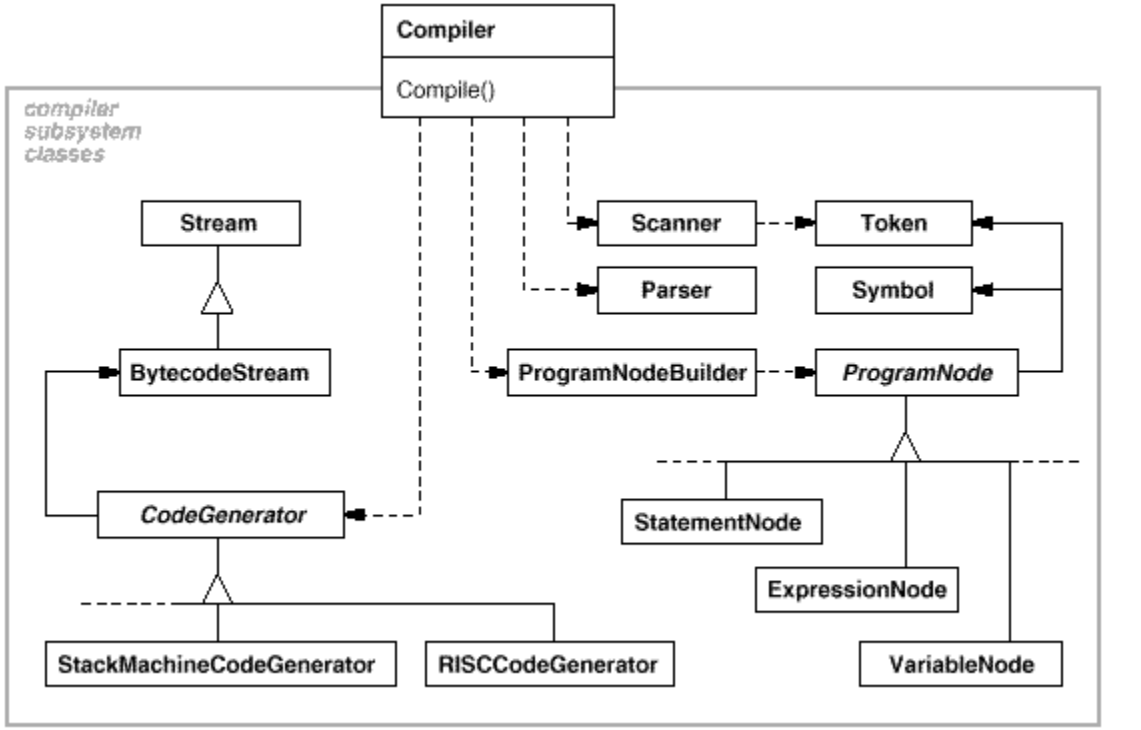


- zatáhnout žaluzie
- zapnout projektor
- zapnout DVD přehrávač
- zapnout ozvučení

- vypnout DVD přehrávač
- vypnout ozvučení
- vypnout projektor
- vytáhnout žaluzie

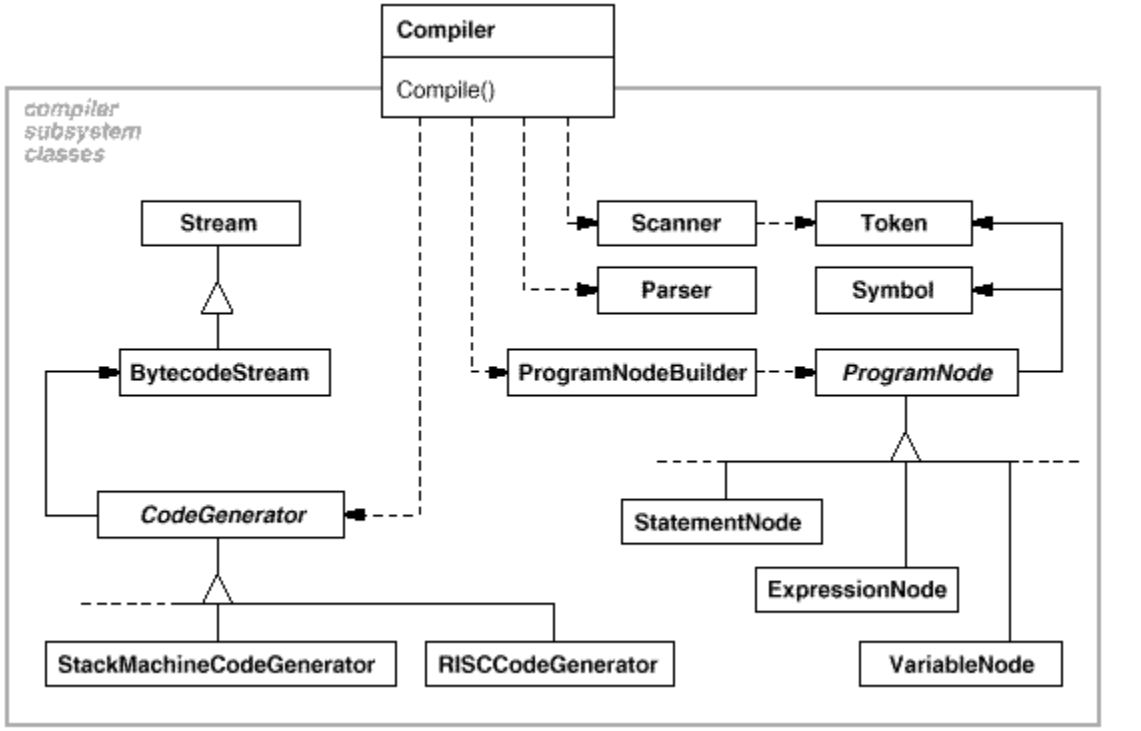
Facade - motivace

- **Příklad**
  - *Compiler*
  - třídy *Scanner*, *Parser*, *ProgramNodeBuilder*, ..
  - Facade poskytuje rozhraní pro práci s kompilátorem na high-level úrovni



Facade - motivace

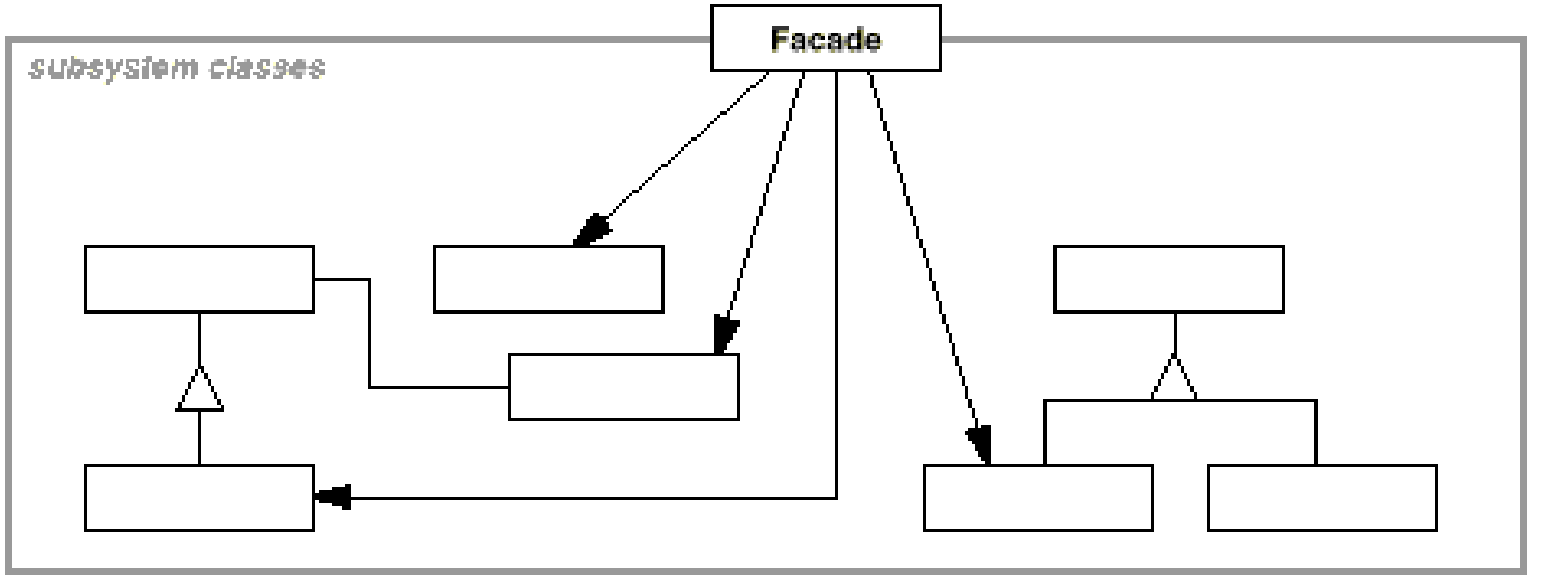
- **Příklad**
  - *transparentnost* – třídy subsystému o Facade nevědí
  - *svoboda volby*- neskrývá třídy subsystému (Parser, Scanner)



Facade - použití

- **Použití**
  - když je subsystém složitý na běžné použití
  - když existuje mnoho závislostí vně subsystému
  - při vytváření vstupních bodů vrstveného systému

- **Struktura**



Facade - použití

- **Účastníci**
  - Facade (kompilátor)
    - zná třídy uvnitř subsystému
    - deleguje požadavky
  - třídy subsystému (Scanner, Parser,...)
    - nevědí o existenci Facade
    - implementují funkčnost

- **Souvislosti**
  - klient se subsystémem komunikuje přes Facade
    - Facade předává požadavky dál
    - může obsahovat vlastní logiku – překlad požadavků na třídy subsystému
  - klienti nemusí používat podsystém přímo

Facade - důsledky

- **Výhody použití**
  - redukuje počet objektů, se kterými klienti komunikují
    - snadnější použití subsystému
  - zmenšuje počet závislostí mezi klienty a subsystémem
    - odstraňuje komplexní a kruhové závislosti
    - méně kompilačních závislostí
  - neskrývá třídy podsytému
    - klient si může vybrat jednoduchost nebo použití na „míru“
  - umožňuje rozšířit stávající funkcionalitu
    - kombinací podsystémů a jejich metod
    - monitorování systému – přístupy, využívání jednotlivých metod

- **Kdy se vyplatí facade do systému implementovat**
  - má cenu o ní uvažovat pouze v případě, kdy je cena za vytvoření fasády menší, než je nastudování systému (podsystémů) uživateli

Facade - příklad

```
class Scanner {
public:
  …
  virtual Token& Scan();
  …
};
```

Třída subsystému

```
class Parser {
public:
  …
  virtual void Parse(Scanner&,ProgramNodeBuilder&);
  …
};
```

Třída subsystému

```
class ProgramNodeBuilder {
public:
  ProgramNodeBuilder();
  virtual ProgramNode* NewVariable(…);
  virtual ProgramNode* NewAssignment(…);
  virtual ProgramNode* NewReturnStatement(…);
  virtual ProgramNode* NewCondition(…) const;
  …
  ProgramNode* GetRootNode();
  …
};
```

Třída subsystému

Facade - příklad

```
class ProgramNode {
public:
  // program node manipulation
  virtual void GetSourcePosition(int& line, int& index);
   …

  // child manipulation
  virtual void Add(ProgramNode*);
  virtual void Remove(ProgramNode*);
  …

  virtual void Traverse(CodeGenerator&);
  protected: ProgramNode();
};
```

Třída subsystému

```
class CodeGenerator {
public:
  virtual void Visit(StatementNode*);
  virtual void Visit(ExpressionNode*);
  …
protected:
  CodeGenerator(BytecodeStream&);
protected:
  BytecodeStream& _output;
};
```

Třída subsystému

Facade - příklad

```
void ExpressionNode::Traverse (CodeGenerator&
  cg)
{
  cg.Visit(this);
  ListIterator i(_children);
  for (i.First(); !i.IsDone(); i.Next()) {
        i.CurrentItem()->Traverse(cg);
  }
}
```

Facade

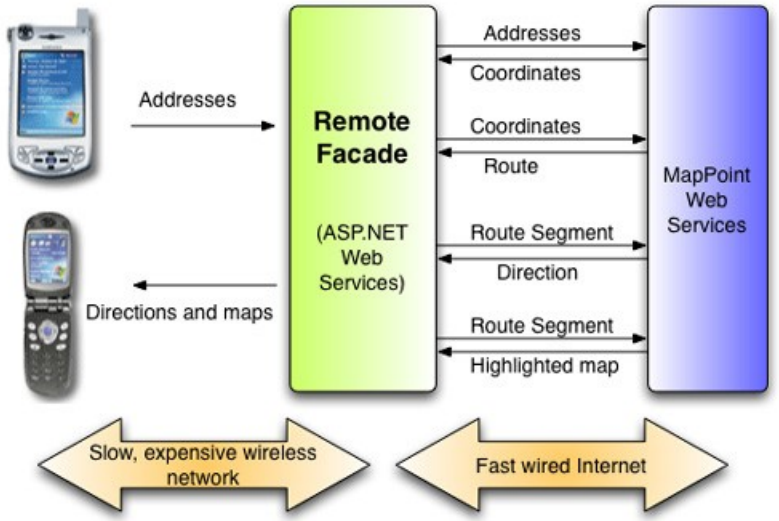
```
class Compiler {
public:
  Compiler();
  virtual void Compile(istream&,BytecodeStream&);
};

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

Facade - implementace

- **Konfigurace fasády**
  - Facade jako abstraktní třída
    - konkrétní implementace podsystému je jejím potomkem
    - klienti komunikují s podsystémem přes rozhraní abstraktní třídy
      - klient neví, která implementace podsystému je použita
      - lze zcela změnit způsob implementace, flexibilita podsystému
  - Facade jako jedna konfigurovatelná třída
    - slabší alternativa předchozího
    - výměna komponent podsystému
- **Viditelnost komponent podsystému**
  - je vhodné určit viditelné a skryté komponenty podsystému
    - analogie private a public metod třídy
    - malá podpora v objektových jazycích

Facade – reálné použití

- **V mobilních aplikacích**



- **Session Facade v J2EE**
  - Fasáda pro webové služby
- **JOptionPane ve Swingu**
  - vytváří různé typy základních dialogových oken a zobrazuje je
  - zjednodušuje používání této rozsáhlé knihovny

Facade – související vzory

- **Abstract Factory**
  - Facade může poskytovat interface pro tvorbu objektů
- **Singleton**
  - Facade jako Singleton - jen jeden vstupní bod do systému
- **Mediator**
  - Mediator také snižuje závislosti
  - na rozdíl od fasády snižuje závislosti mezi komponentami subsystému