GoHotDraw
Evaluating the Go Programming Language with Design Patterns

Frank Schmager, Nicholas Cameron, James Noble
Victoria University of Wellington, New Zealand

PLATEAU 2010

Go

- Google, Nov 2009, Robert Griesemer, Rob Pike and Ken Thompson
- OO without inheritance
  - Embedding for code reuse
- Type inference
  - Inference of interfaces
- C like syntax (curly braces, optional semicolons)
- Package and Public scope
- Garbage collected
- First class functions
- No overloading

Type and method declaration

```
type BoardGame struct {
   player int
}
```

Type and method declaration

```
type BoardGame struct {
  player int
}
func (this *BoardGame) Move() {
  fmt.Println(this.player + “ moved”)
}
```

Type and method declaration

```
type BoardGame struct {
  player int
}
func (this *BoardGame) Move() {
  fmt.Println(this.player + “ moved”)
}
type Chess struct{
  *BoardGame
}
```

Type and method declaration

```
type BoardGame struct {
  player int
}
func (this *BoardGame) Move() {
  fmt.Println(this.player + " moved")
}
type Chess struct{
  *BoardGame
}
func (this *Chess) Move() {
  fmt.Println(this.player + " checkmate")
}
```

Interfaces

```
type BoardGame struct{...}
func (this *BoardGame) Move() {...}
type Chess struct{...}
func (this *Chess) Move() {...}

type Game interface {
  Move()
}
```

Interfaces

```
type BoardGame struct{...}
func (this *BoardGame) Move() {...}
type Chess struct{...}
func (this *Chess) Move() {...}

type Game interface {
   Move()
}

func (this *BoardGame) Play(game Game) {
   //...
   game.Move()
   //...
}
```

Template Method

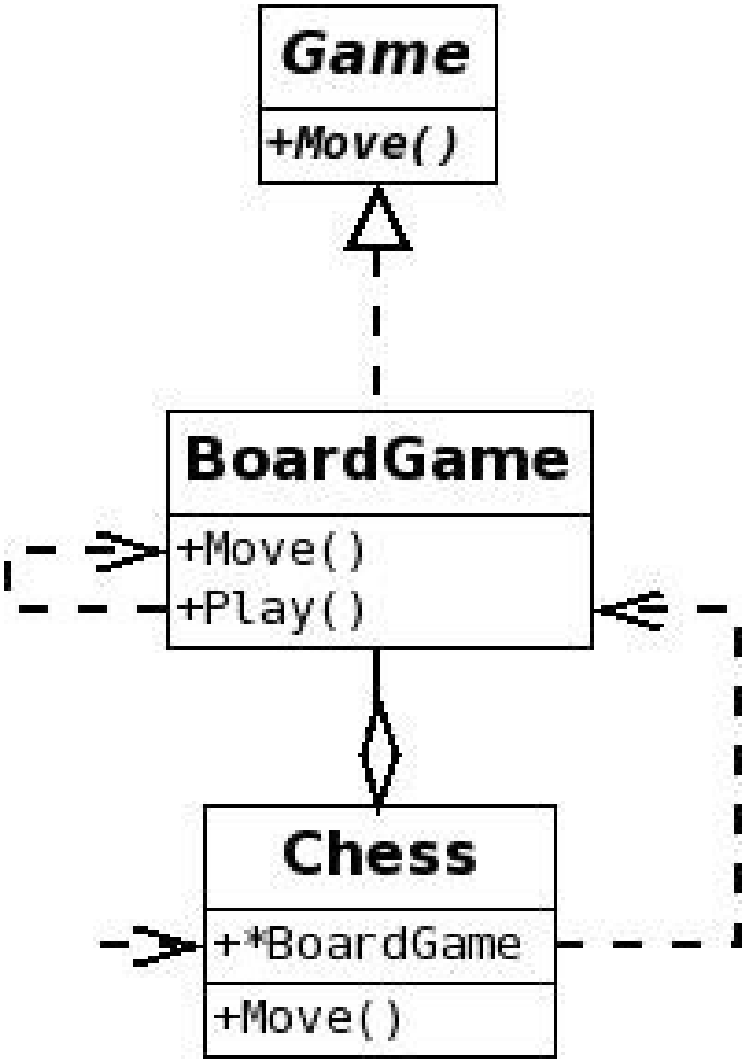
“Define the skeleton of an algorithm in an operation, deferring some steps to subclasses.”

Subclasses redefine certain steps with out changing the algorithm's structure.

```
func (this *BoardGame) Play(game Game) {
  //...
  game.Move()
  //...
}
```

Client-Specified Self

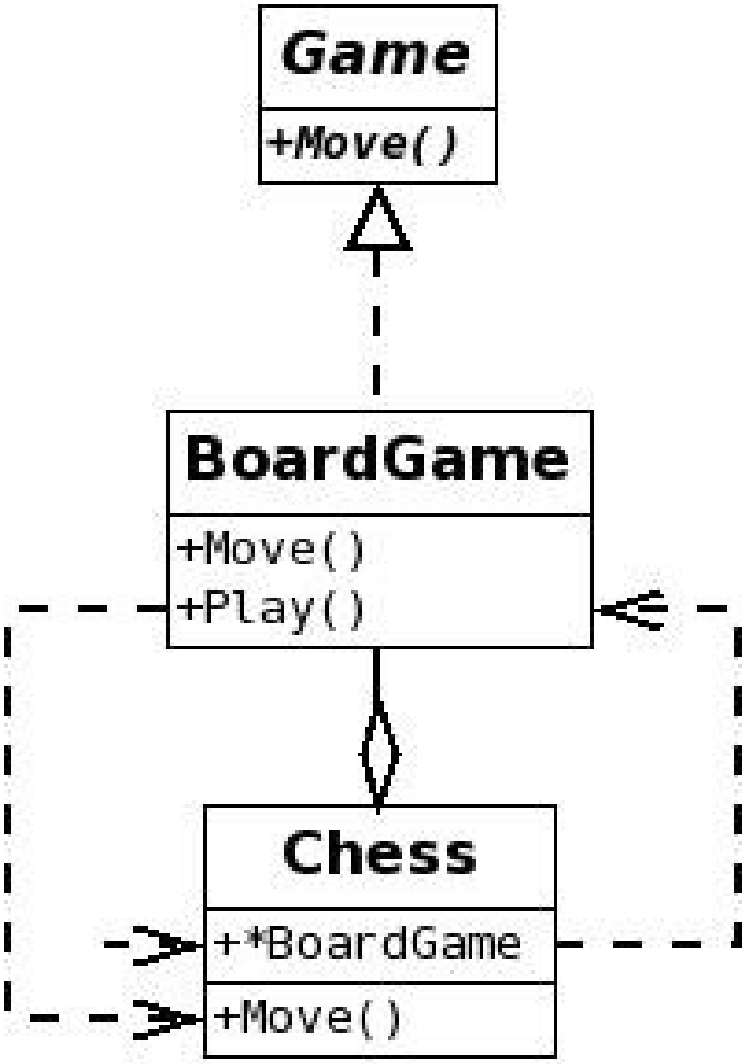
```
chess.Play()
```




```
func (this *BoardGame) Play() {
  //...
  this.Move()
  //...
}
```

Client-Specified Self

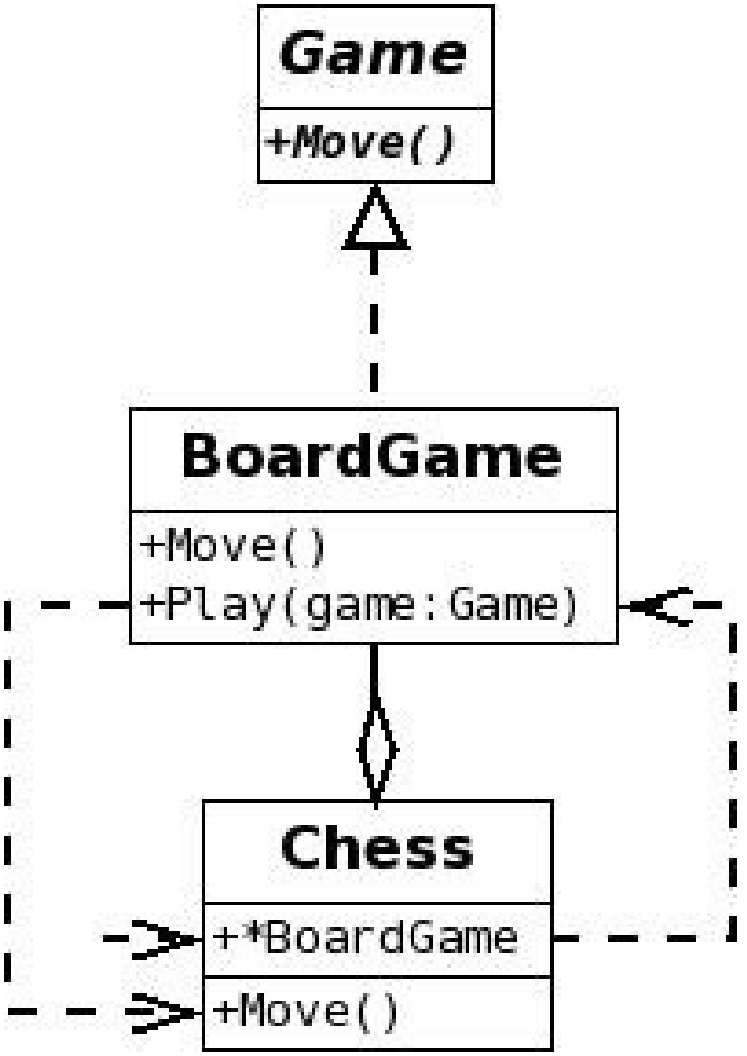
```
chess.Play()
```




```
func (this *BoardGame) Play() {
  //...
  this.Move()
  //...
}
```

Client-Specified Self

```
chess.Play(chess)
```




```
func (this *BoardGame) Play(game Game) {
   //...
   game.Move()
   //...
}
```

Findings about Template Method

- Client-specified self
  - necessary for polymorphism
- Abstract class workaround
  - necessary to provide default and common functionality
- Lack of final methods
  - Required for prevention of overriding of the algorithm

Patterns as evaluation tool

- Patterns are abstractions of programming knowledge
- Patterns provide basis for comparison of programming languages
- Selection of patterns important
  - General purpose
  - Concurrency
  - Systems
  - Architecture

Findings about Go

- Embedding enables code reuse
  - not a replacement for inheritance
- Restricted dynamic dispatch
  - Client-specified self necessary for polymorphism
- No abstract classes
  - Requires workaround
    - Define interface
    - Provide default implementation
    - Embed default implemenation
    - Implement or override missing or default methods

Findings about Go

- Type inference useful
  - Dynamic types increase productivity
  - Go is type safe
- Implicit interfaces
  - Can be defined after the fact
  - Encourages “program to an interface, not an implementation”
  - Small interfaces increases the risk of using the wrong type

Summary

- We used patterns to evaluate Go
  - Embedding is poor substitute for inheritance
  - Interface inference reduces syntactic overhead
  - More experience with Go would have been good
- Patterns are a good tool to explore programming languages
  - Known designs provide basis for comparison
  - Identification of language features that help or hinder pattern implementation

Thank you

Question time