OODP- OOAD– Session 7a

Session times

PT group 1 – Monday	18:00-21:00	room: Malet 403
PT group 2 – Thursday	18:00-21:00	room: Malet 407
FT - Tuesday	13:30-17:00	room: Malet 404

Email: oded@dcs.bbk.ac.uk

Web Page: http://www.dcs.bbk.ac.uk/~oded

Visiting Hours: Tuesday 17:00 to 19:00

Advanced Patterns

Previously on OOAD

- Strategy
- Decorator
- Creation Patterns
  - **Factory method**
  - **Abstract Factory**
  - Singleton

(don’t think in patterns, think patterns)

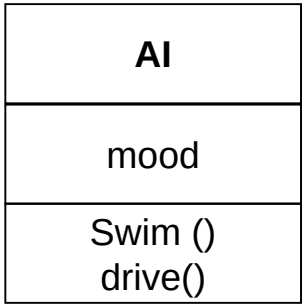
State

DriverAI

- We want to simulate an AI that among other things drives
- To make it feel “human” it should have moods
- The driving and only the driving should depend on the AI’s mood

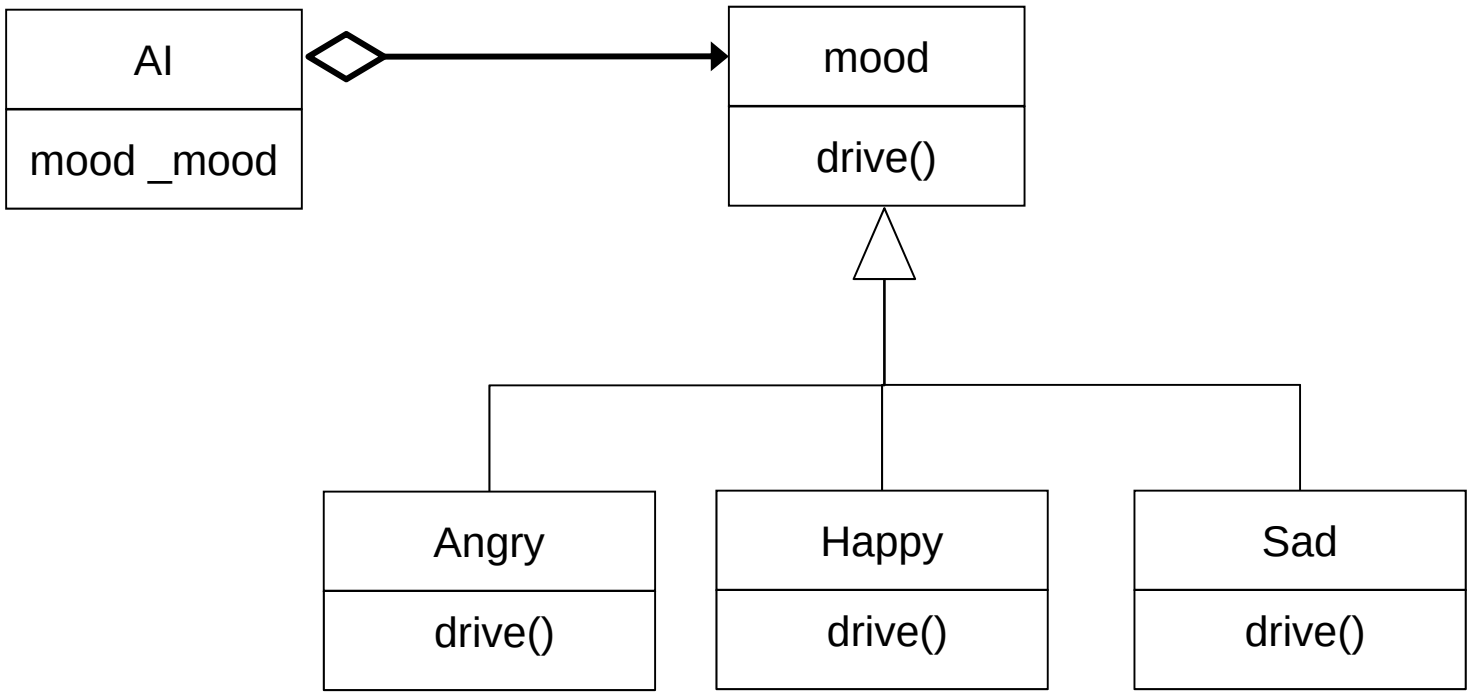
One solution is to, give it a mood attribute.

What is the problem with this solution?



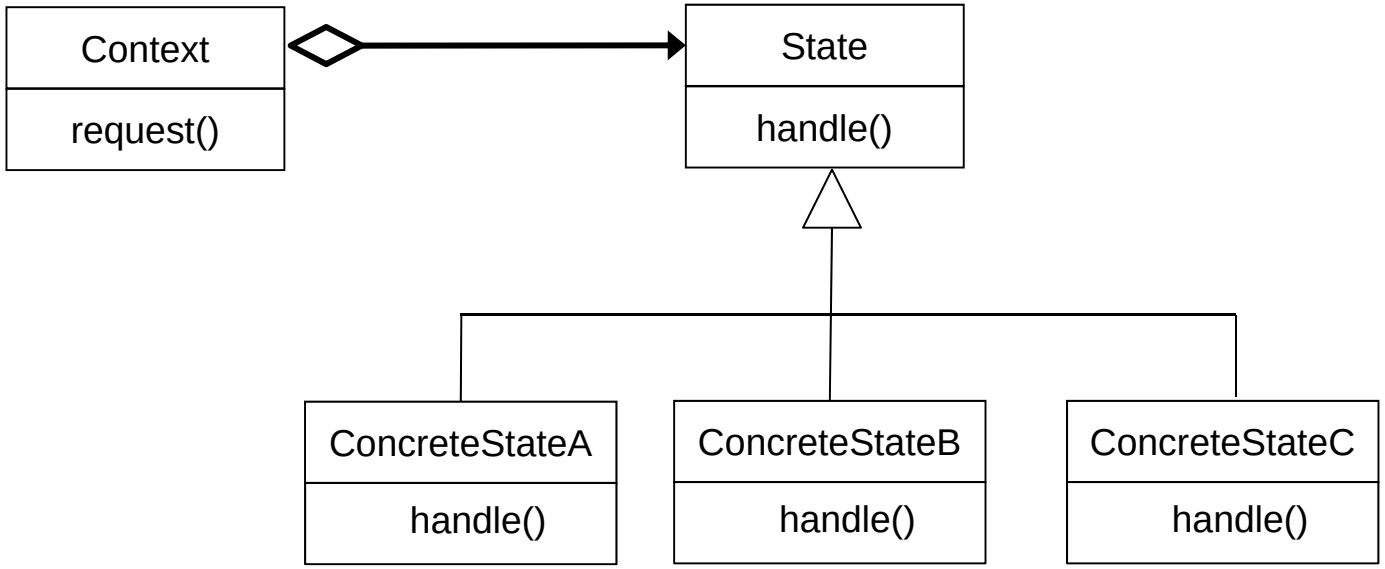
State Pattern

Solution: delegate mood dependent behaviors



State Pattern, Consequences

- Easy to add more states
- More classes
- Less code
- State transition appears explicitly in the DCD

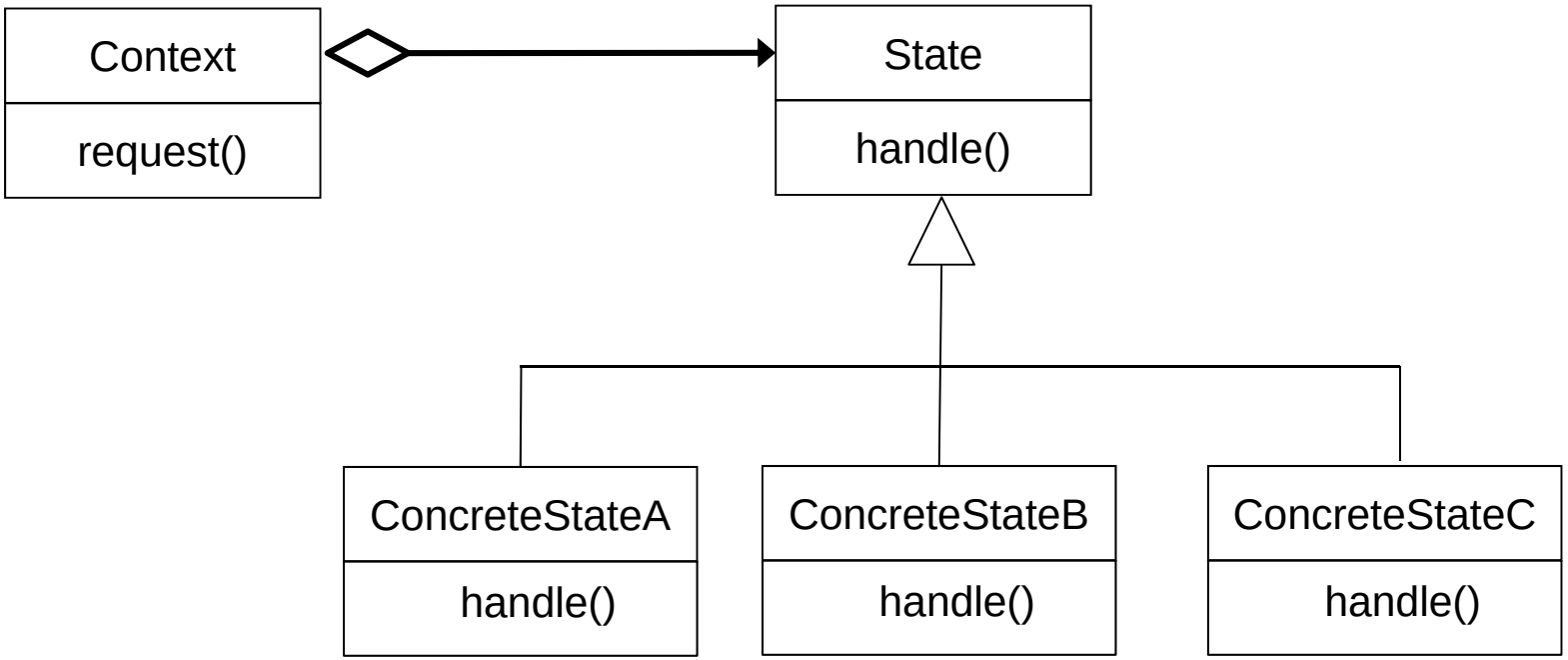


State Pattern, What's Missing?

Who is responsible for the state transitions?

- Context
- ConcreteState classes

What are the pros and cons of each option?



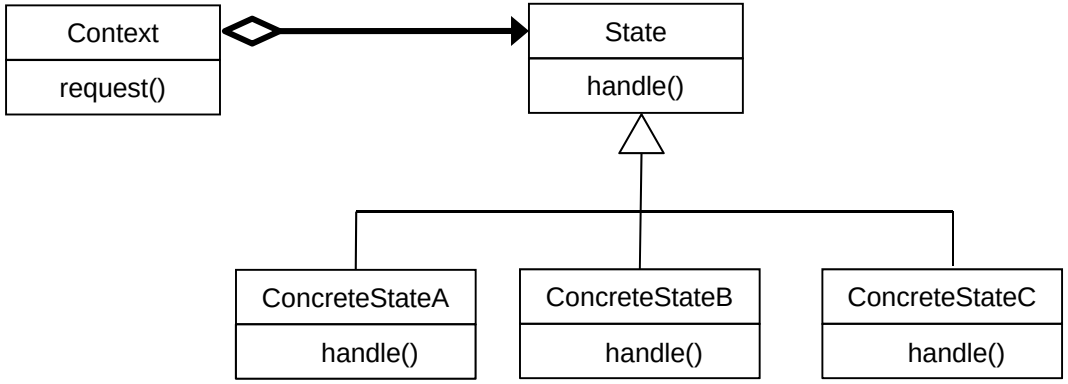
Discussion

Why Use?

- Objects behaviour depends on its state + messy code to implement this

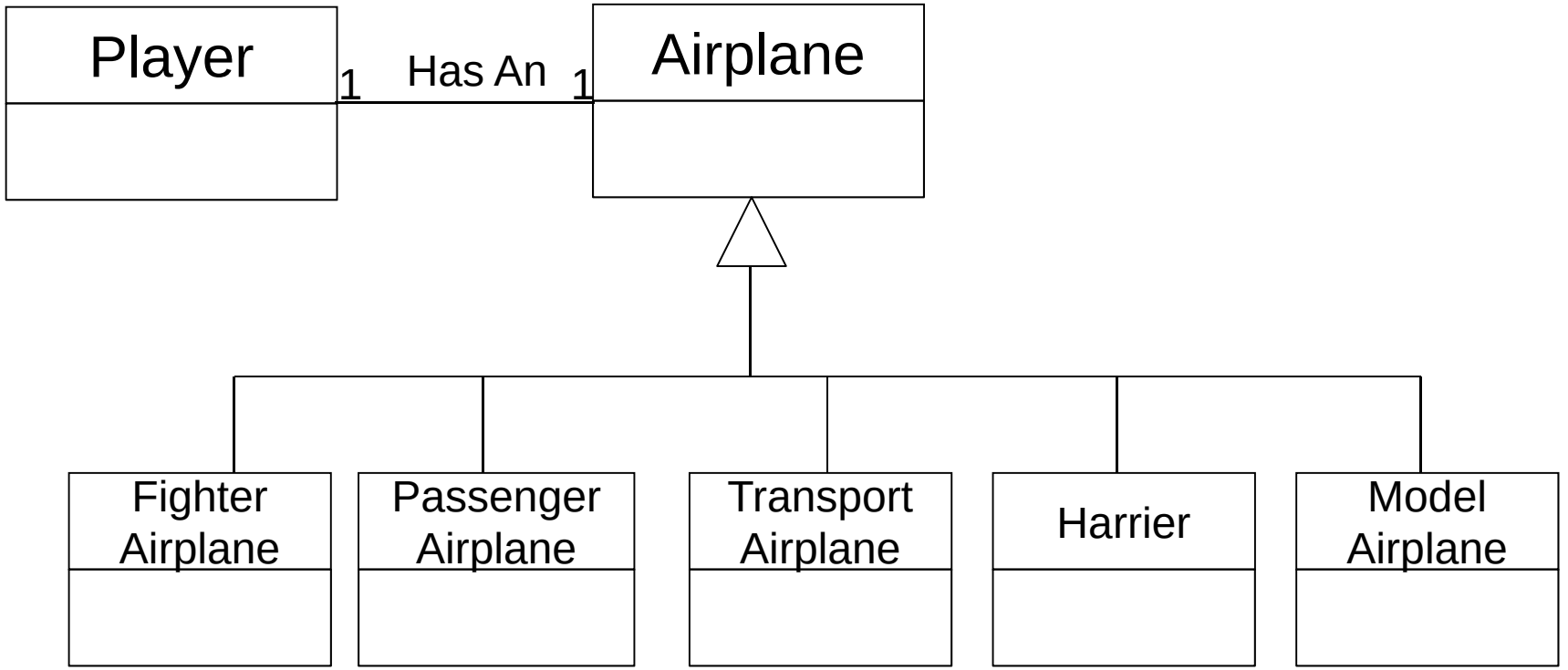
Issues

- When are states created
- How to add more states



Builder

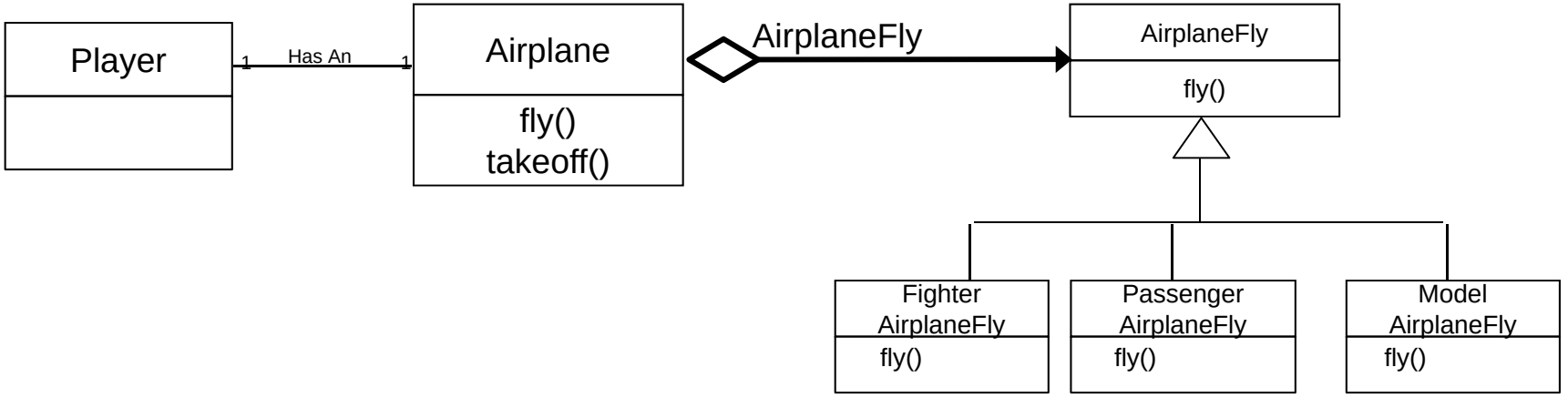
Airplane Game, Domain Model



Problem: Who should construct airplane

If classes inheriting airplane were not composite what would GRASP patterns tell us?

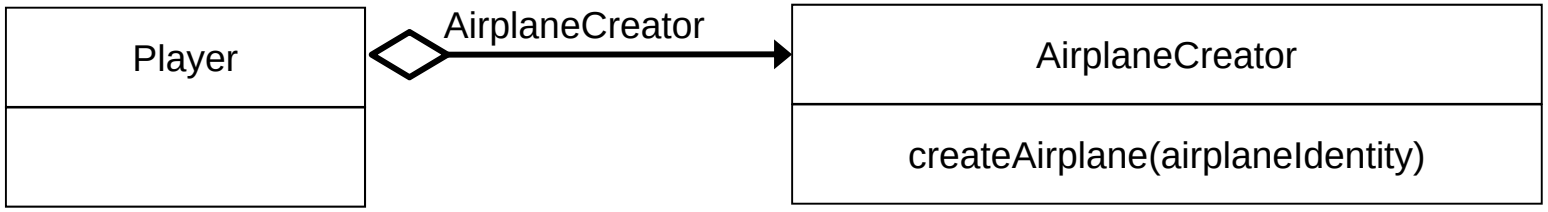
Airplane Game, Domain Model



Classes inheriting airplane are composite, what do GRASP patterns tell us?

- Cohesion
- Coupling
- Expert

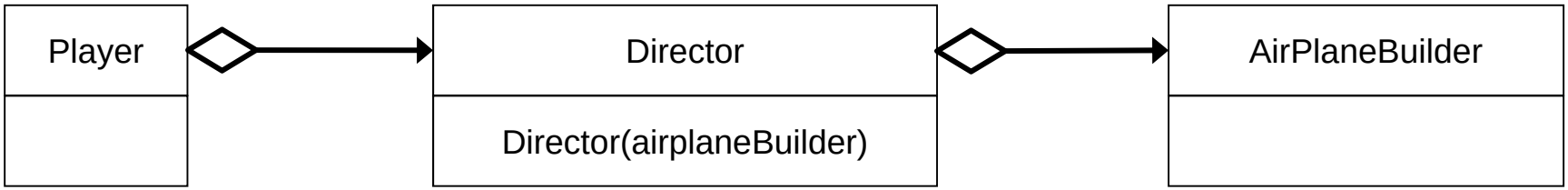
Airplane Game, Delegate



- Idea: create a class that has the expertise to build an airplane

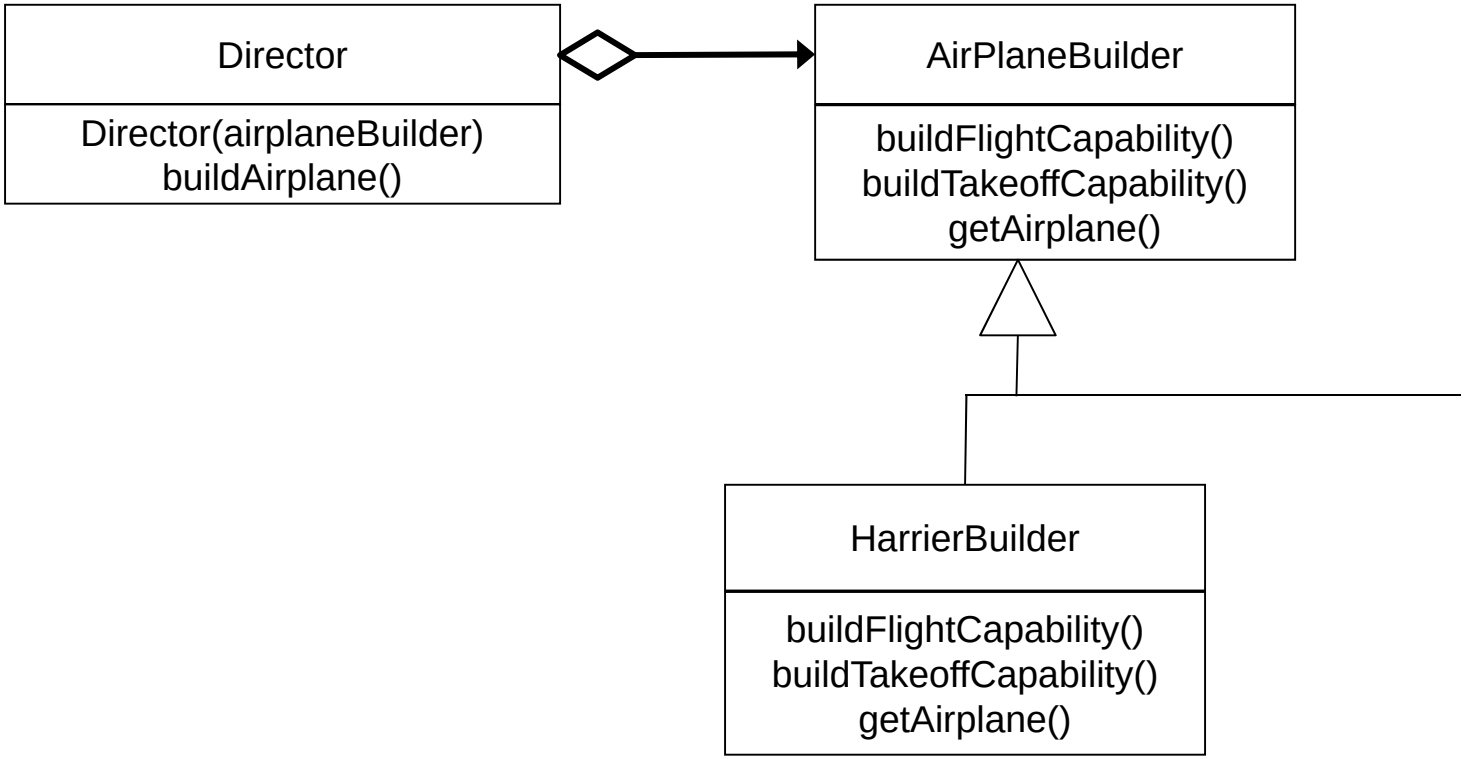
Why is this not such a good idea?

Airplane Game, Delegate,Delegate



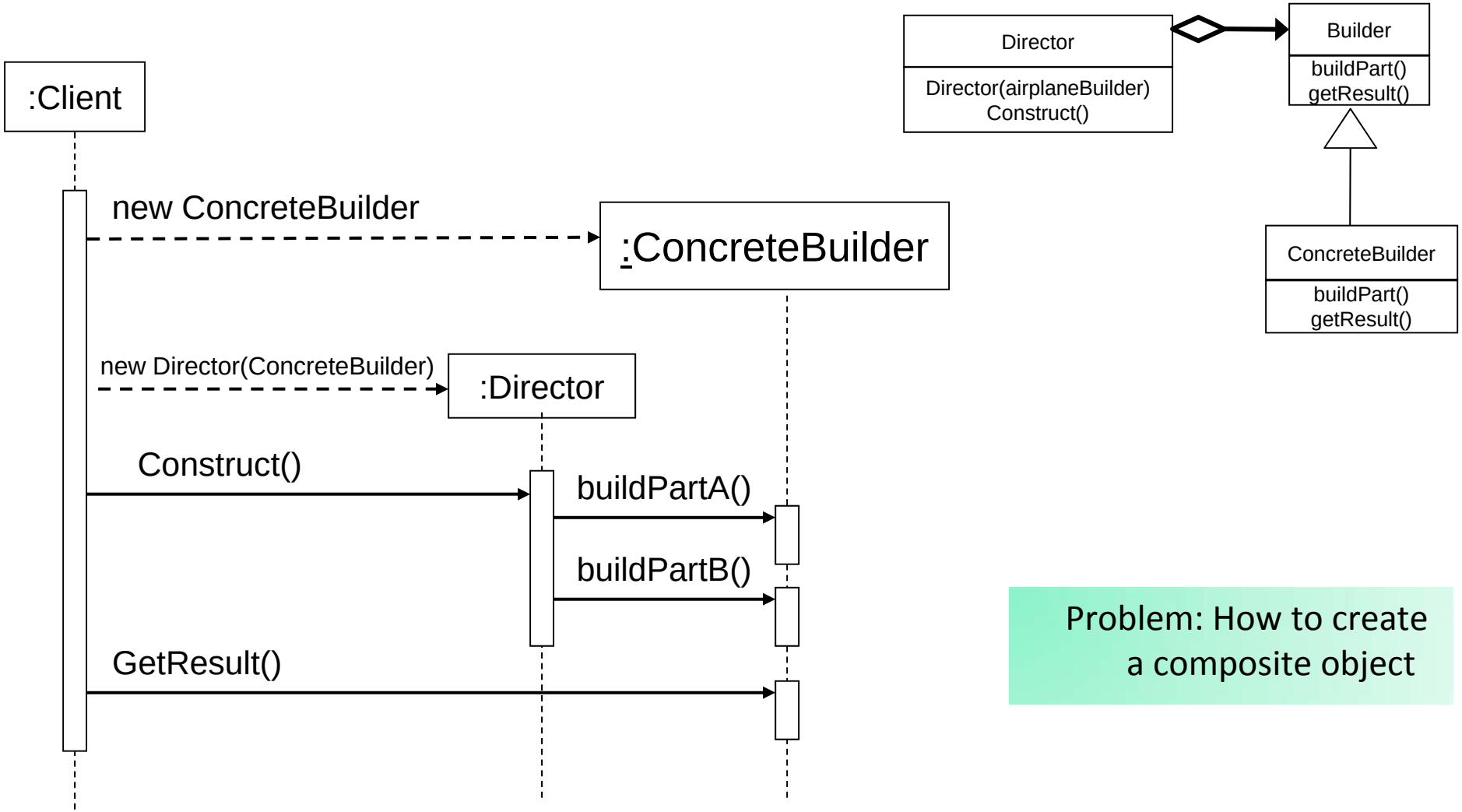
- Director knows how to manage the airplane building
- The Builder knows how to build the parts

Airplane Game, Director, Builder



- Director knows how to manage the airplane building
- The Builder knows how to build the parts

Builder Pattern



Problem: How to create a composite object

- Why does the client create ConcreteBuilder?

Builder Pattern

Pros

- Isolates code for construction and representation
- More control over construction process

Cons

- More work
- More classes

Back to GRASP

More GRASP patterns

- **Information Expert, Creator, Low Coupling, High Cohesion, Controller**

- **Polymorphism**
- **Indirection**
- **Pure Fabrication**
- **Protected Variations**

Polymorphism

Pattern Name: Polymorphism

- **Problem It Solves:** How to handle alternatives based on type? How to create pluggable software components?
- **Solution:** When related alternatives or behaviour vary by type, assign responsibilities for the behaviour using polymorphism

Alternatives based on type – replacing code by classes and types

Pluggable software components – viewing components in client server relationships, how can you replace one server component with another without affecting the client

Where have we seen Polymorphism

- GoF Design Patterns:
  - Strategy
  - Abstract Factory

Indirection

Pattern Name: Indirection

- **Problem It Solves:** Where to assign a responsibility, to avoid direct coupling between two or more things
- **Solution:** Assign the responsibility to an intermediate object

Where have we seen Indirection

- GoF Design Patterns:
  - Strategy
  - Abstract Factory

Pure Fabrication

Pattern Name: Pure Fabrication

- **Problem It Solves:** What object should have the responsibilities, when you don't want to violate high cohesion and low coupling, or other goals, but solutions offered by Expert (for example) are not appropriate.

- **Solution:** Assign a highly cohesive set of responsibilities to an artificial class that does not represent a problem domain concept (such a class is a fabrication of the imagination).

Where have we seen Pure Fabrication

- GoF Design Patterns:
  - Strategy
  - Abstract Factory

Protected Variation

Pattern Name: Protected Variation

- **Problem It Solves:** How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?
- **Solution**: Identify points of predicted variation or instability; assign responsibilities to create a stable interface around them.

Where have we seen Protected Variation

- GoF Design Patterns:
  - Strategy
  - Decorator
  - Abstract Factory

  ...