M8(b)- (More on) Design Patterns

Jin L.C. Guo

Image source: https://www.goodfreephotos.com/albums/animals/mammals/red-squirrel-climbing-up-a-tree.jpg

What is design pattern again?

A standard solution to a common programming problem

A technique for making code more flexible

Shorthand for describing program design

Vocabulary for communication & documentation

……

	Creational	Structural	Behavioral
Class	Factory Method	Adapter (class)	Integreter
			Template Method ✓
Object	Abstract Factory	Adapter (object)	Chain of Responsibility
	Builder	Bridge	Command ✓
	Prototype ✓	Composite ✓	Iterator ✓
	Singleton ✓	Decorator ✓	Mediator
		Flyweight ✓	Momento
		Façade	Observer ✓
		Proxy	State
			Strategy ✓
			Visitor ✓

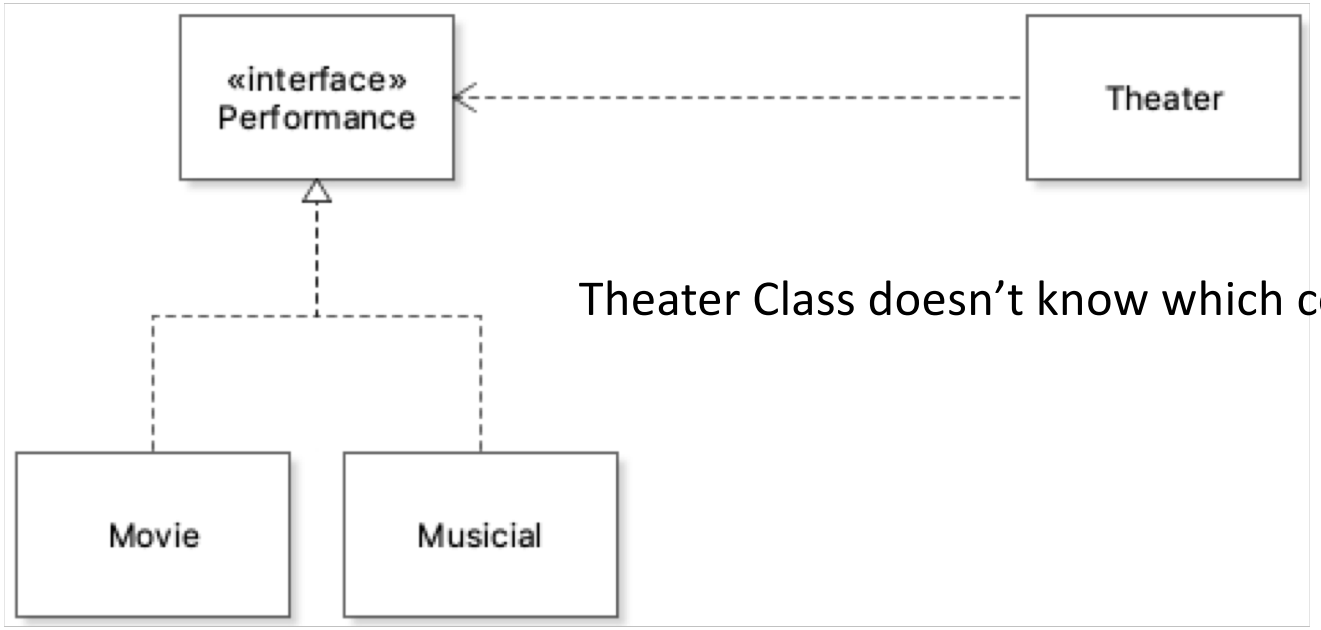
Creational Patterns

- Creational Patterns control object creation
  - encapsulate knowledge about which concrete classes the system uses
  - hide how instances of concrete classes are created and combined.

Why not using the constructors?

Cannot return a subtype
Always create a new object

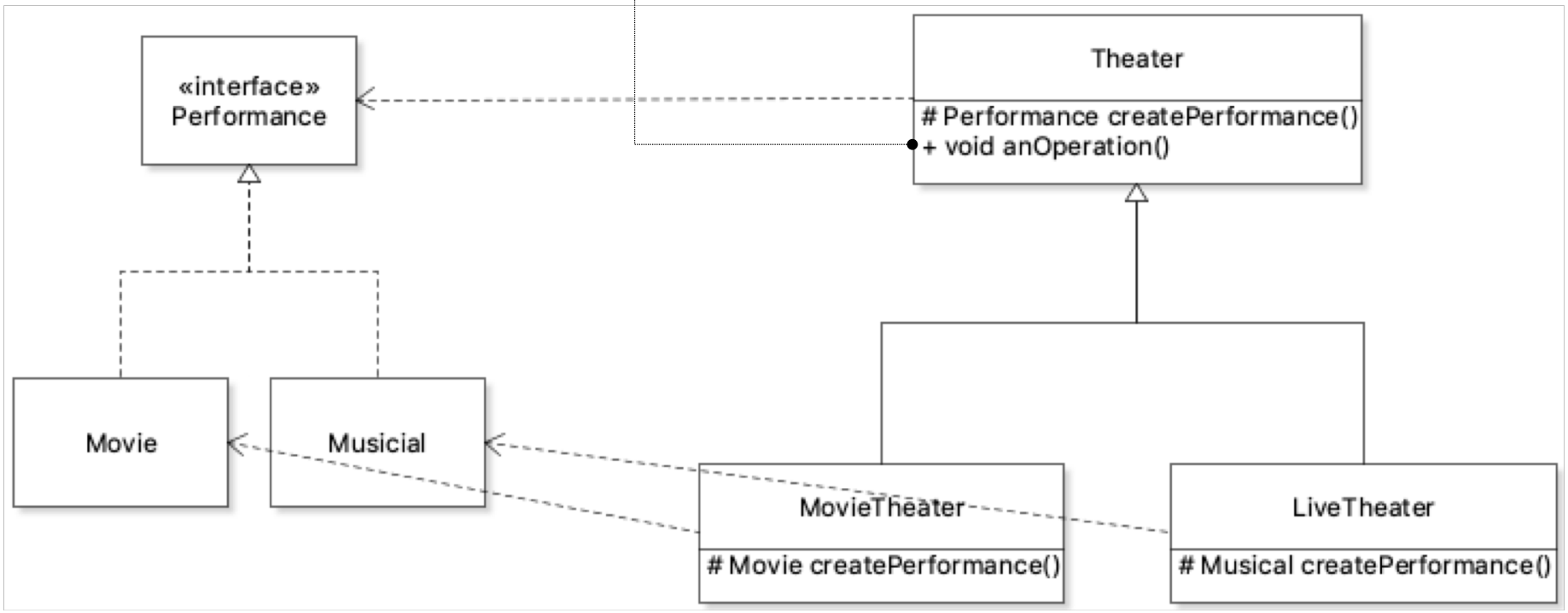
How to have LiveTheater and MovieTheater



Theater Class doesn’t know which constructor to use

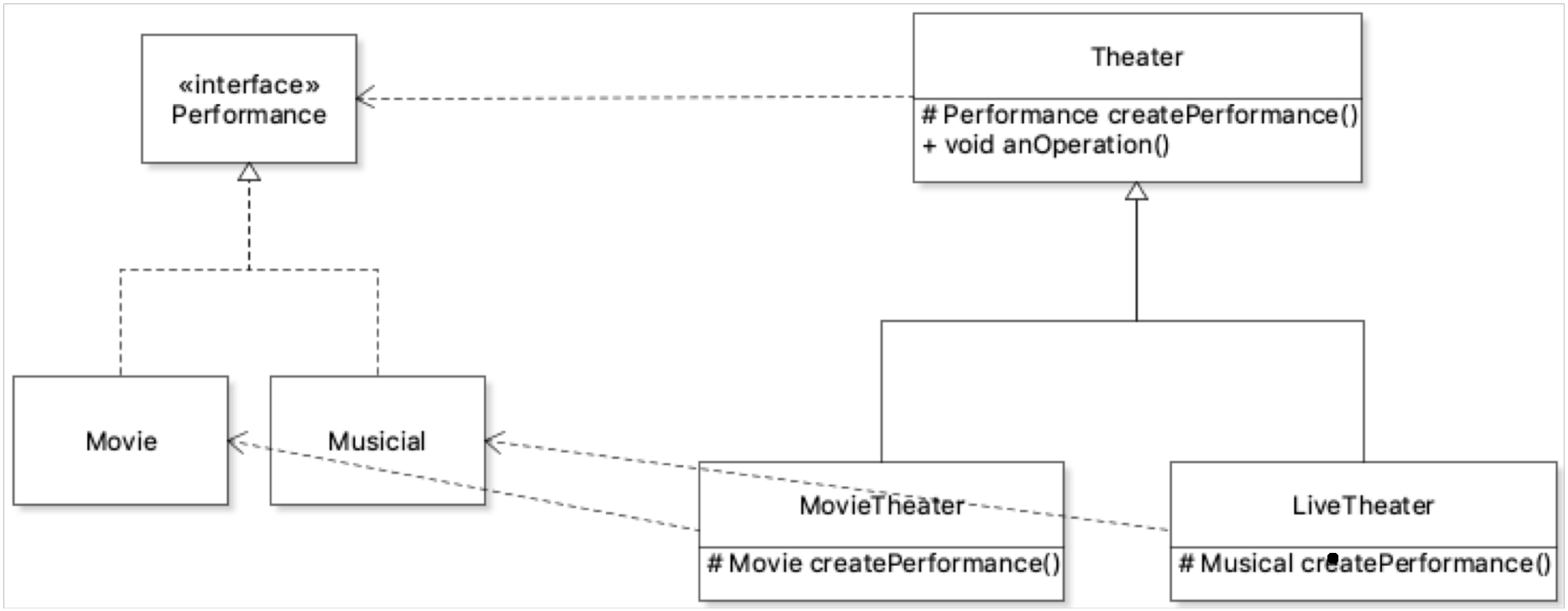
Factory Method

```
public void anOperation() {
    Performance p1 = createPerformance();
    // Some other Operations.
}
```



Factory Method

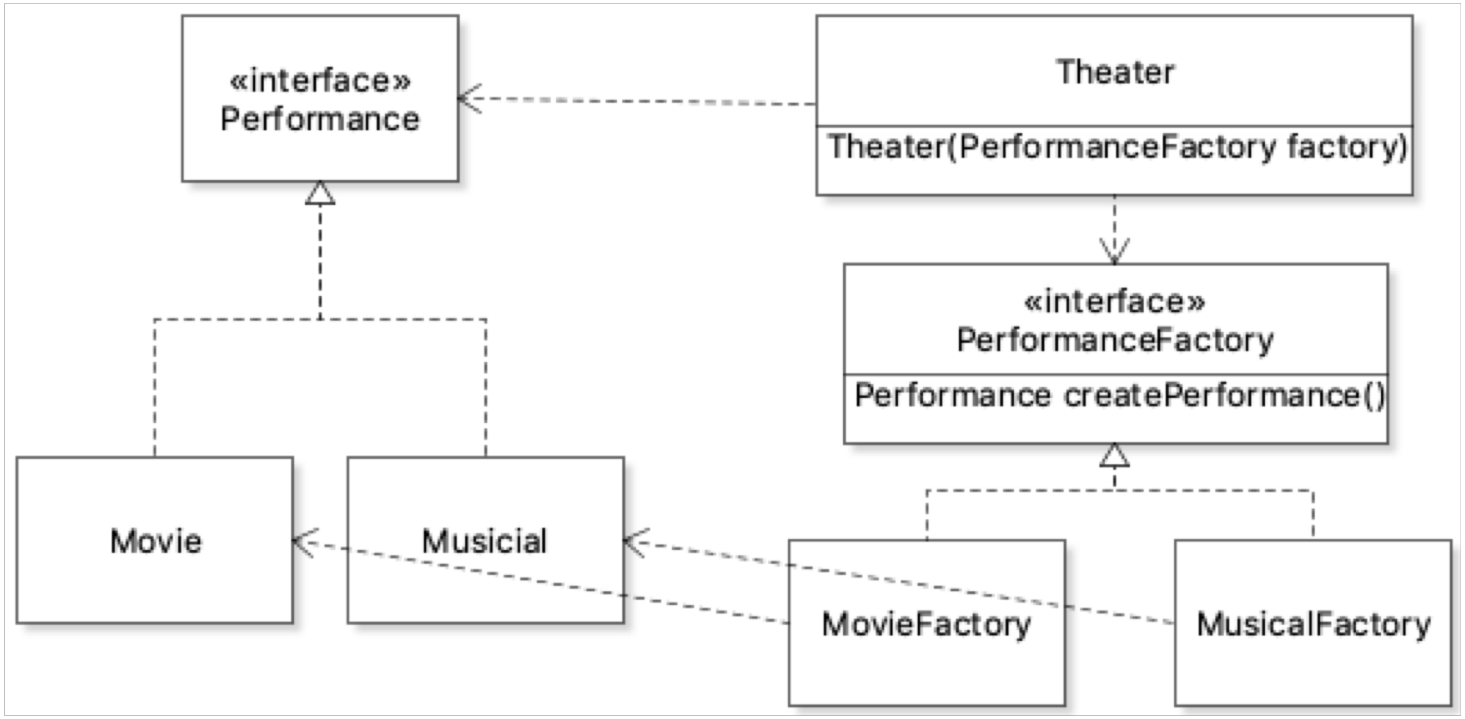
```
protected Musical createPerformance()
{
    return new Musical();
}
```



Abstract Factory

Performance Factory can also in charge of creating a family of related objects: movie, movie audio, movie ads, etc.

```
Performance p1 = factory.createPerformance();
```



Abstract Factory

```
Performance p1 = factory.createPerformance();
```

Builder

```
Performance p1 = builder.getResult();
```

Prototype

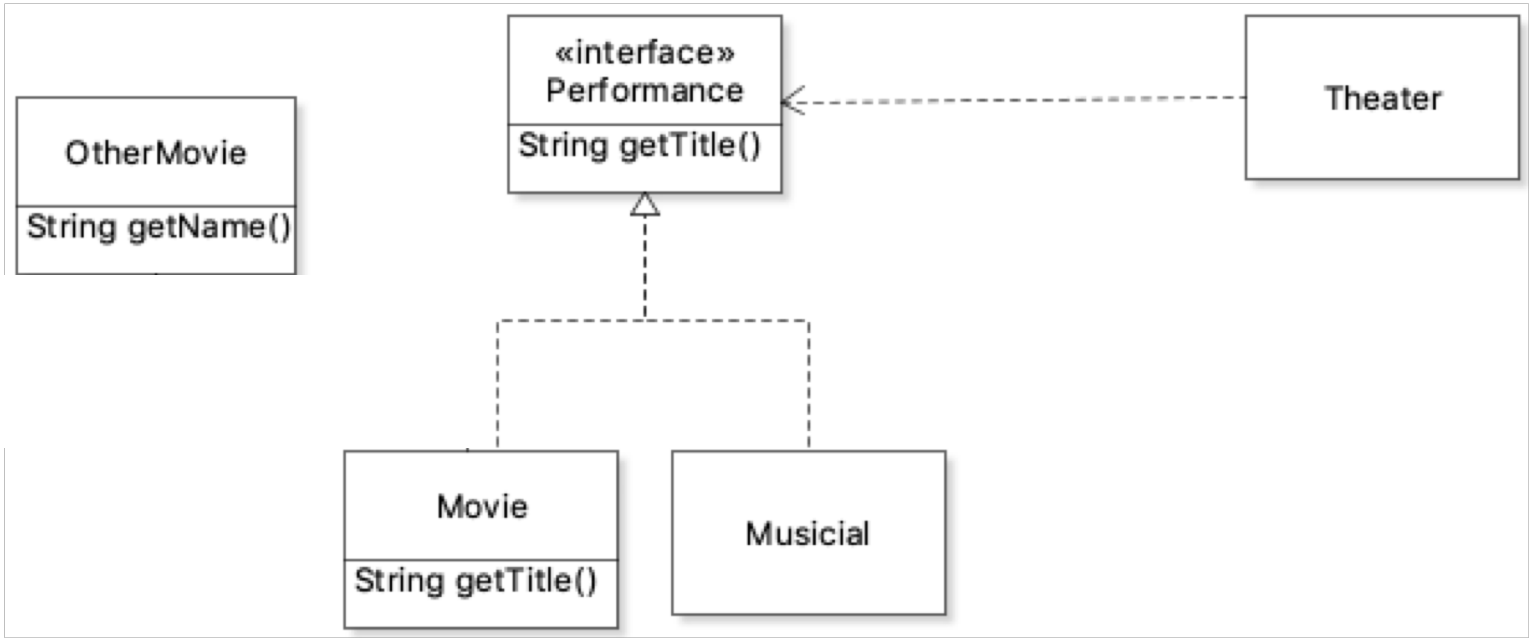
```
Performance p1 = performance.clone();
```

Define an object that’s responsible for knowing the class of the product objects, and make it a parameter of the system.

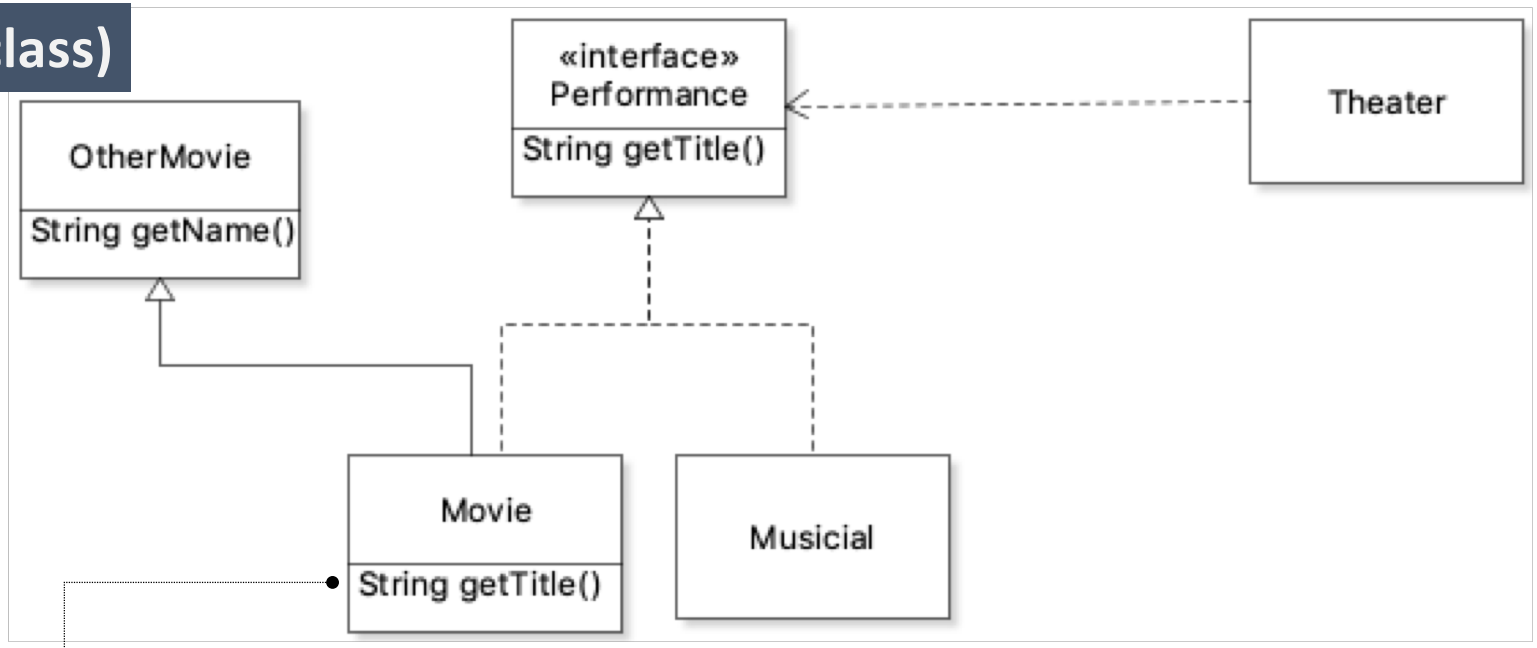
Structural Patterns

- Concerned with how classes and objects are composed to form larger structures.
- Those patterns are similar on structure:
  - single and multiple inheritance for class-based patterns
  - object composition for object patterns.
- Those patterns have distinct intention

	Creational	Structural	Behavioral
Class	Factory Method	Adapter (class)	Integreter
			Template Method ✓
Object	Abstract Factory	Adapter (object)	Chain of Responsibility
	Builder	Bridge	Command ✓
	Prototype ✓	Composite ✓	Iterator ✓
	Singleton ✓	Decorator ✓	Mediator
		Flyweight ✓	Momento
		Façade	Observer ✓
		Proxy	State
			Strategy ✓
			Visitor ✓

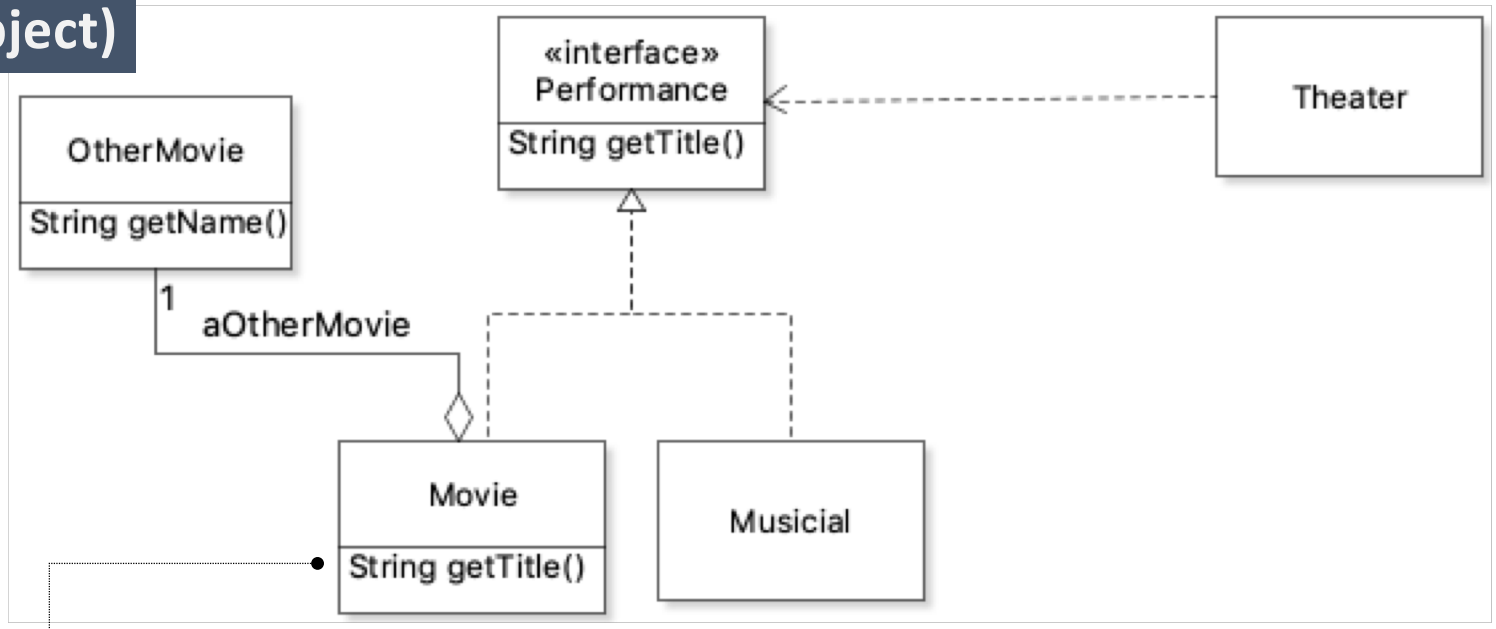

«interface»
Performance
String getTitle()
Theater
OtherMovie
String getName()
Movie
String getTitle()
Musicial

Adapter (class)



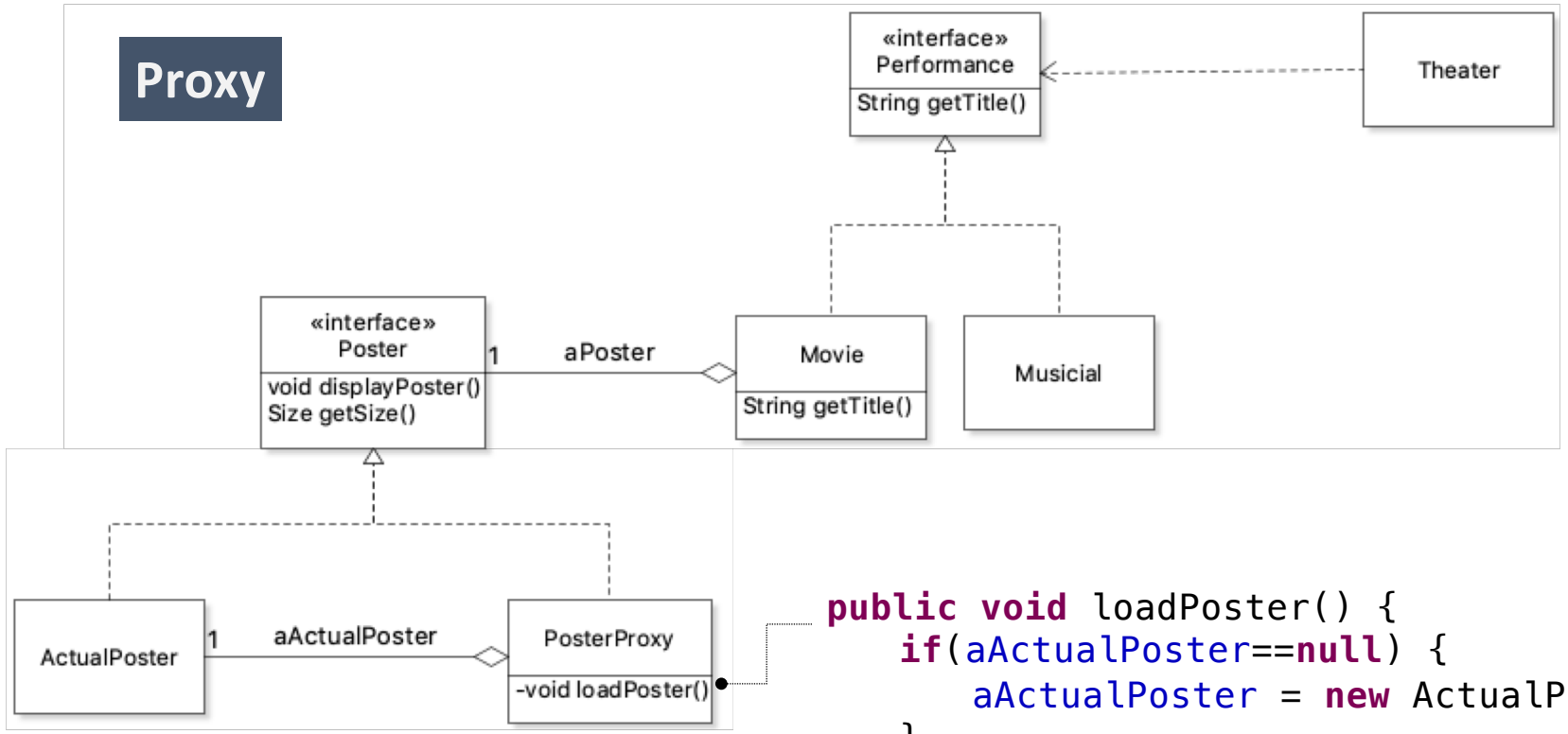

```
public String getTitle() {
   return super.getName();
}
```

Adapter (object)




```
public String getTitle()
{
    return aOtherMovie.getName();
}
```

Proxy




```
public void loadPoster() {
    if(aActualPoster==null) {
        aActualPoster = new ActualPoster();
    }
}
```

Activity

	Interface	Functionality
Adapter (object)	*Different*	*Same*
Proxy	*Same*	*Same*
Decorator	*Same*	*Different*
Composite	*Same*	*Different*

Behavioral Patterns

- Concerned with algorithms and the assignment of responsibilities between objects.
- Describe not just patterns of objects or classes but also the patterns of communication between them.

	Creational	Structural	Behavioral
Class	Factory Method	Adapter (class)	Interpreter
			Template Method ✓
Object	Abstract Factory	Adapter (object)	Chain of Responsibility
	Builder	Bridge	Command ✓
	Prototype ✓	Composite ✓	Iterator ✓
	Singleton ✓	Decorator ✓	Mediator
		Flyweight ✓	Momento
		Façade	Observer ✓
		Proxy	State
			Strategy ✓
			Visitor ✓

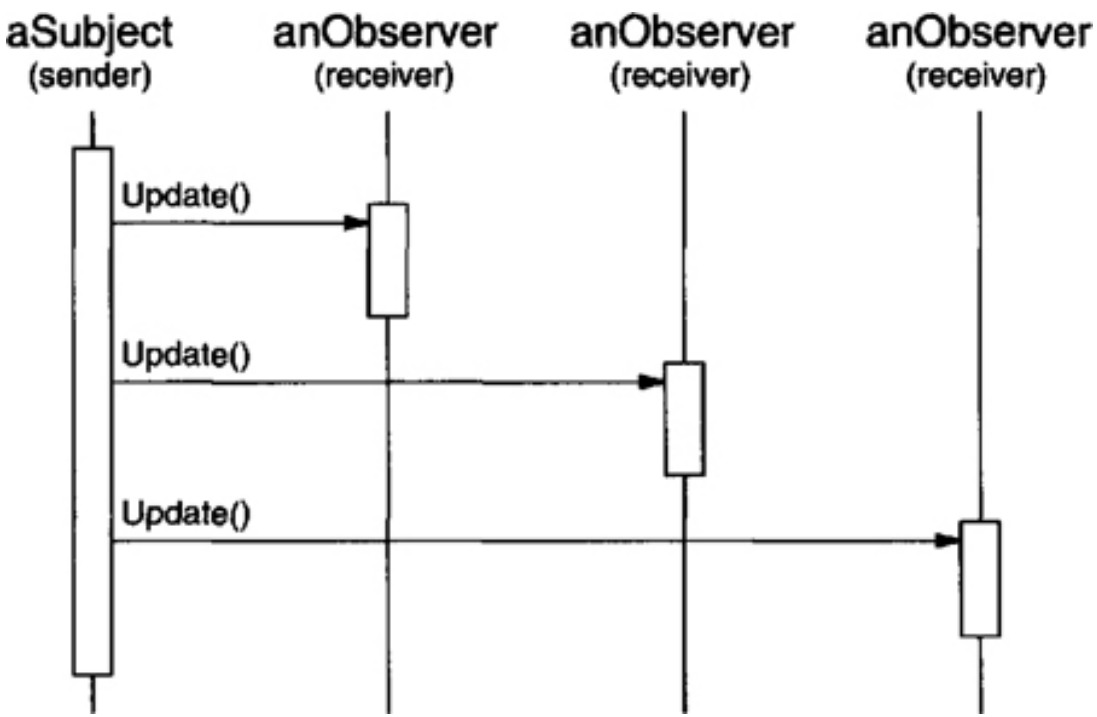
Encapsulating Variation

- When an aspect of a program changes frequently, the behavioral patterns define an object that encapsulate that aspect:
  - A Strategy object encapsulates _____;
  - An iterator object encapsulates _____;
  - An command object encapsulates _____;
  - An visitor object encapsulates_____;

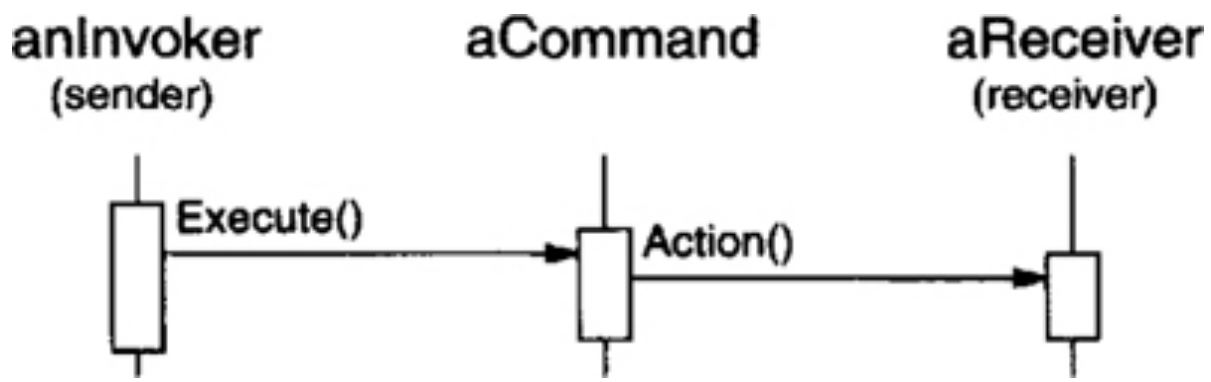
Passing around object

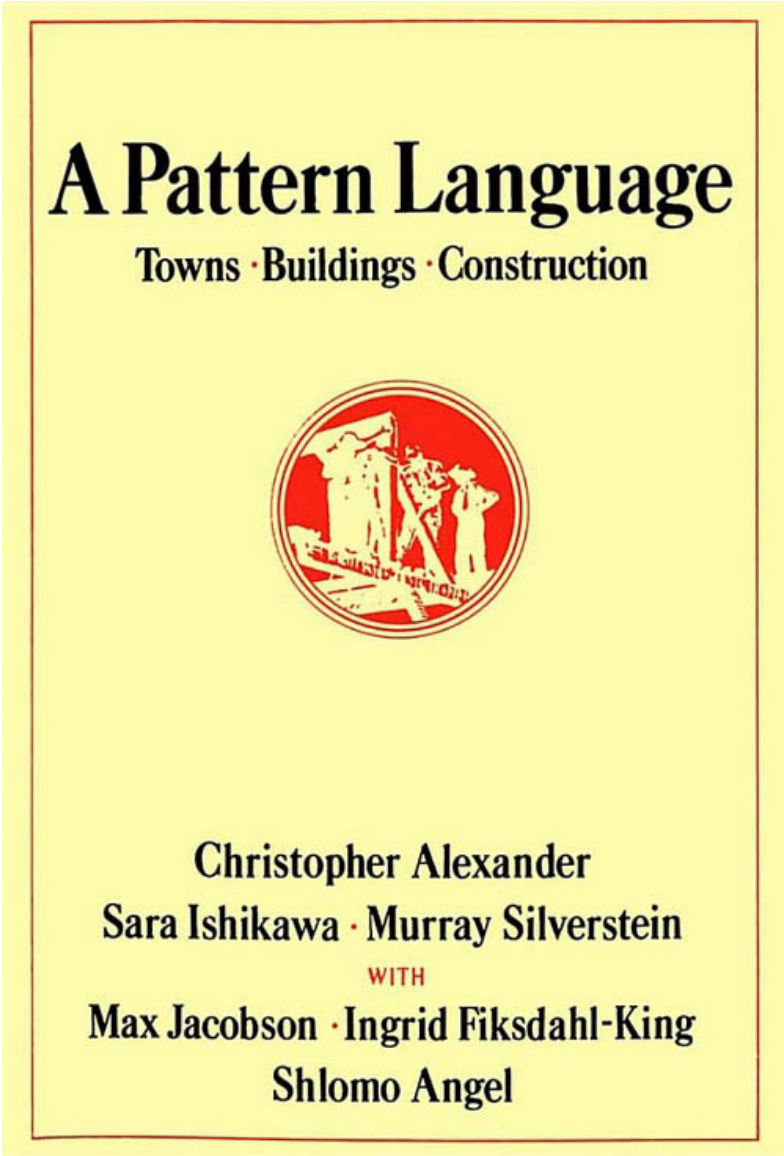
- Object is returned to the client and will be used at later time.
  - An iterator
  - A command
- Object is passed into the system as argument
  - A strategy
  - A visitor

Decoupling information sender and receiver



Decoupling information sender and receiver





"It is possible to make buildings by stringing together patterns, in a rather loose way. A building made like this, is an assembly of patterns. It is not dense. It is not profound. But it is also possible to put patterns together in such a way that many patterns overlap in the same physical space: the building is very dense; it has many meanings captured in a small space; and through this density, it becomes profound."

- Christopher Alexander