COM S 362
Object-Oriented Analysis & Design

Programming Paradigms

Criticism of Design Patterns

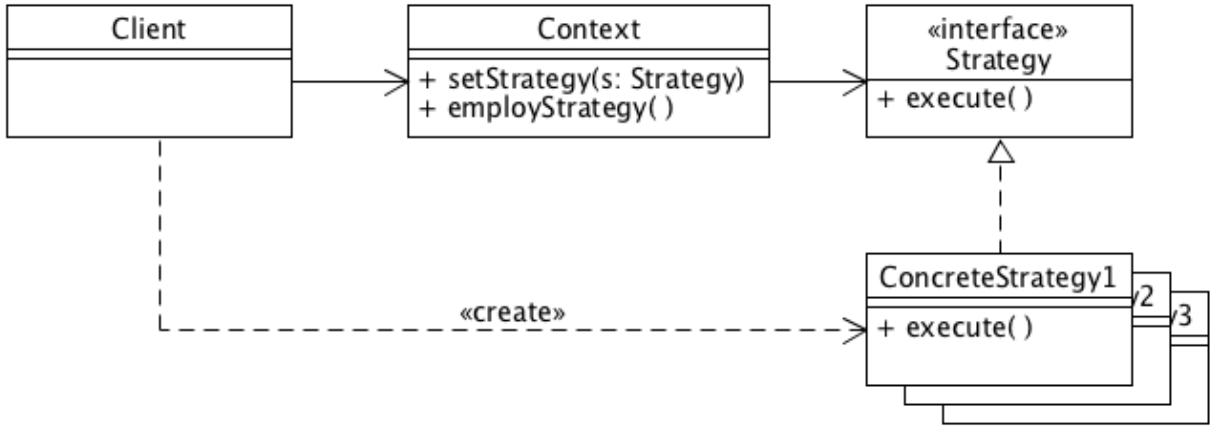
- Common criticism of design patterns is they just cover up missing features in OO based languages
- Behavioral patterns are about dealing with algorithms (e.g., functions) as objects
- In functional programming the basic unit is the function (not object), makes the behavioral patterns simple
- … but now Java has functional programming!

Object-Oriented vs Functional

	Object-Oriented	Functional
Basic Type	Object	Function

Example	`class Sum implements Strategy {`
	`  public int execute(int a, int b) {`
	`  return a + b;`
	`  }`
	`}`
	`...`
	`Strategy sumOp = new Sum();`
		`Strategy sumOp = (a, b) -> a + b;`

Strategy Pattern: Structure



- Strategy: interface common to all supported algorithms
- ConcreteStrategy: implements the algorithm
- Context: maintains a reference to one of ConcreteStrategy and communicates with it via the strategy interface
- Client: creates a ConcreteStrategy and sets it in the Context

Strategy Pattern in Functional

Concrete Strategies as Classes

```
interface Strategy {
    public int execute(int a, int b);
}

class Sum implements Strategy {
    public int execute(int a, int b) {
        return a + b;
    }
}

class Multiply implements Strategy {
    public int execute(int a, int b) {
        return a * b;
    }
}

public class Context {
    Strategy strategy;

    public void setStrategy(Strategy s) {
        strategy = s;
    }

    public int employStrategy(int a, int b) {
        return strategy.execute(a, b);
    }

    public static void main(String[] args) {
        Context op = new Context();
        op.setStrategy(new Sum());
        int result = op.employStrategy(2, 4);
        System.out.println(result);
```

Concrete Strategies as Functions

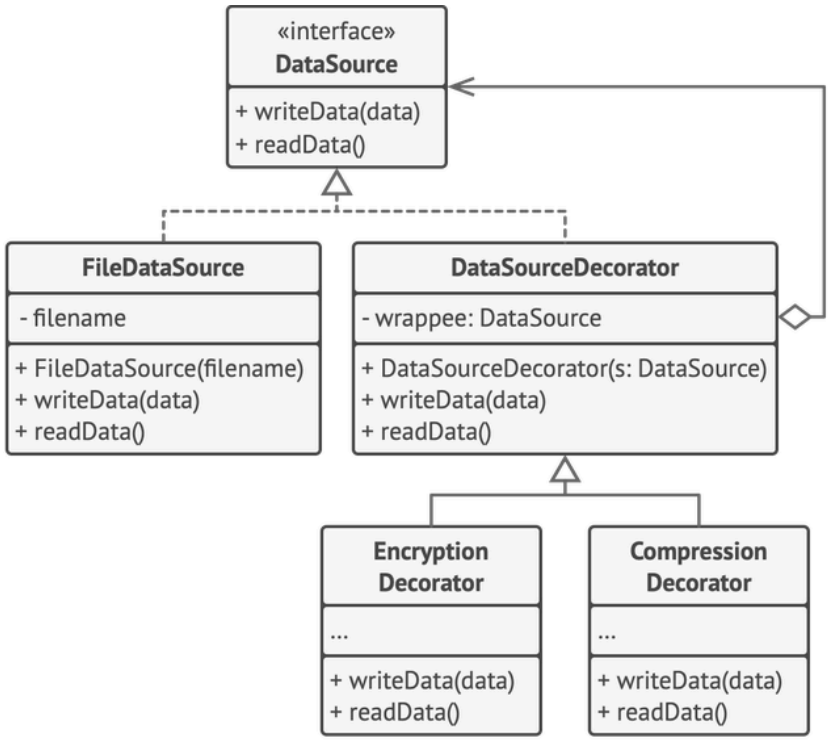
```
interface Strategy {
    public int execute(int a, int b);
}

public class Context {
    Strategy strategy;

    public void setStrategy(Strategy s) {
        strategy = s;
    }

    public int employStrategy(int a, int b) {
        return strategy.execute(a, b);
    }

    public static void main(String[] args) {
        Context op = new Context();
        Strategy sum = (a, b) -> a + b;
        op.setStrategy(sum);
        int result = op.employStrategy(2, 4);
        System.out.println(result);
```

Functional Composition

- Powerful concept in function programming is composition
- What pattern is midpoint looking like?

```
Strategy sum = (a, b) -> a + b;
Strategy divide = (a, b) -> a / b;
Strategy midpoint = (a, b) -> divide.execute(sum.execute(a, b), 2);
```

Decorator Pattern




```
DataSource ds = new EncryptionDecorator(new CompressionDecorator(new FileDataSource()));
```

Decorator in Functional

Wrapping a base object

```
DataSource ds = new EncryptionDecorator(
        new AuthSignDecorator(
                new CompressionDecorator(
                        new FileDataSource()
                )
        )
);
```

Composing a Function

```
Function<FileDataSource, FileDataSource> process =
        compress
        .andThen(nonce)
        .andThen(authSign)
        .andThen(encrypt);
```