Design patterns

Software design patterns

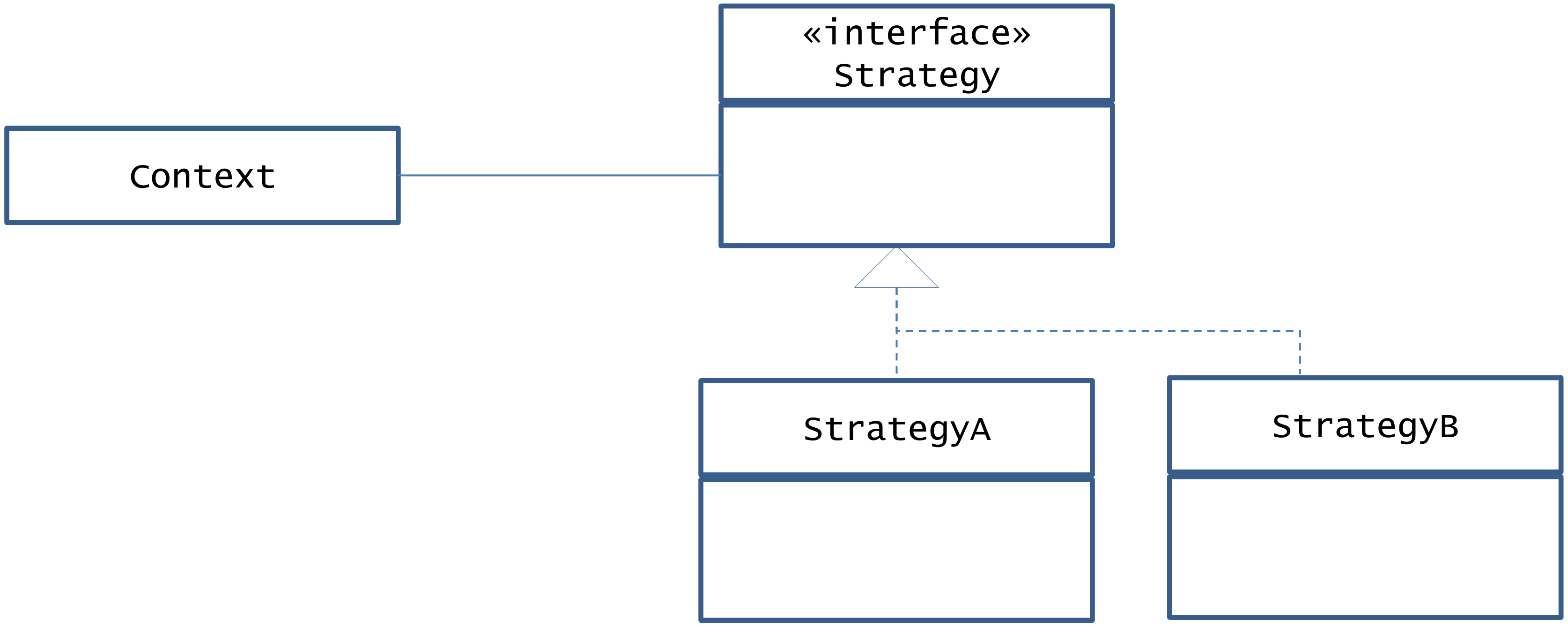
- Each pattern:
  - Describes a recurring problem
  - Captures the static and dynamic structure, as well as the collaboration between the main actors
  - Basic Categories:
    - "creational" -- "Simple" Factory, Factory Method, Abstract Factory, Singleton
    - "structural" -- Bridge, Composite, Proxy, ...
    - "behavioral" -- Command, Iterator, Strategy, Visitor, ...

Summary

1. Strategy
2. Command
3. Factory Method
4. Template Method
5. Singleton
6. Proxy
7. Adapter
8. Flyweight
9. Observer , Observed
10. Model-View-Controller
11. Façade
12. Composite

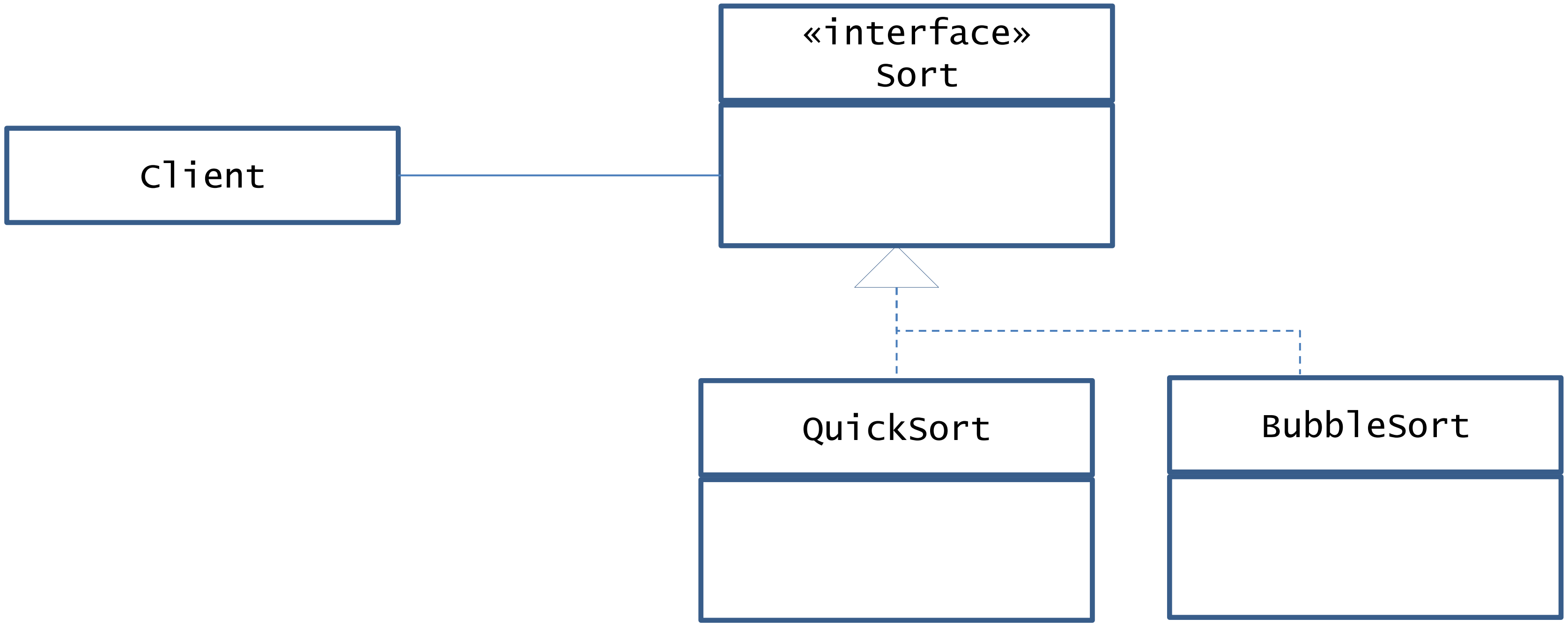
Strategy/Policy

- Concrete algorithm can vary independently of the client/context

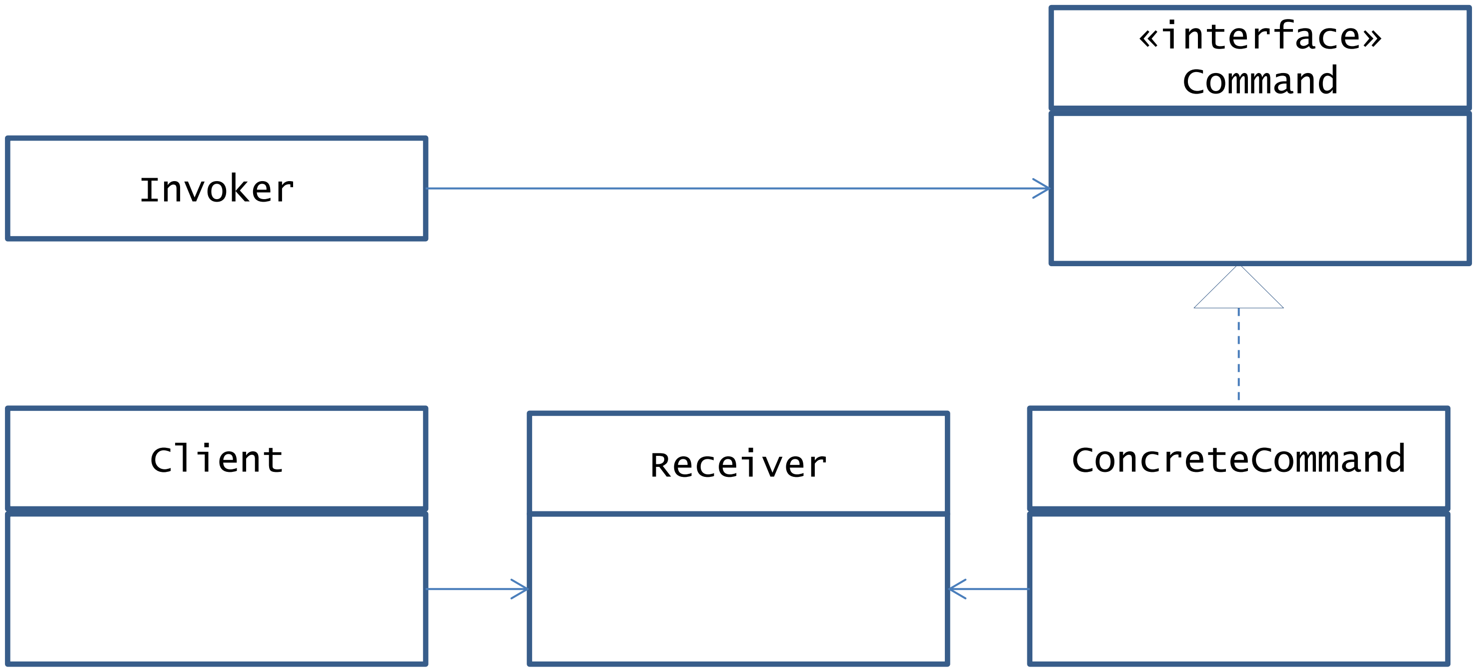


Example: *strategy*

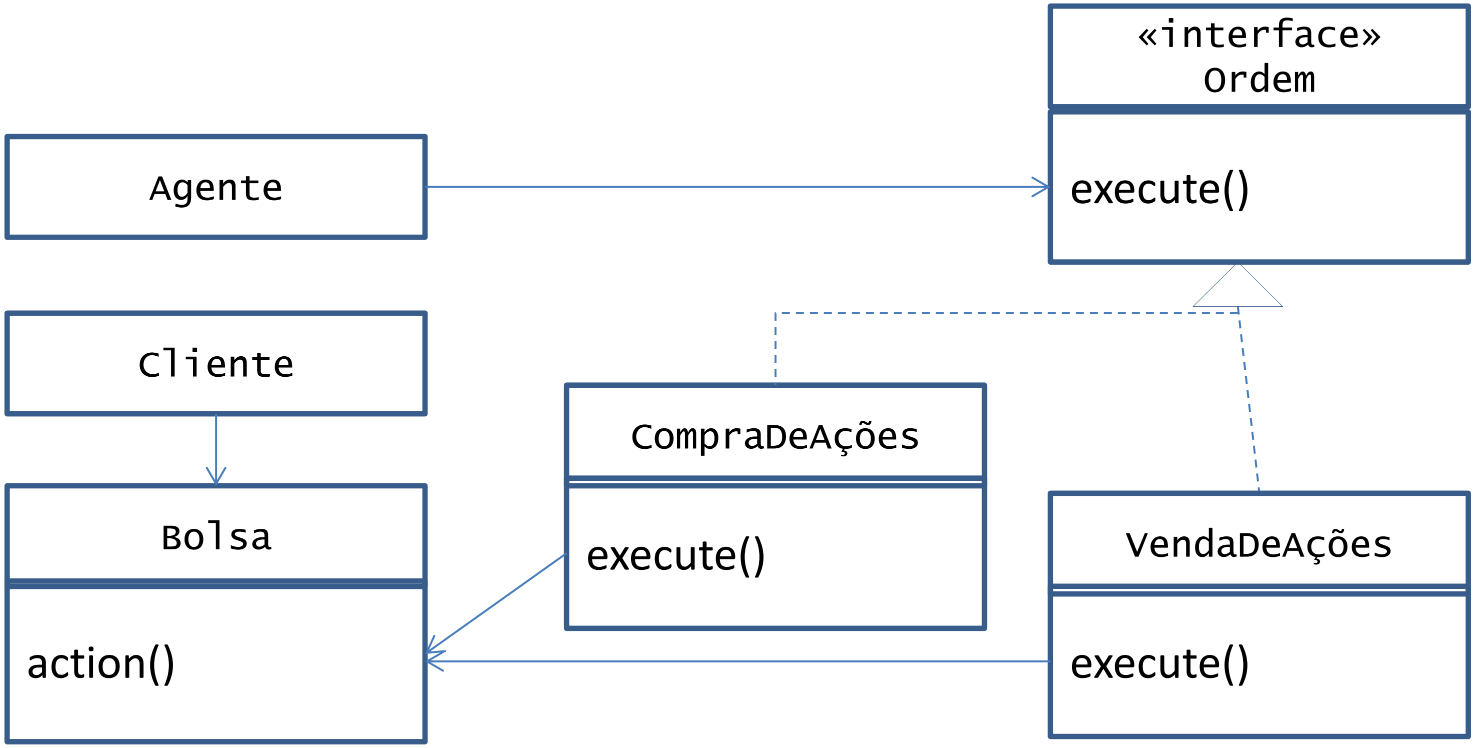
- Client calls sort() without knowing the type of algorithm that will be used



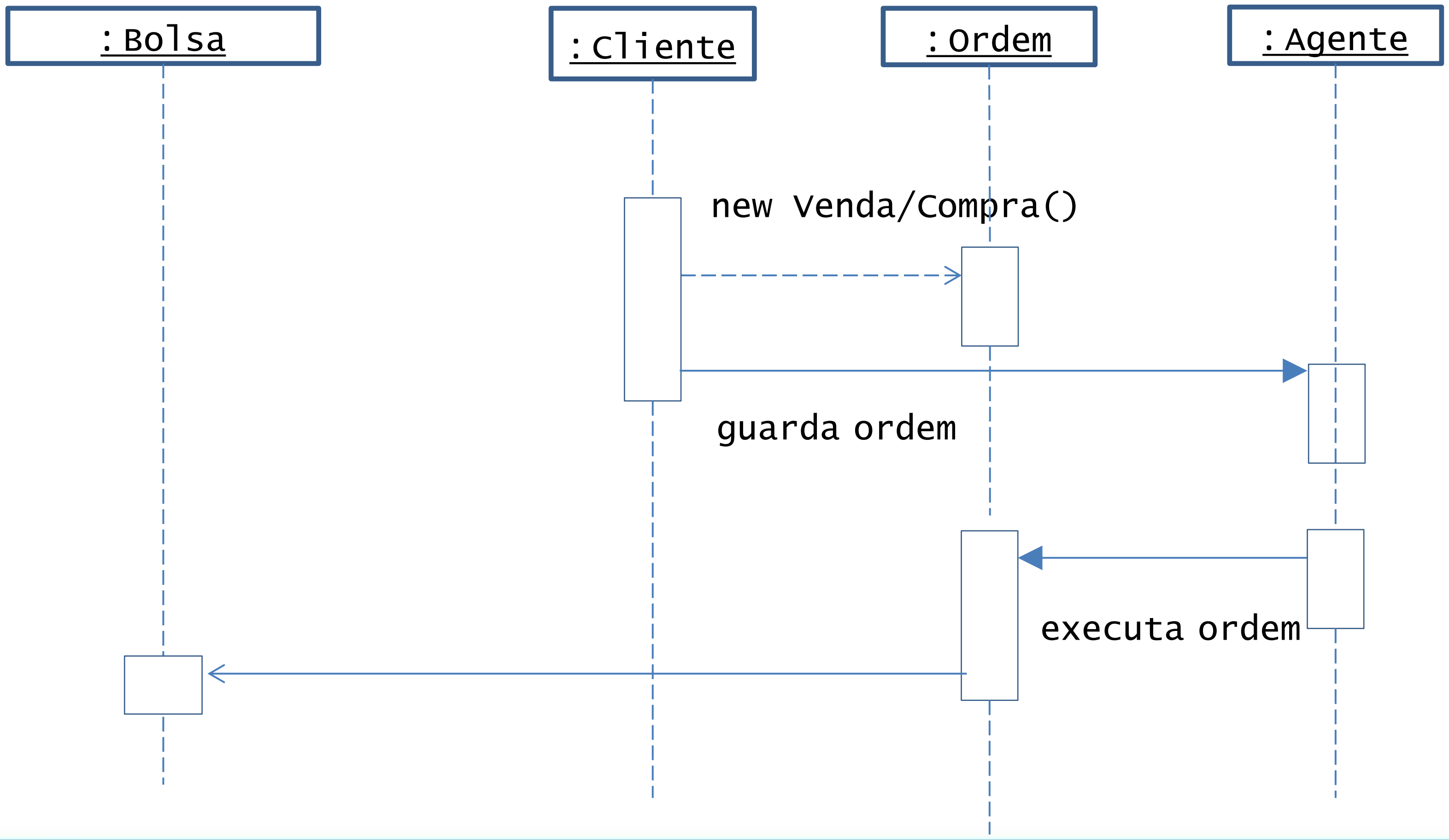
Command


«interface»
Command
Invoker
Client
Receiver
ConcreteCommand

Example: *command*

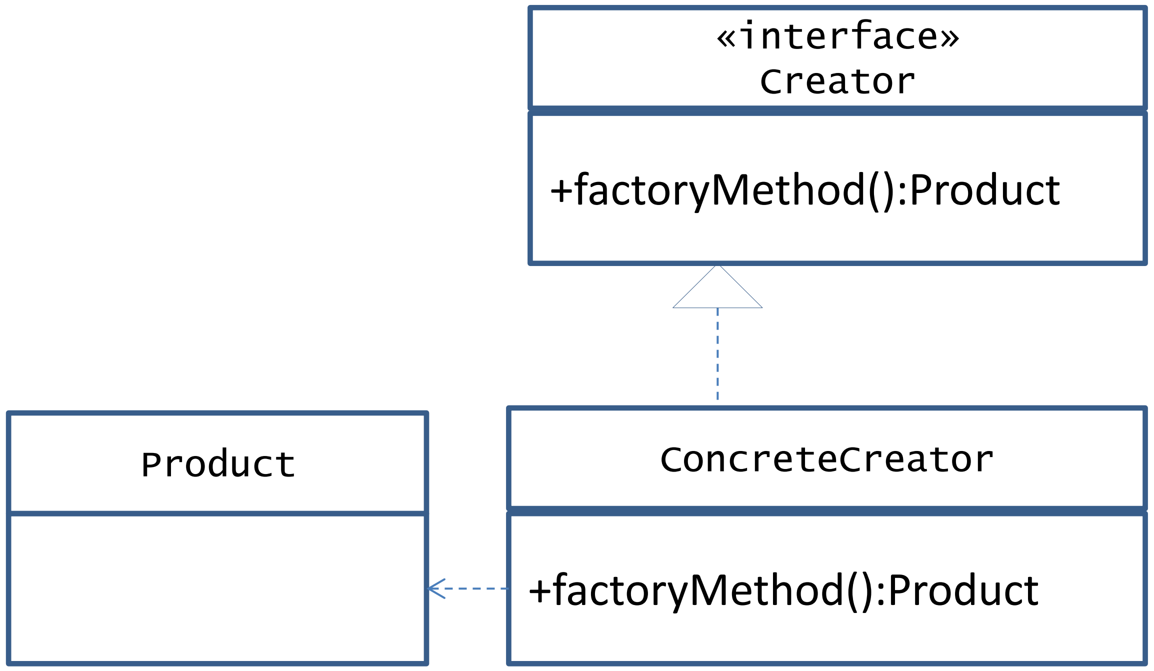


Example: *command*



Factory Method

- Creates object without class specification

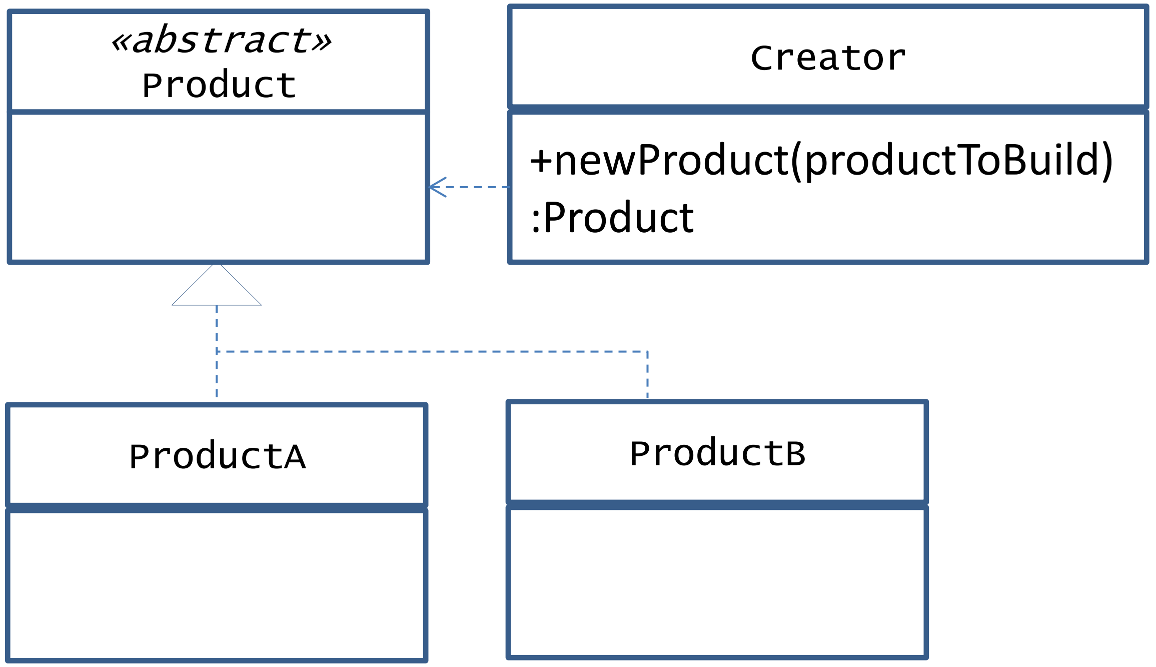


Factory Method

```
public abstract class Form {
     public abstract newForm();
     ...
}
Form a = new Circle(...);
Form b = new Rectangle(...);
Form c = a.newForm(); // Circle
Form c = b.newForm(); // Rectangle
```

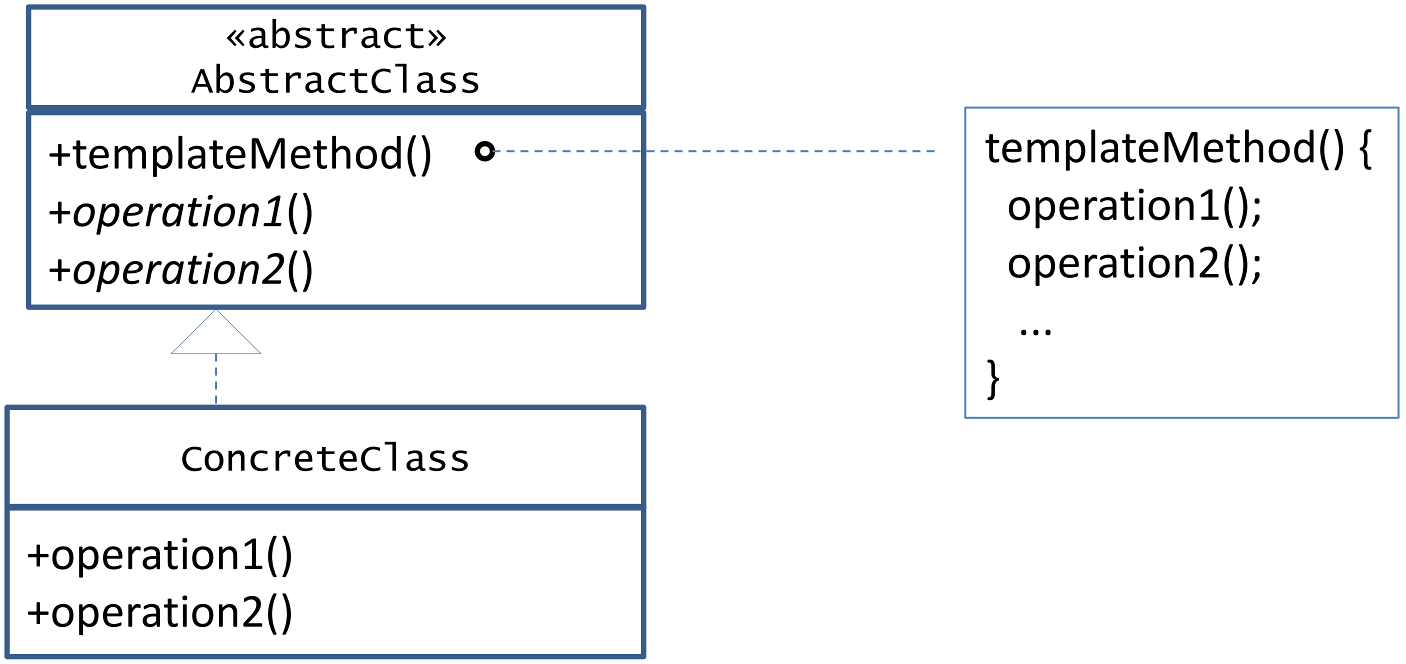
Factory

- Creates object without class specification



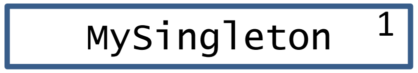
Template method

- Defines algorithm skeleton without defining details



```
templateMethod() {
  operation1();
  operation2();
   …
}
```

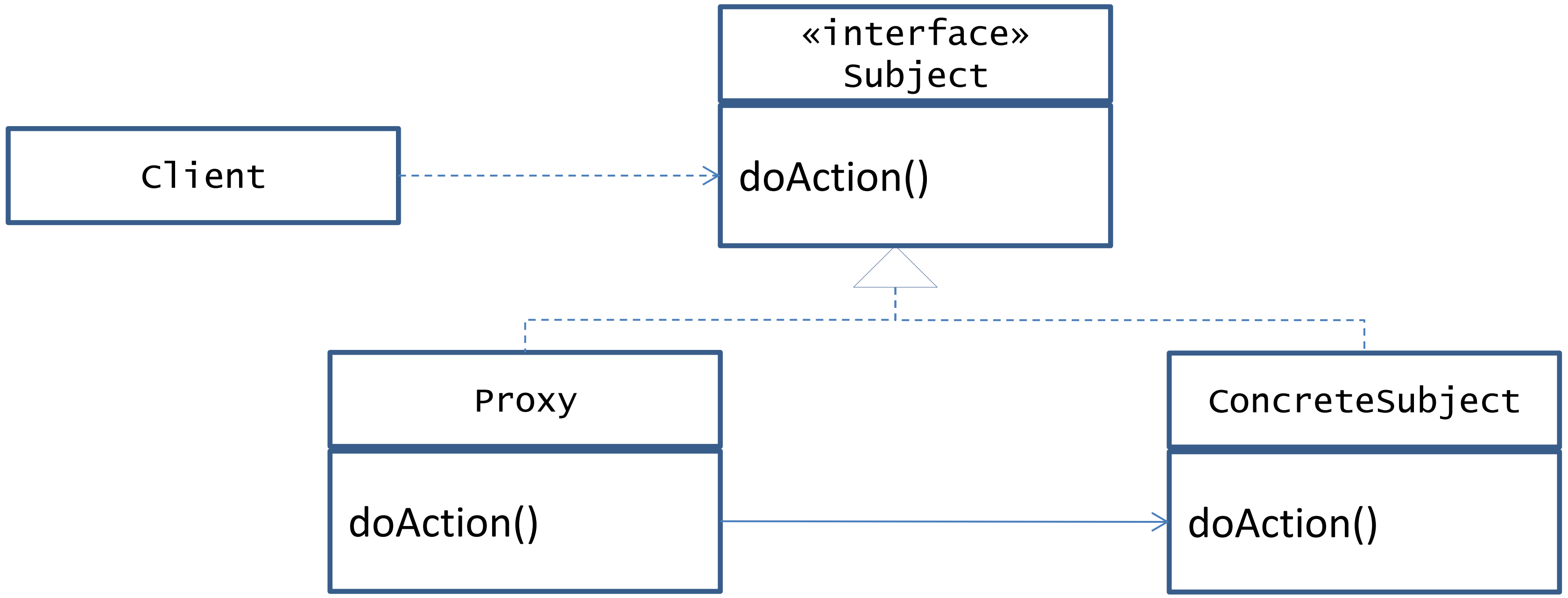
Singleton



```
package mypackage;

public final class MySingleton {

    private static final MySingleton INSTANCE =
        new MySingleton();

    private MySingleton() {
        assert INSTANCE == null : …;
    }

    public static MySingleton getInstance() {
        return INSTANCE;
    }    …
}
```

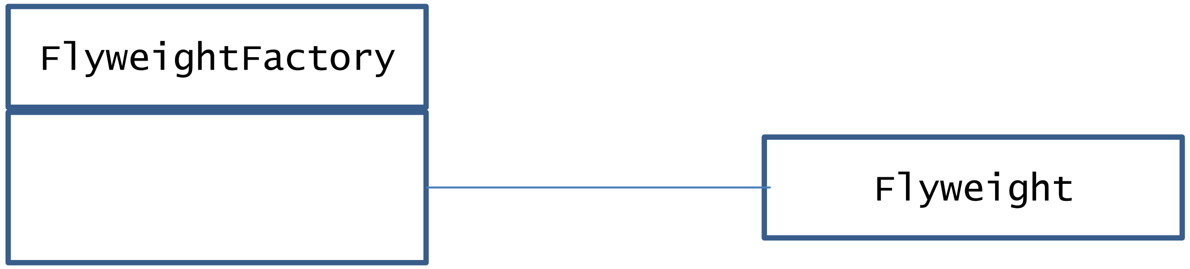
Proxy

- A class works as another's proxy, passing requests and returning replies, eventually limiting the access to the represented class



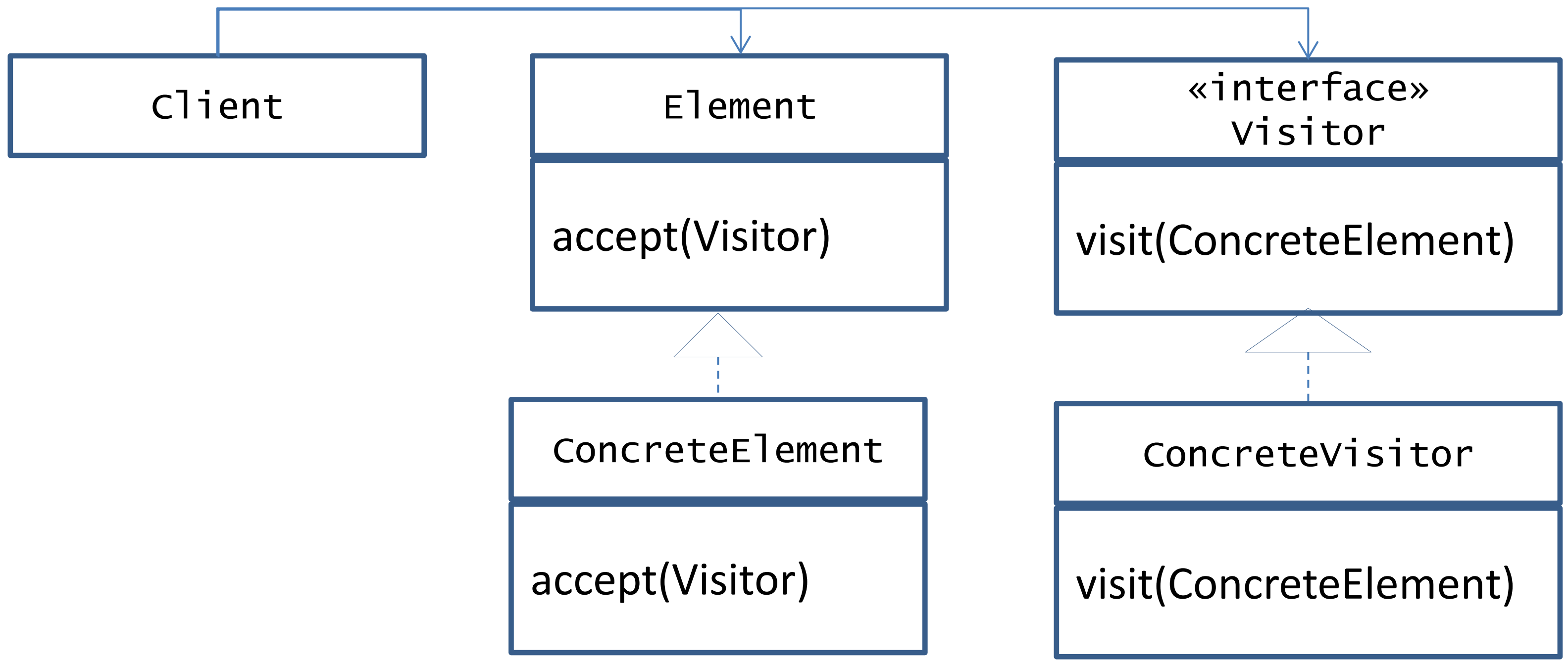
Flyweight

- Object that minimizes memory consumption by sharing resources.



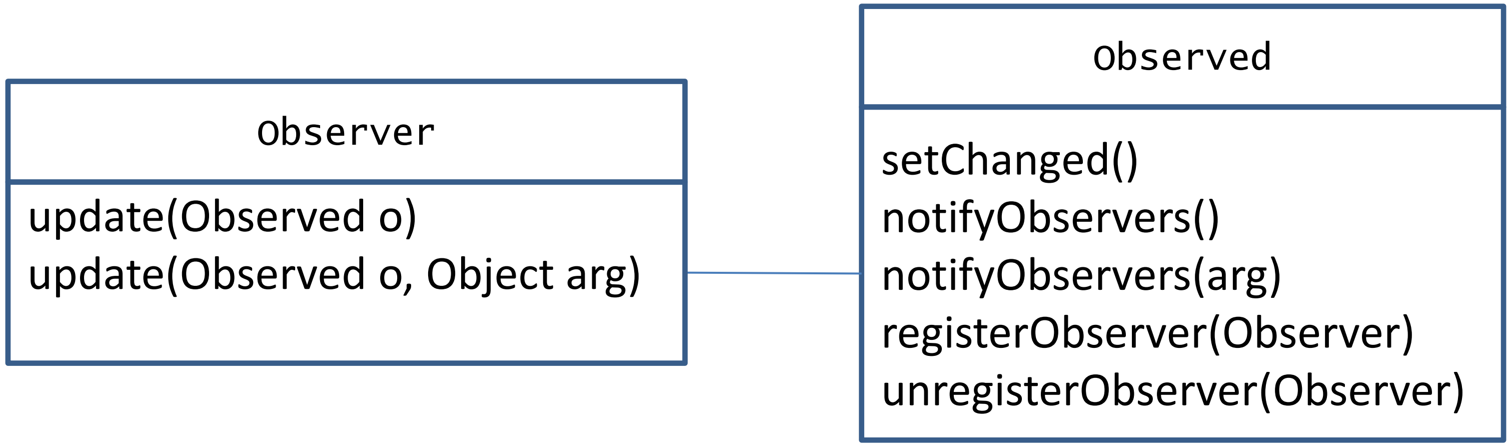
Visitor

- Separates the algorithm from the structure that uses it

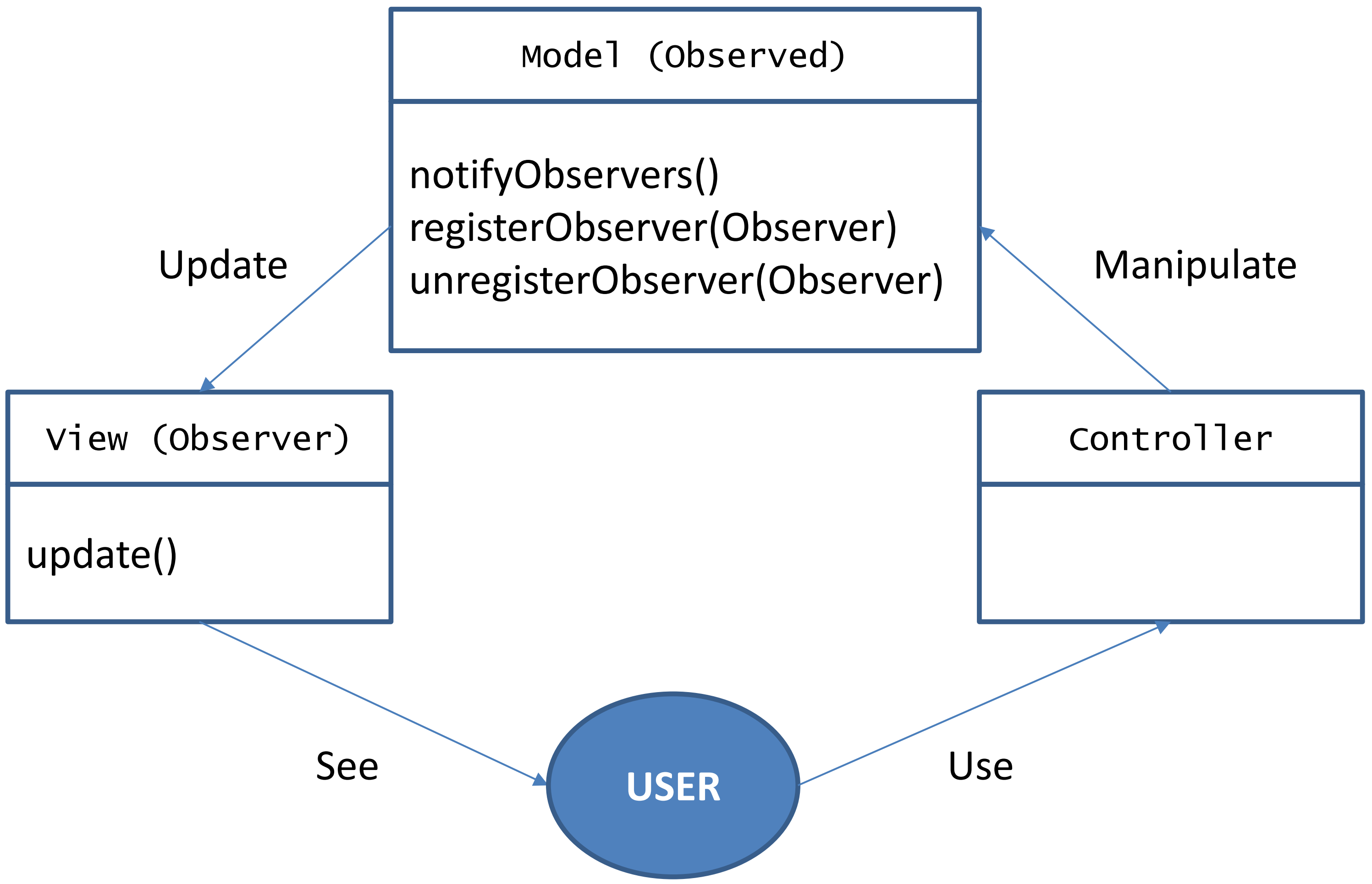


Observer - Observed

- Observed maintains list of observers and warns when there is a change

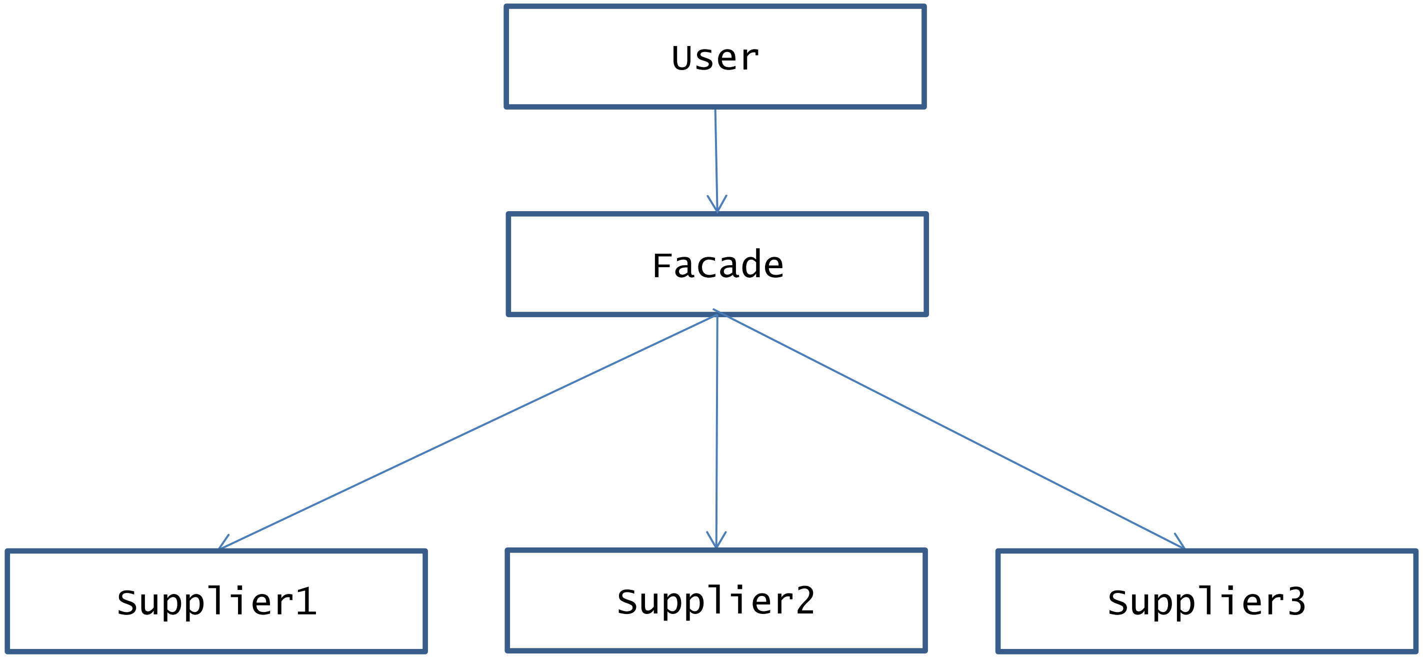

Observer
update(Observed o)
update(Observed o, Object arg)
Observed
setChanged()
notifyObservers()
notifyObservers(arg)
registerObserver(Observer)
unregisterObserver(Observer)

Model – View - Controller



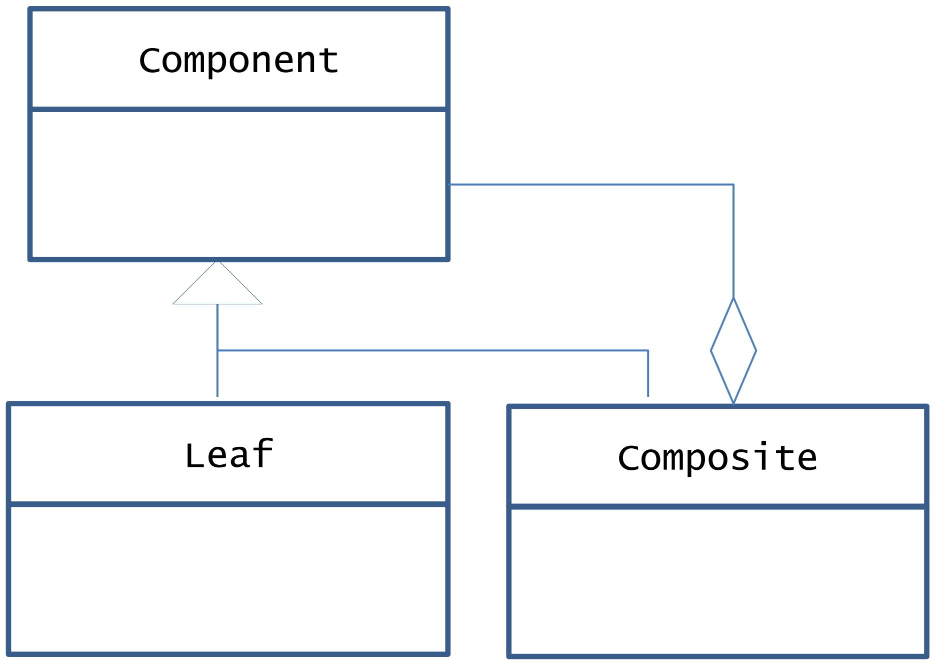
Façade

- Offers a simplified interface to a set of code



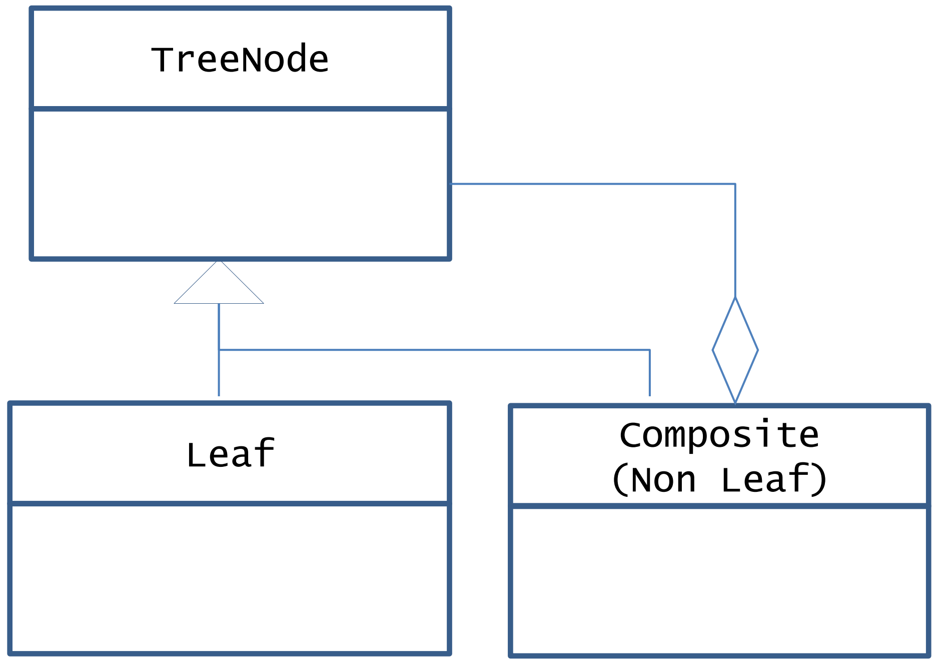
Composite

- Group of objects that can be treated as a single instance of an object



Example: Composite

- Tree



References

- Y. Daniel Liang, *Introduction to Java Programming*, 7.ª edição, Prentice-Hall, 2008.
- Gamma, Helm, Johnson & Vlissides, *Design Patterns*. Addison-Wesley. ISBN 0-201-63361-2, 1994.
- Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, *Head First Design Patterns*, O'Reilly Media, October 2004