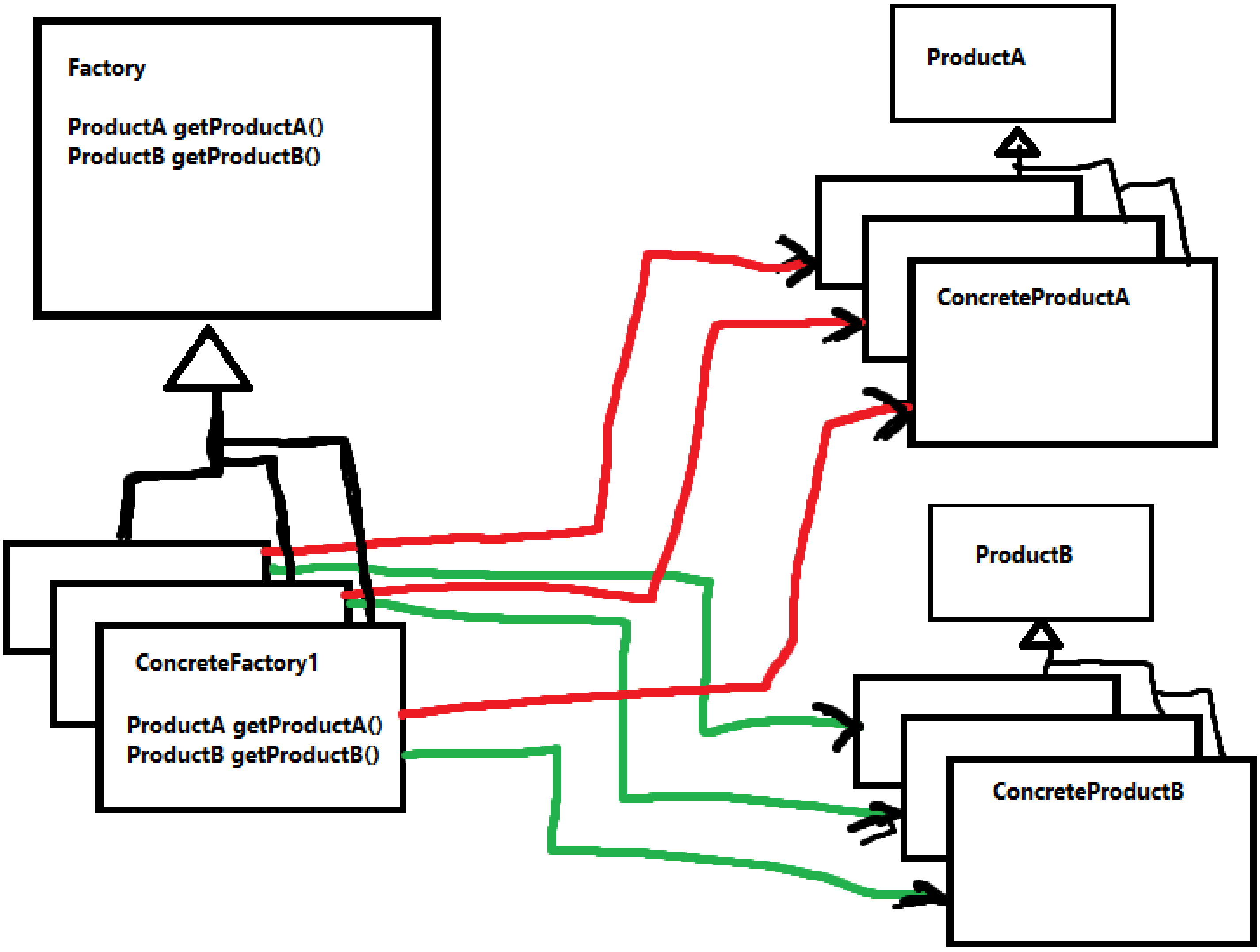
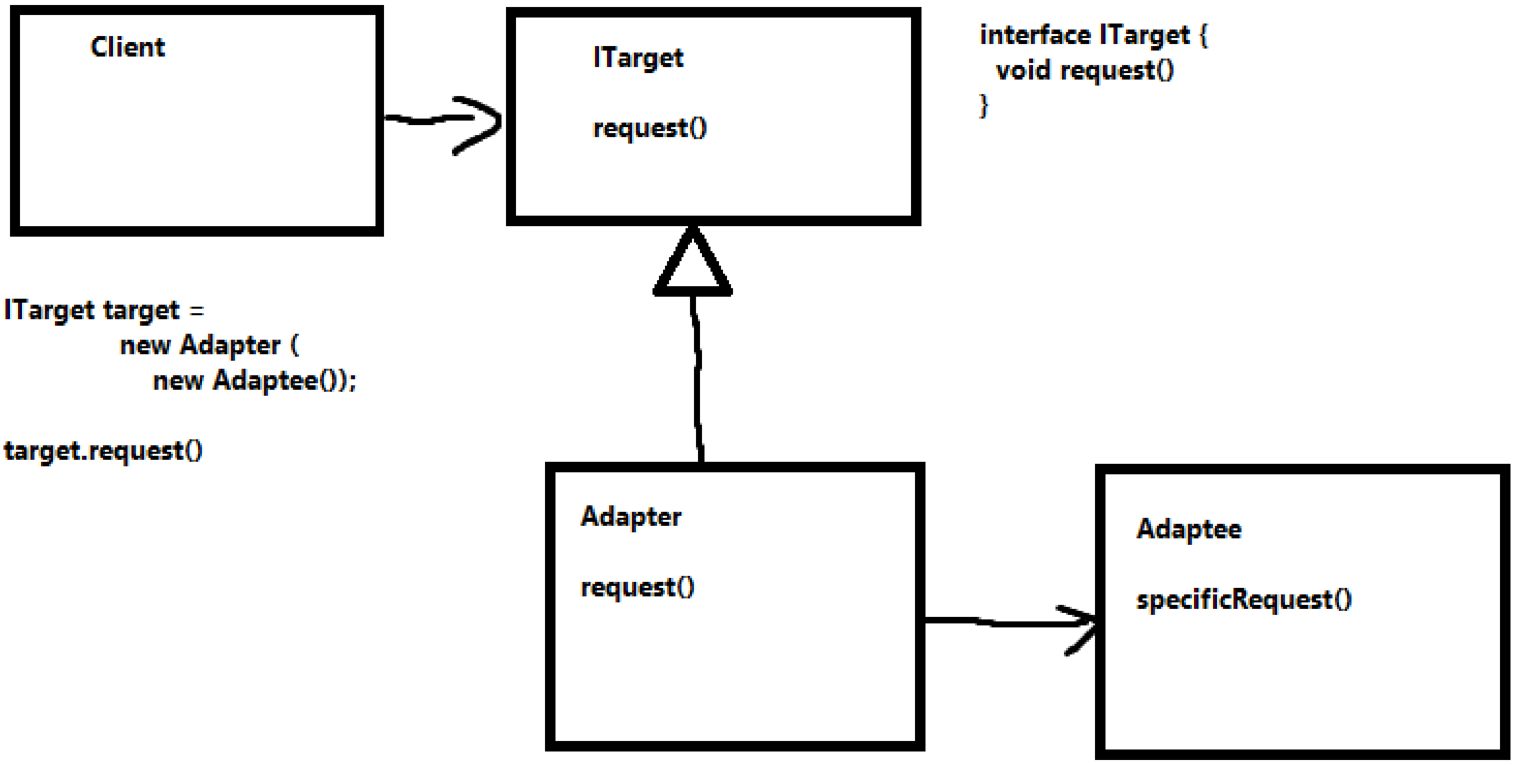


1. ABSTRACT FACTORY



```
interface ITarget {
  void request()
}
```

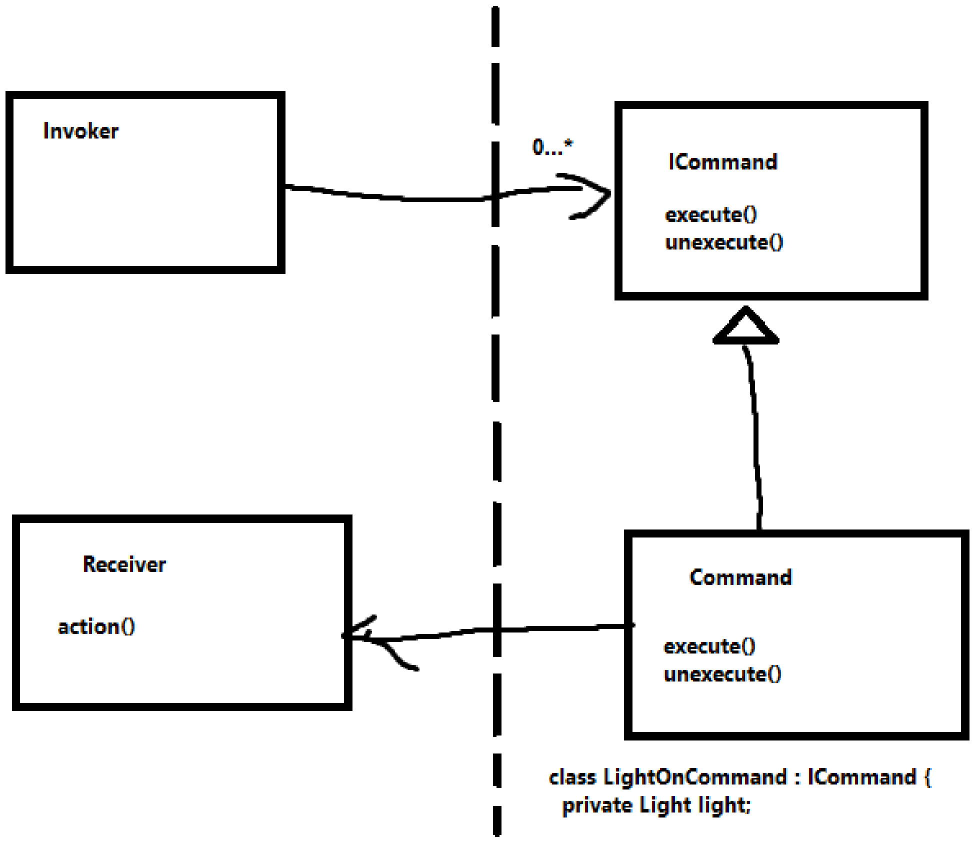
```
ITarget target =
        new Adapter (
          new Adaptee());

target.request()
```

```
class Adapter : ITarget {

 private Adaptee adaptee;

 public Adapter(Adaptee a){
  this.adaptee = a;

 public void request(){
  this.adaptee.specificRequest()
 }
}
```

```
class Adaptee {
 public void specificRequest

   ........
  }
 }
```

1. ADAPTER

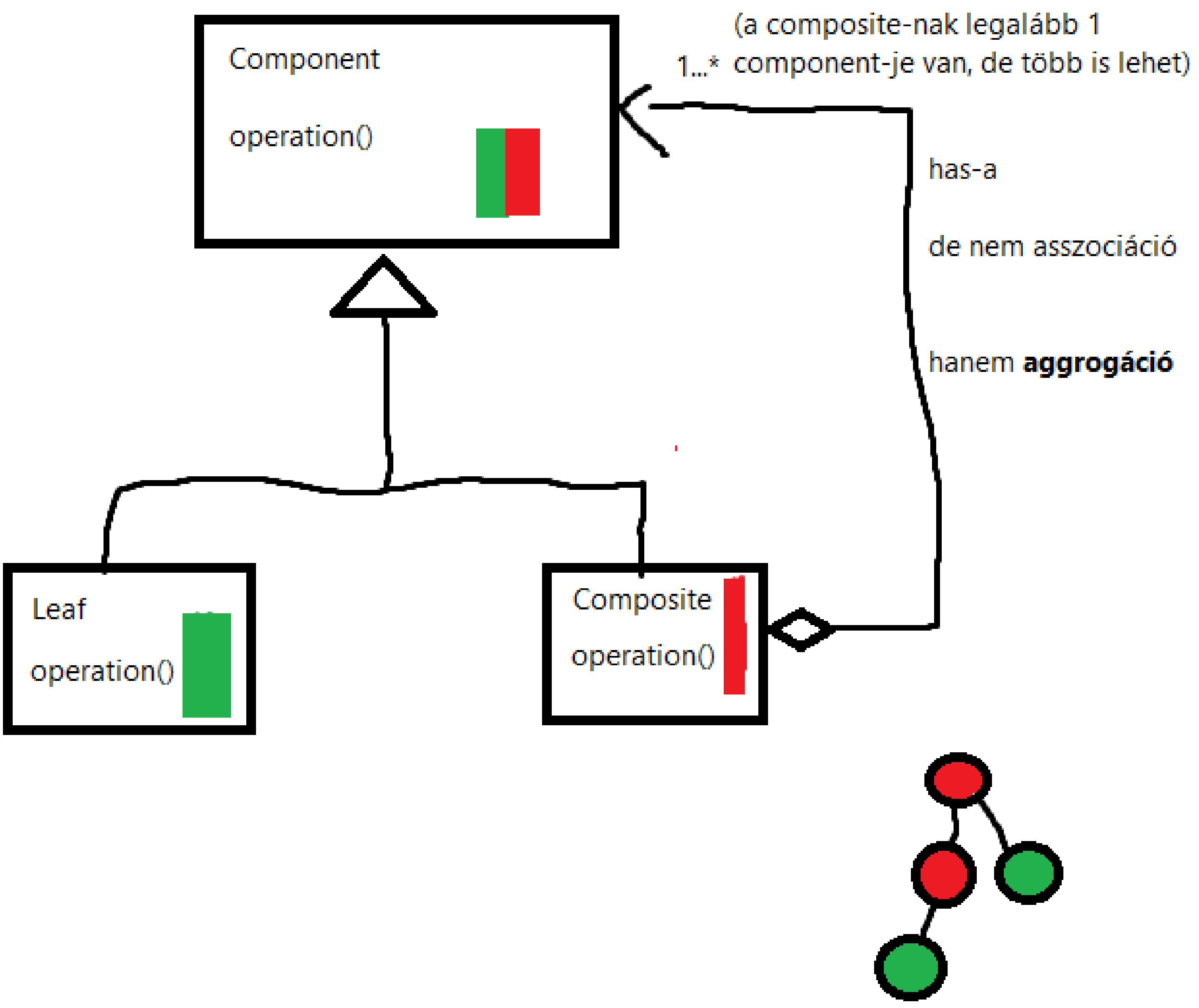
```
class Invoker {

  ICommand on;
  ICommand off;

  public Invoker (ICommand on, ICommand off){
    this.on = on;
    this.off = off;
  }

  public void clickOn(){
    this.on.execute();
  }

  public void clickOff(){
    this.off.execute();
  }
```



```
class LightOnCommand : ICommand {
  private Light light;

  public LightOnCommand(Light l) {
    this.light = l;
  }

  public void execute() {
    this.light.on()
  }

  public void unexecute() {
    this.light.off()
  }
}
```

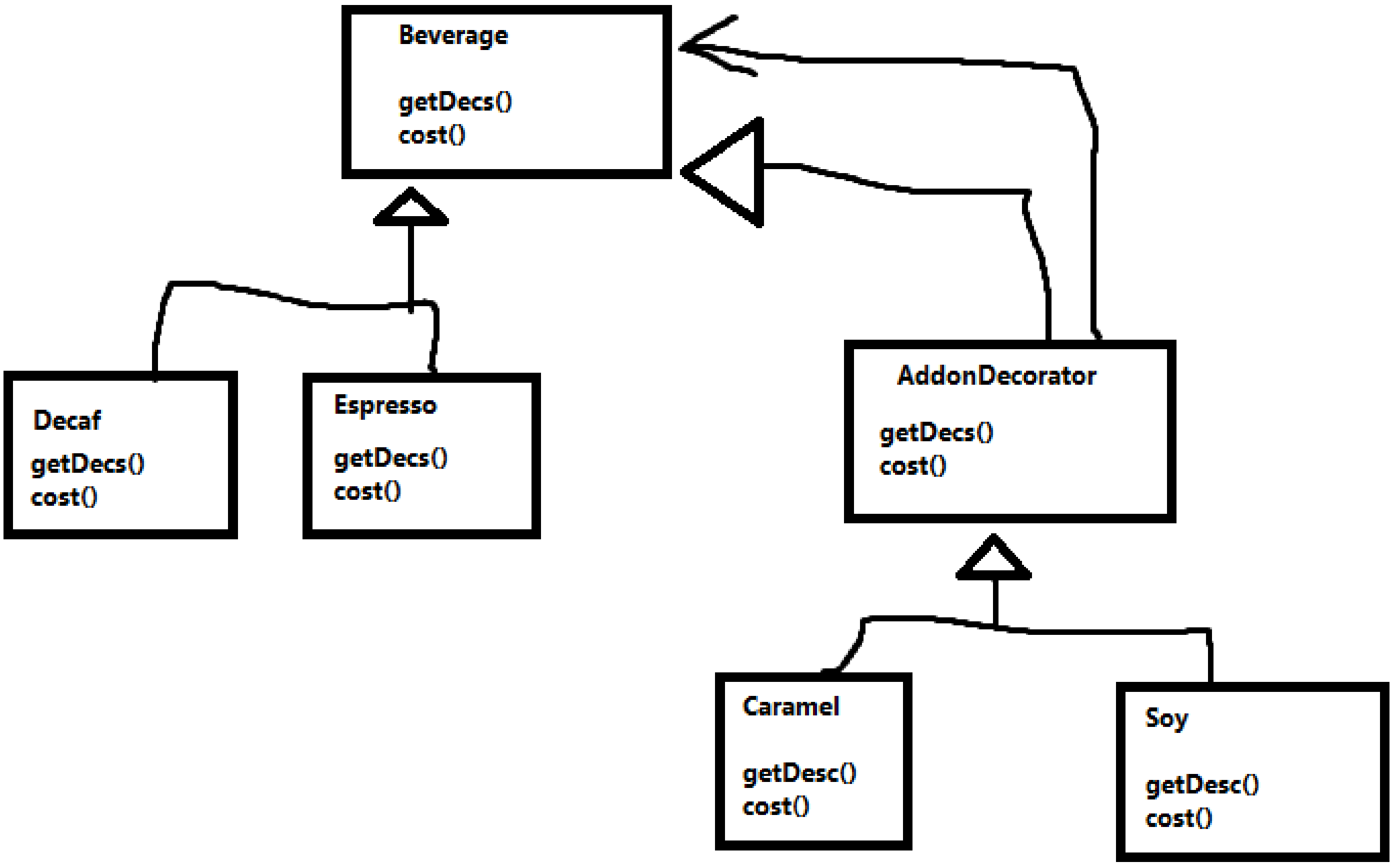
```
new Invoker (new LightOnCommand(light), new LightOffCommand(light))
```

1. COMMAND

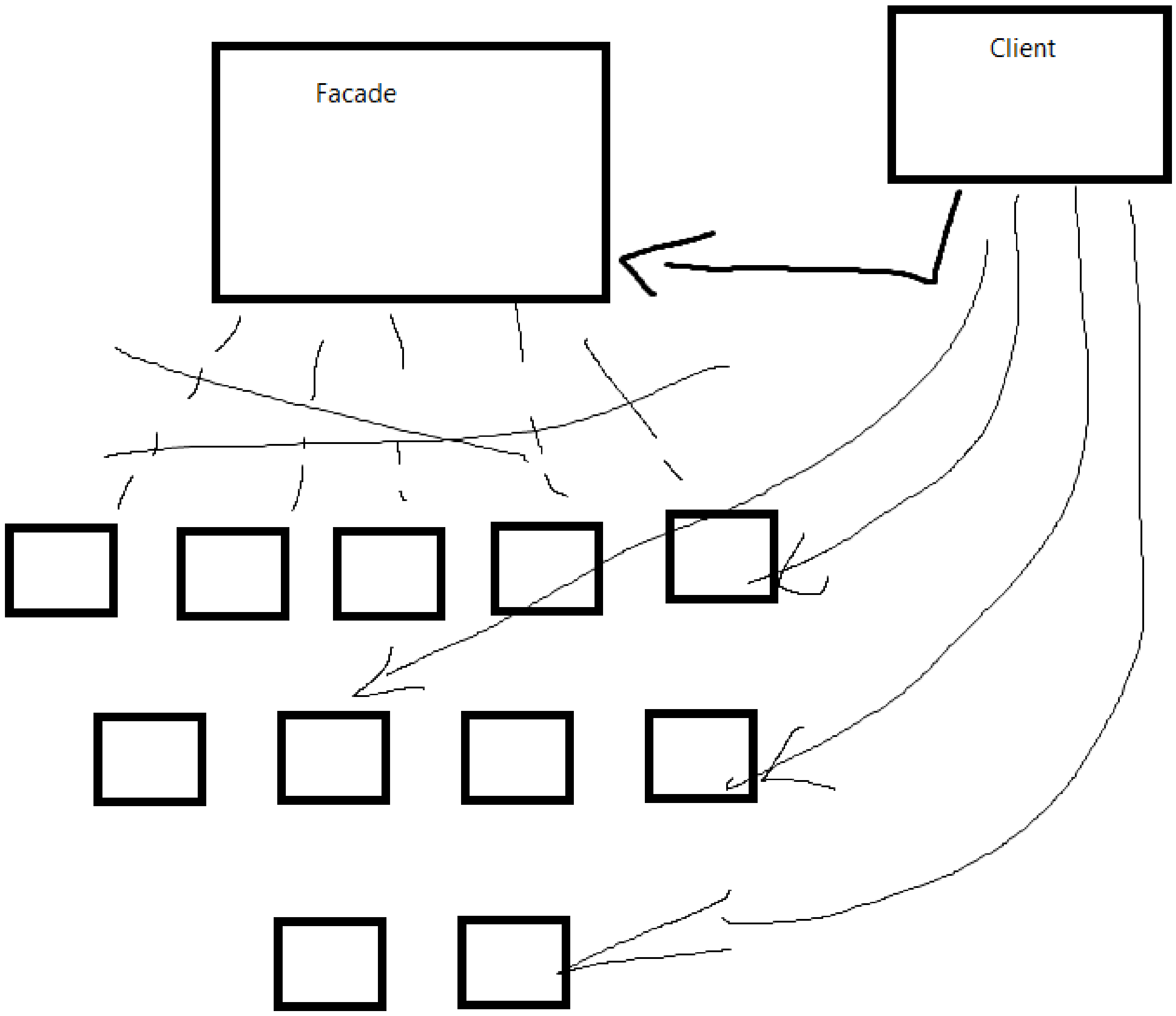


(a composite-nak legalább 1
1...* component-je van, de több is lehet)
Component
operation()
has-a
de nem asszociáció
hanem **aggrogáció**
Leaf
operation()
Composite
operation()

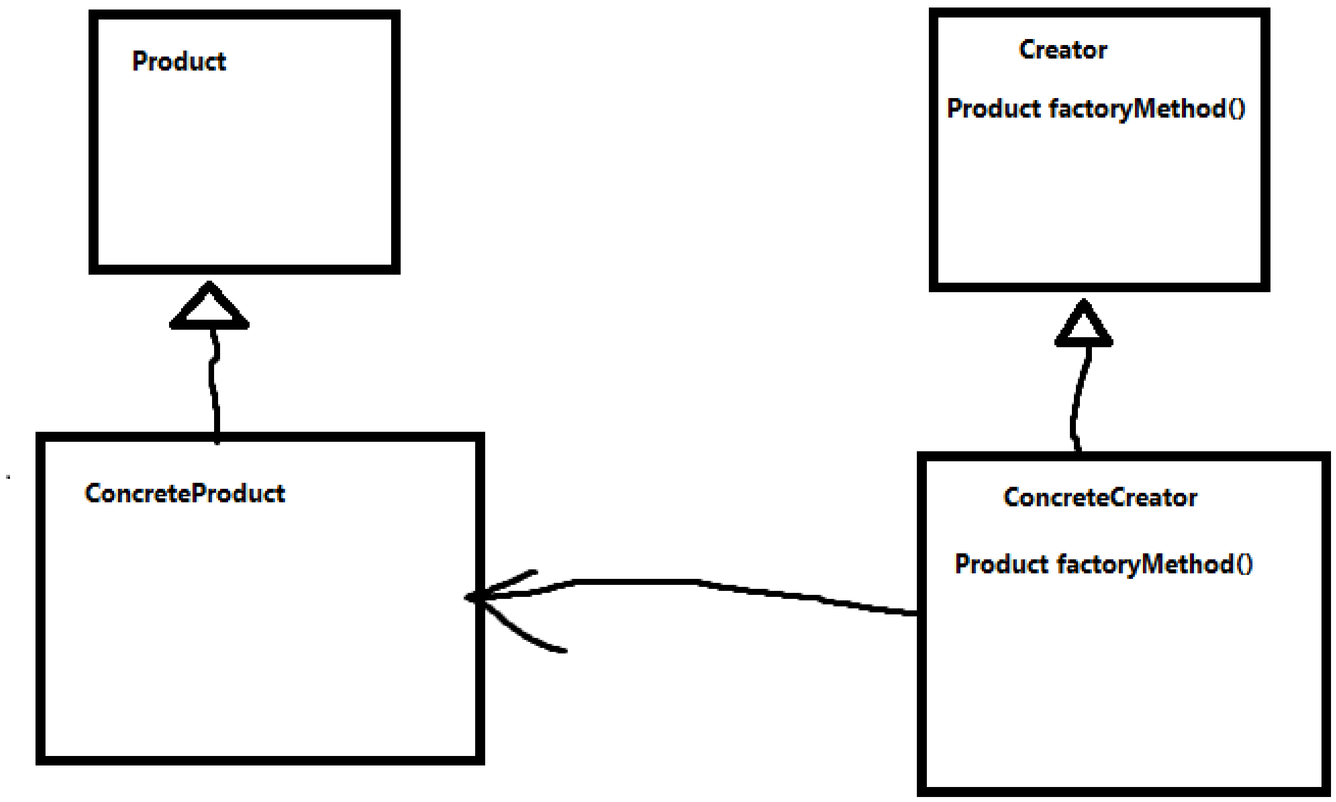
1. COMPOSITE


Beverage
getDecs()
cost()
Decaf
getDecs()
cost()
Espresso
getDecs()
cost()
AddonDecorator
getDecs()
cost()
Caramel
getDesc()
cost()
Soy
getDesc()
cost()

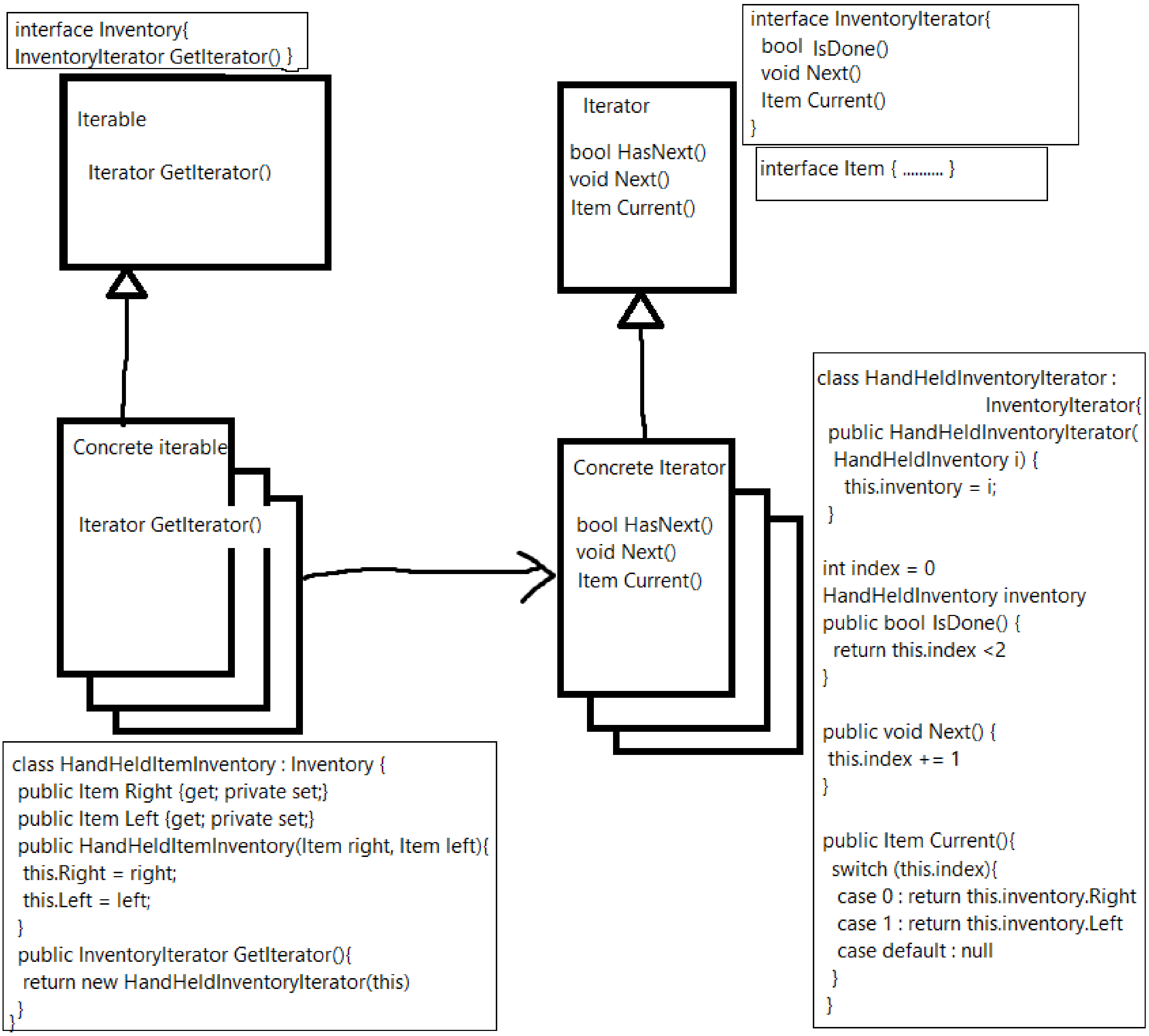
1. DECORATOR



1. FACADE


Product
Creator
Product factoryMethod()
ConcreteProduct
ConcreteCreator
Product factoryMethod()

1. FACTORY



```
interface Inventory{
InventoryIterator GetIterator() }
```

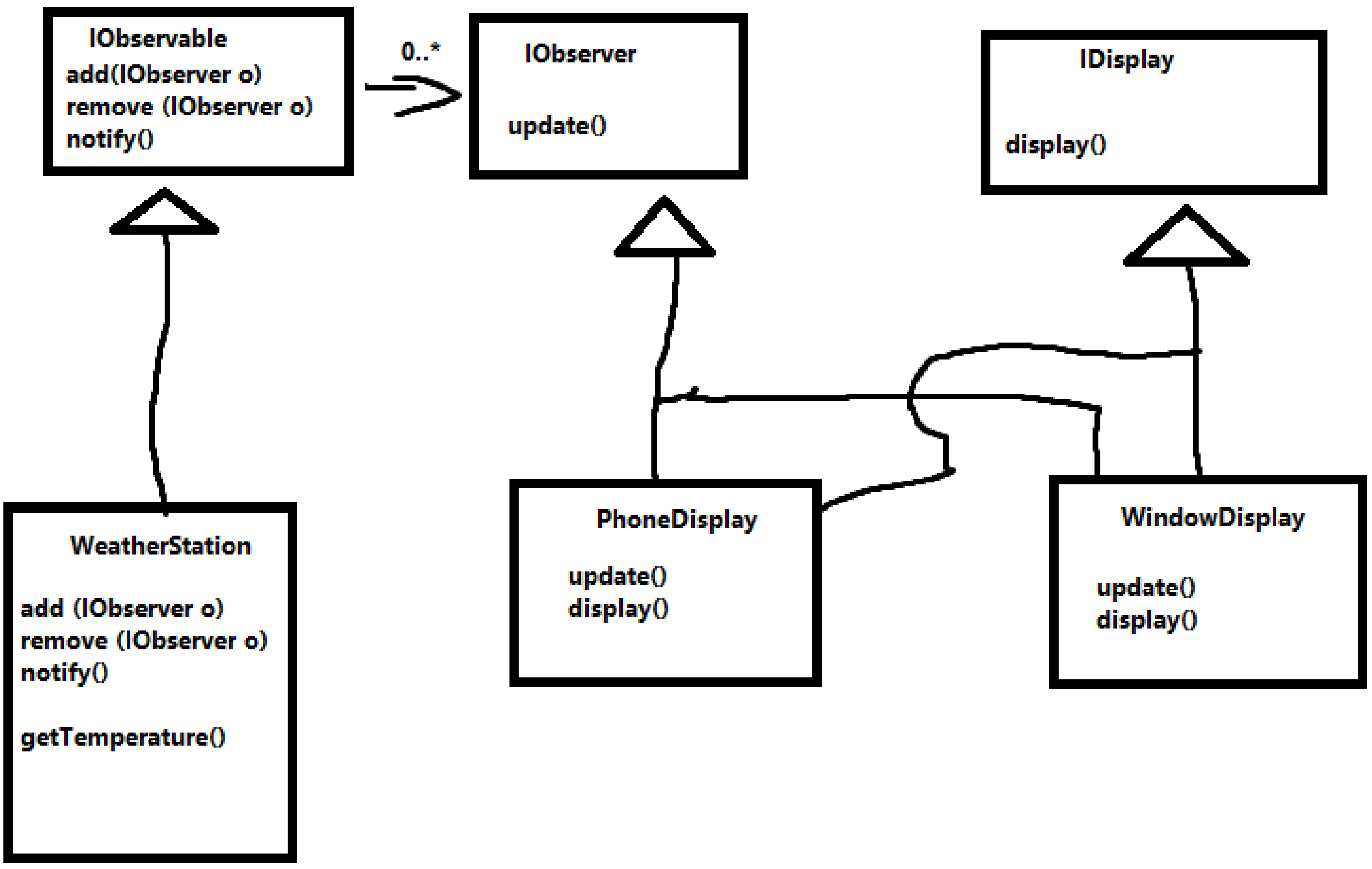
```
interface InventoryIterator{
  bool  IsDone()
  void Next()
  Item Current()
}
```

```
interface Item { .......... }
```

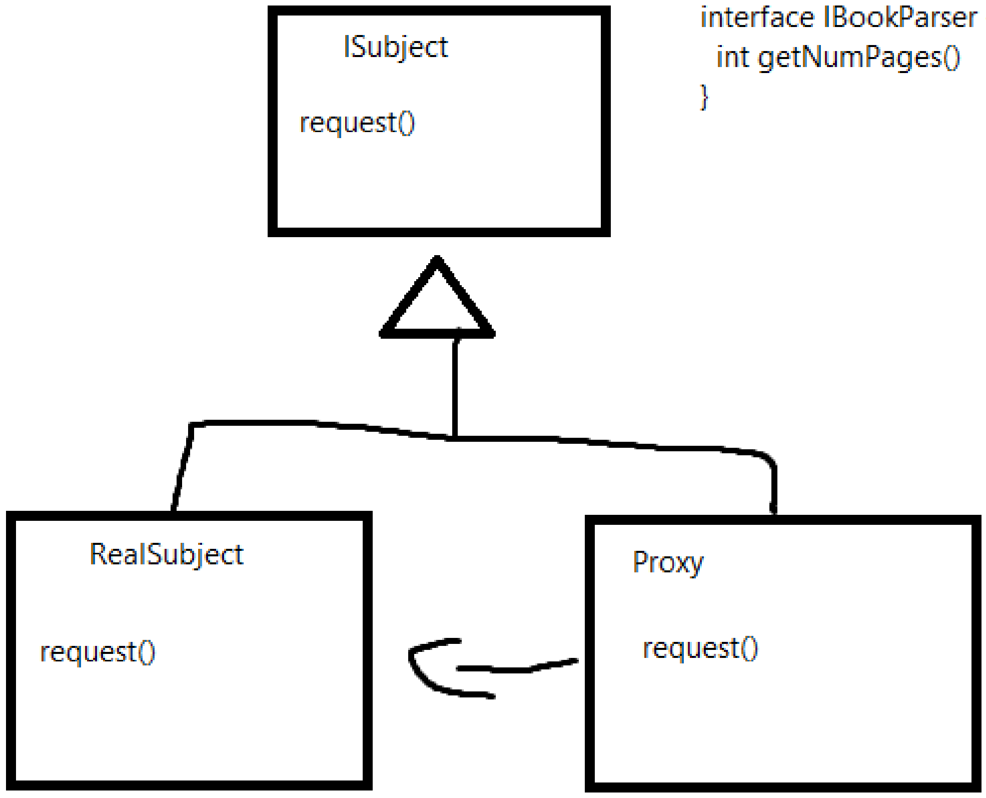
```
class HandHeldItemInventory : Inventory {
 public Item Right {get; private set;}
 public Item Left {get; private set;}
 public HandHeldItemInventory(Item right, Item left){
  this.Right = right;
  this.Left = left;
 }
 public InventoryIterator GetIterator(){
  return new HandHeldInventoryIterator(this)
 }
}
```

```
class HandHeldInventoryIterator :
                              InventoryIterator{
 public HandHeldInventoryIterator(
  HandHeldInventory i) {
    this.inventory = i;
 }

int index = 0
HandHeldInventory inventory
public bool IsDone() {
  return this.index <2
}

public void Next() {
 this.index += 1
}

public Item Current(){
 switch (this.index){
  case 0 : return this.inventory.Right
  case 1 : return this.inventory.Left
  case default : null
 }
}
```

1. ITERATOR


IObservable
add(IObserver o)
remove (IObserver o)
notify()
0..*
IObserver
update()
IDisplay
display()
WeatherStation
add (IObserver o)
remove (IObserver o)
notify()
getTemperature()
PhoneDisplay
update()
display()
WindowDisplay
update()
display()

1. OBSERVER



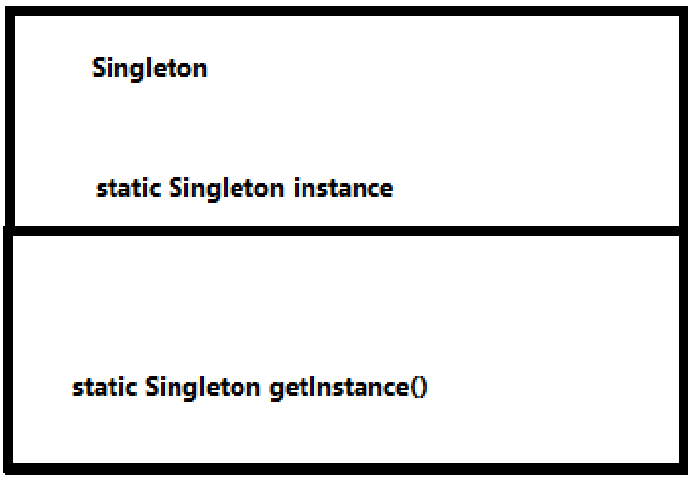
```
interface IBookParser {
  int getNumPages()
}
```

```
class BookParser : IBookParser {
  public BookParser (string book) {
    ...//expensive parsing...
  }

  public int getNumPages() {
    ...
  }
}
```

```
class LazyBookParserProxy : IBookParser {
  private BookParser parser = null
  private string book = null;

  public LazyBookParserProxy(string book){
    this.book = book
  }

  public int getNumPages() {
    if (this.parser == null) {
      this.parser = new BookParser(this.book)
    return this.parser.getNumPages()
  }
```

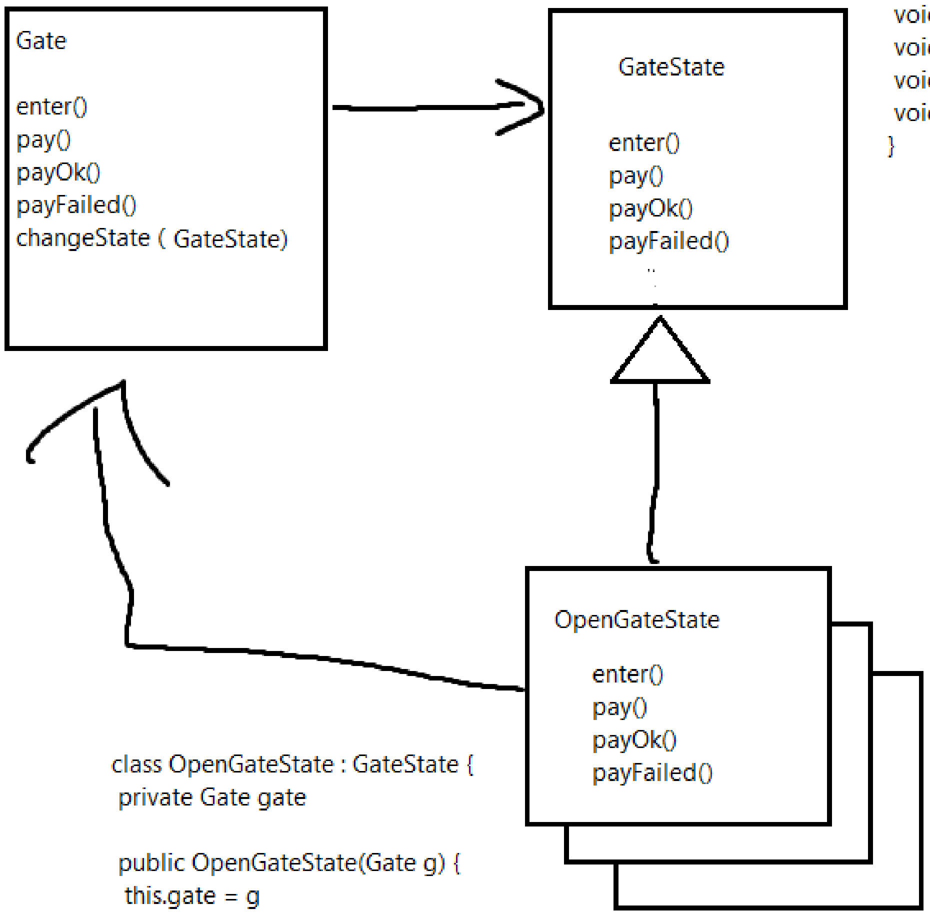
1. PROXY

```
class Singleton {
 static private Singleton instance

 private Singleton () {}

 public static Singleton getInstance() {
  if (instance == null) {
    instance = new Singleton()
  }
  return instance
 }
.
.
.
.
.
.
}
```



1. SINGLETON

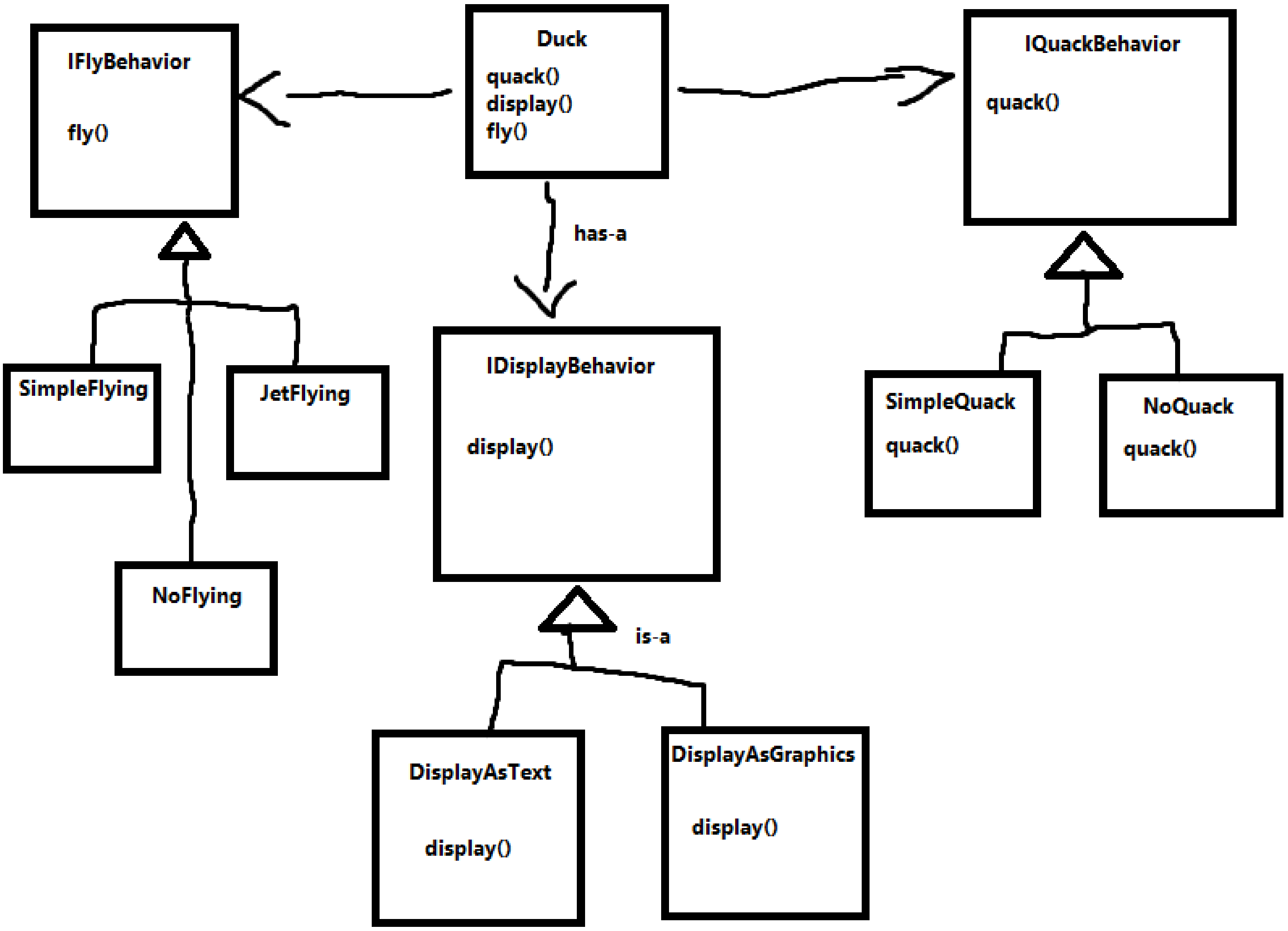
```
class Gate {
 private GateState state
 public Gate() {
  this.state = new ClosedGateState(this)
 }

 public void pay() {
  this.state.pay()
 }

 public void payOk() {
  this.state.payOk()
 }

 public void payFailed() {
  this.state.payFailed()
 }

 public void changeState (GateState s) {
  this.state = s
 }
```



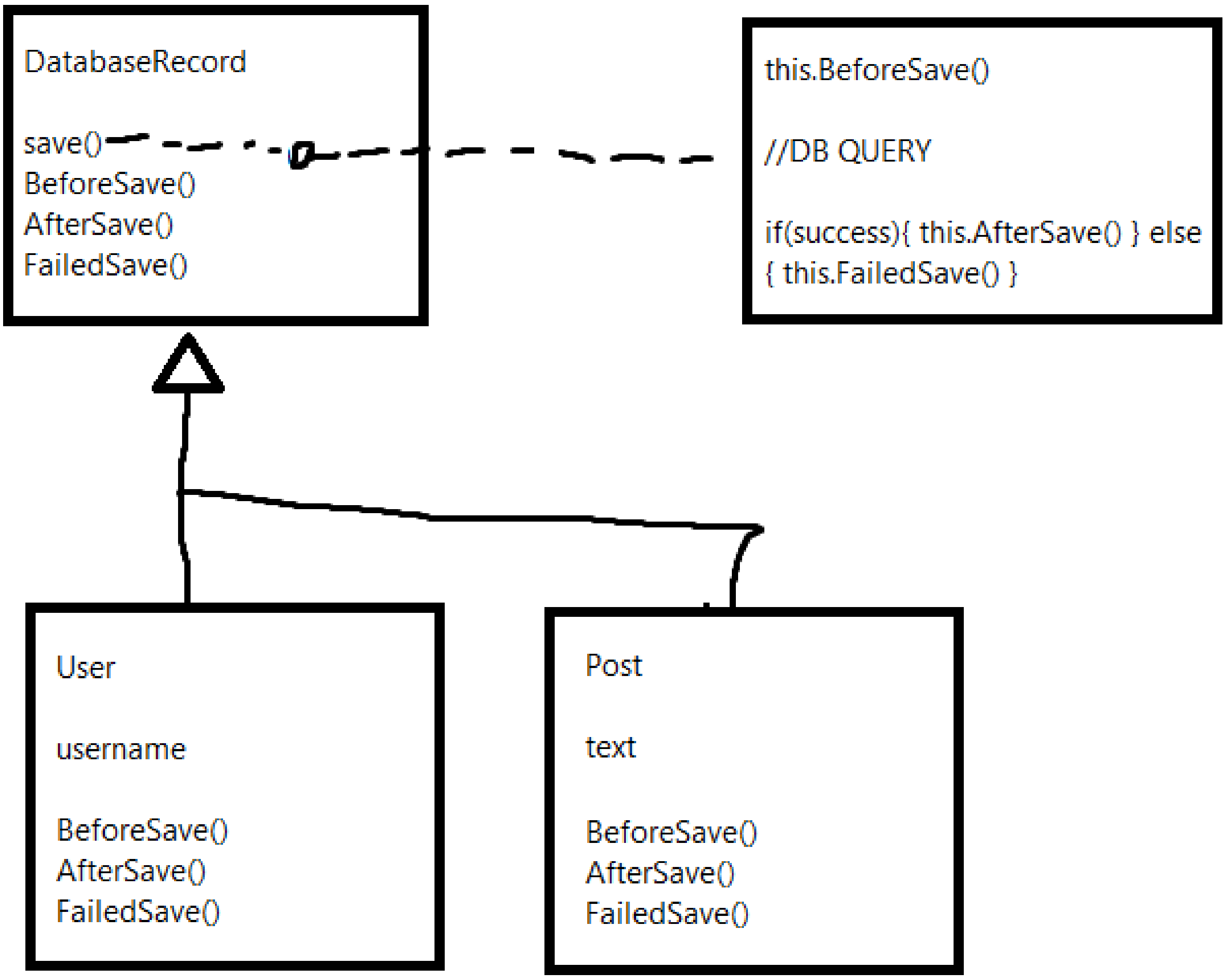
```
interface GateState {
 void enter()
 void pay()
 void payOk()
 void payFailed()
}
```

```
class OpenGateState : GateState {
 private Gate gate

 public OpenGateState(Gate g) {
  this.gate = g
 }

 public void payOk() {
  //let user in...
  this.gate.changeState (new ClosedGateState (this.gate))
 }
 ....
}
```

1. STATE


Duck
quack()
display()
fly()
IFlyBehavior
fly()
IQuackBehavior
quack()
has-a
IDisplayBehavior
display()
SimpleFlying
JetFlying
NoFlying
SimpleQuack
quack()
NoQuack
quack()
is-a
DisplayAsText
display()
DisplayAsGraphics
display()

1. STRATEGY



1. TEMPLATE METHOD