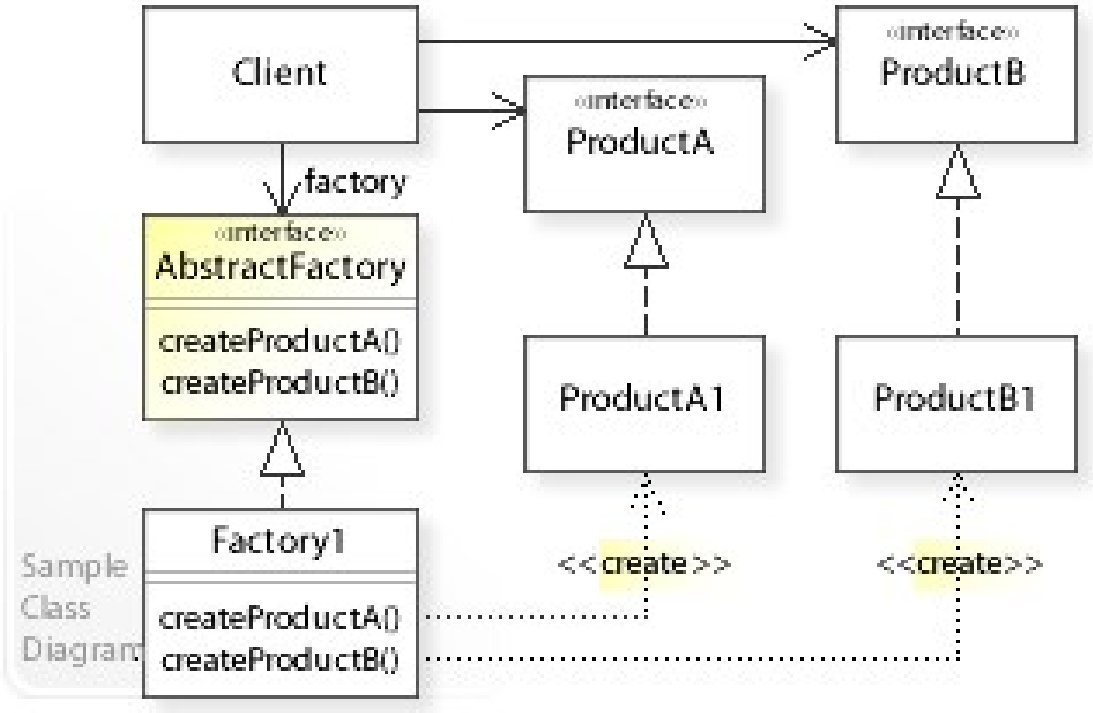
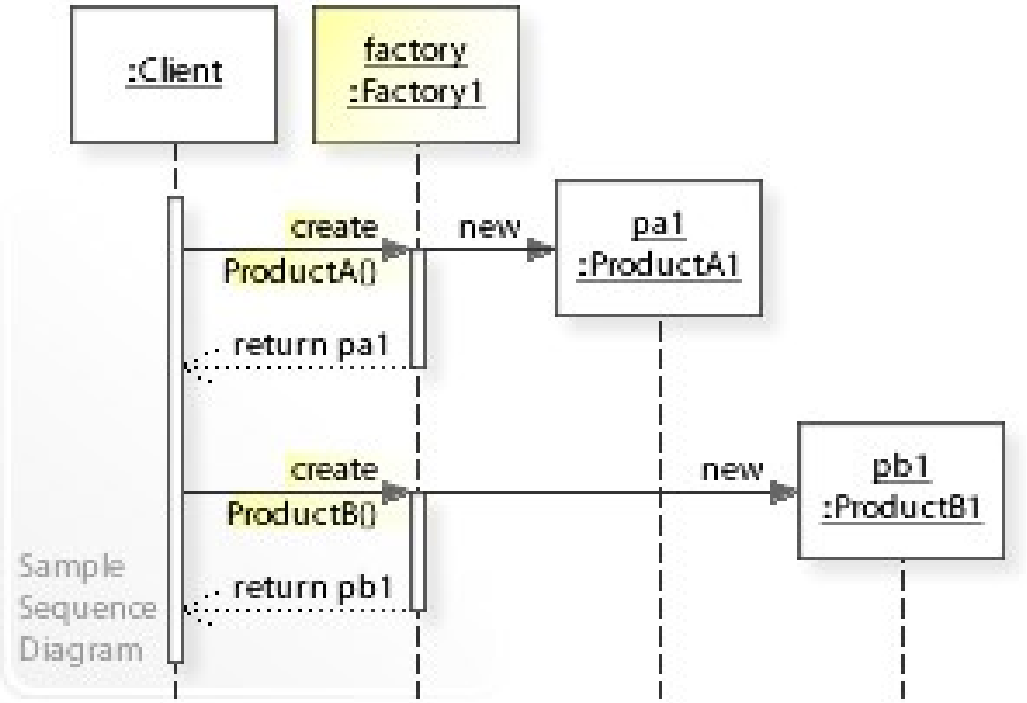
Steve Hostettler

Design Patterns

Design Patterns

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides,*Design Patterns: Elements of Reusable Object-Oriented Software*,Pearson Education - 1994, (ISBN 9780321700698)





Comment décrire un “design pattern”

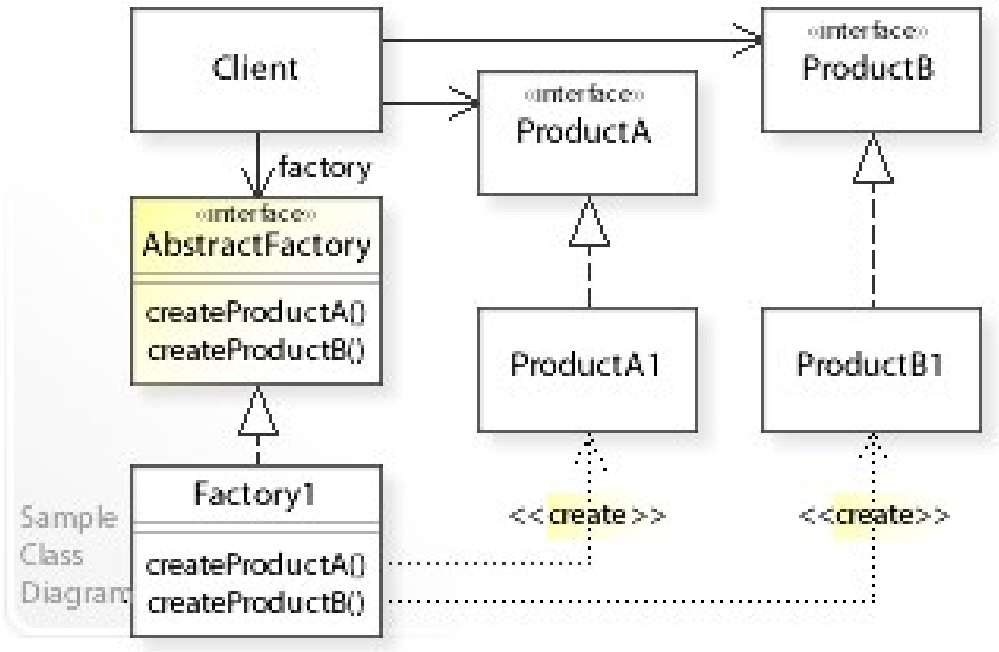
- Un diagramme de classe pour décrire la structure statique
- Un diagramme de sequence (ou du code) pour décrire le comportement dynamique

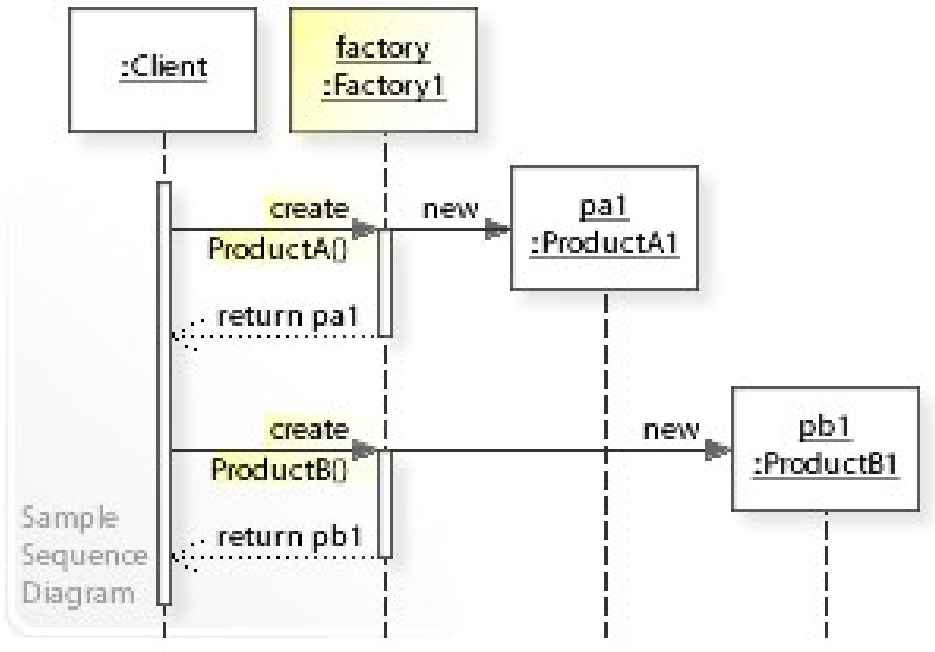
Source: https://en.wikipedia.org/wiki/Abstract_factory_pattern#/media/File:W3sDesign_Abstract_Factory_Design_Pattern_UML.jpg

VIP : Very Important Patterns

VIP : Very Important Patterns

Factory/Abstract Factory





Source Wikipedia : https://en.wikipedia.org/wiki/Abstract_factory_pattern#/media/File:W3sDesign_Abstract_Factory_Design_Pattern_UML.jpg

Factory Factory

```
public void useFactories() {
//Le client a une dépendance sur la l'implémentation ProduitA1
ProduitA produitA1 = new ProduitA1();
produitA1.methodeA();

//Le client a une dépendance sur la factory uniquement
ProduitA produitA2 = ProduitFactory.getProduitA();
produitA2.methodeA();

}
```

```
class ProduitFactory {
    static ProduitA createProduitA() {
        return new ProduitA1();
    }
}

abstract class ProduitA {
        public abstract void methodeA();
}

class ProduitA1 extends ProduitA {
    public void methodeA() {
        System.out.println("ProduitA1.methodeA()");
    }
}
```

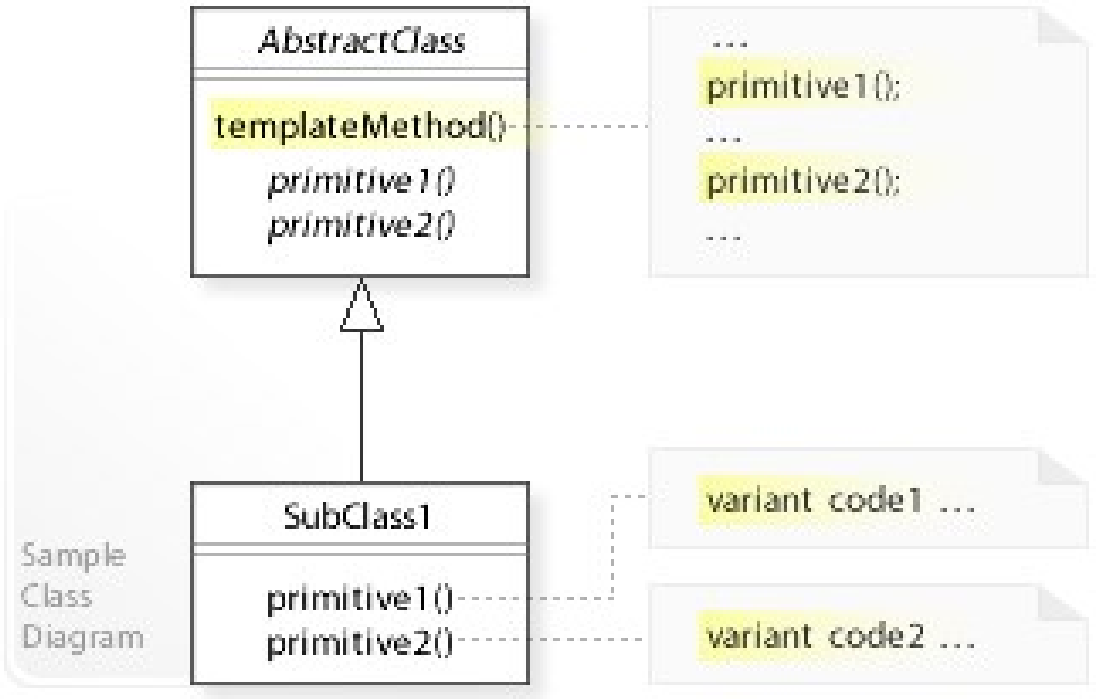
Source Wikipedia : https://en.wikipedia.org/wiki/Abstract_factory_pattern#/media/File:W3sDesign_Abstract_Factory_Design_Pattern_UML.jpg

Singleton

```
public class Singleton {

    private static volatile Singleton instance;

    private Singleton() {
    }

    public static Singleton getInstance(){
        if (instance==null)
        {
            synchronized (Singleton.class){
                if(instance==null)
                        instance=new Singleton();
                }
            }
        return instance;
    }

}
```

Source Wikipedia : https://en.wikipedia.org/wiki/File:Singleton_UML_class_diagram.svg

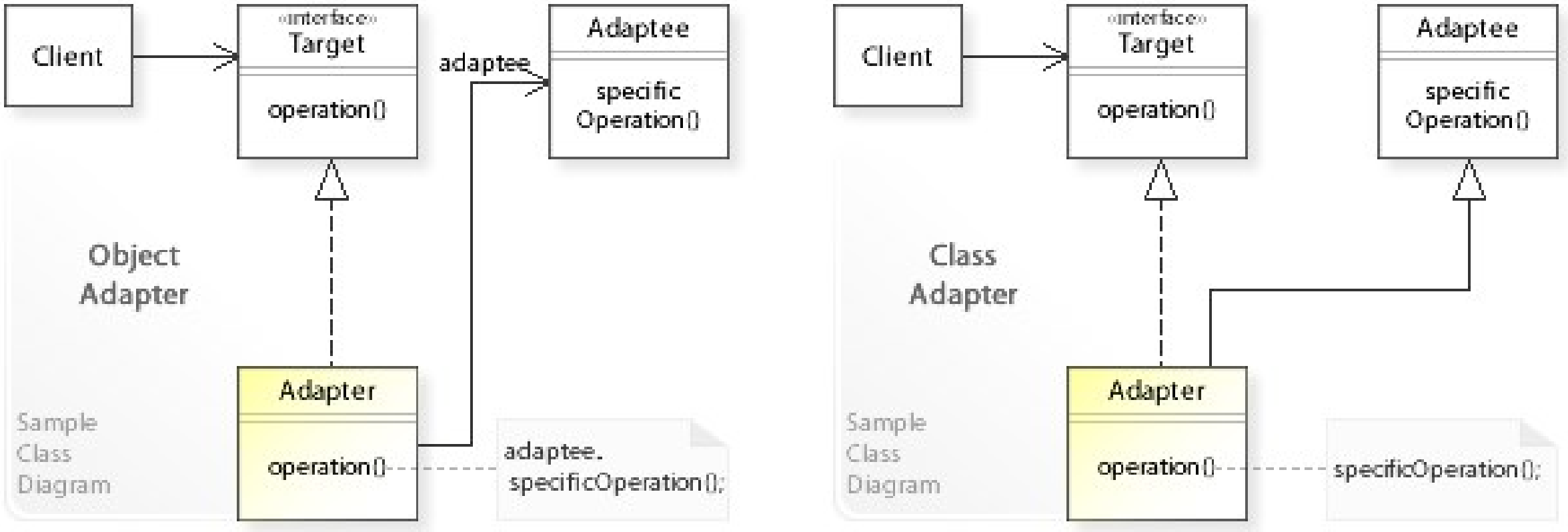
Template Method




```java
public abstract class RecetteCharlotte {
	public void createCharlotte() {
		laverLesFuits();
		prepareSirop();
		rajouterAlcoolDeFruit();
		miseEnPlaceBiscuits();
		miseEnPlaceFruits();
	}

	protected abstract void laverLesFuits();
	protected abstract void rajouterAlcoolDeFruit();
	protected abstract void miseEnPlaceFruits();

	private void prepareSirop() {}
	private void miseEnPlaceBiscuits() {}
}

public class RecetteCharlotteAuxFraises extends RecetteCharlotte {
	@Override
	protected void laverLesFuits() { }
	@Override
	protected void rajouterAlcool() { }
	@Override
	protected void miseEnPlaceFruits() { }
}
public class RecetteCharlotteAuxFramboise extends RecetteCharlotte {
	@Override
	protected void miseEnPlaceFruits() { }
	@Override
	protected void laverLesFuits() { }
	@Override
	protected void rajouterAlcool() { }
}
```

Source Wikipedia : https://en.wikipedia.org/wiki/File:W3sDesign_Template_Method_Design_Pattern_UML.jpg

Adapter/Wrapper



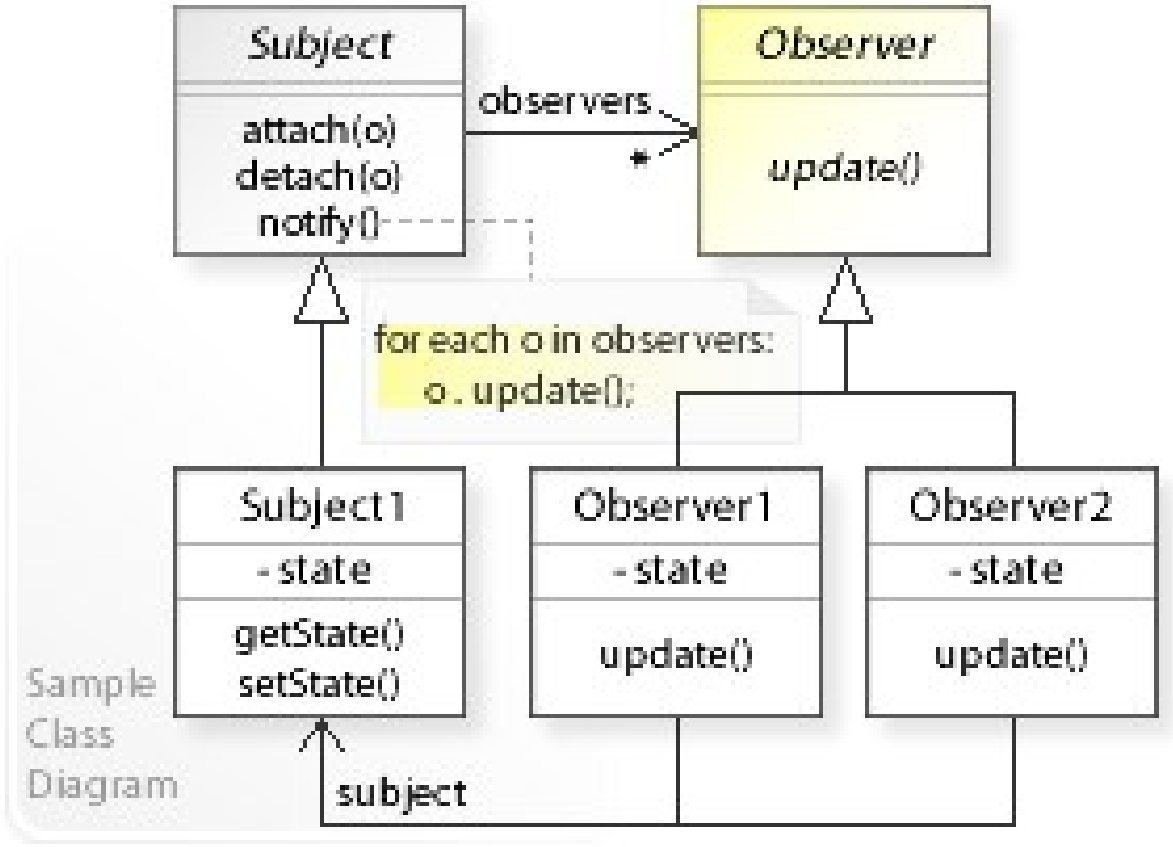
Source Wikipedia : https://en.wikipedia.org/wiki/File:W3sDesign_Adapter_Design_Pattern_UML.jpg

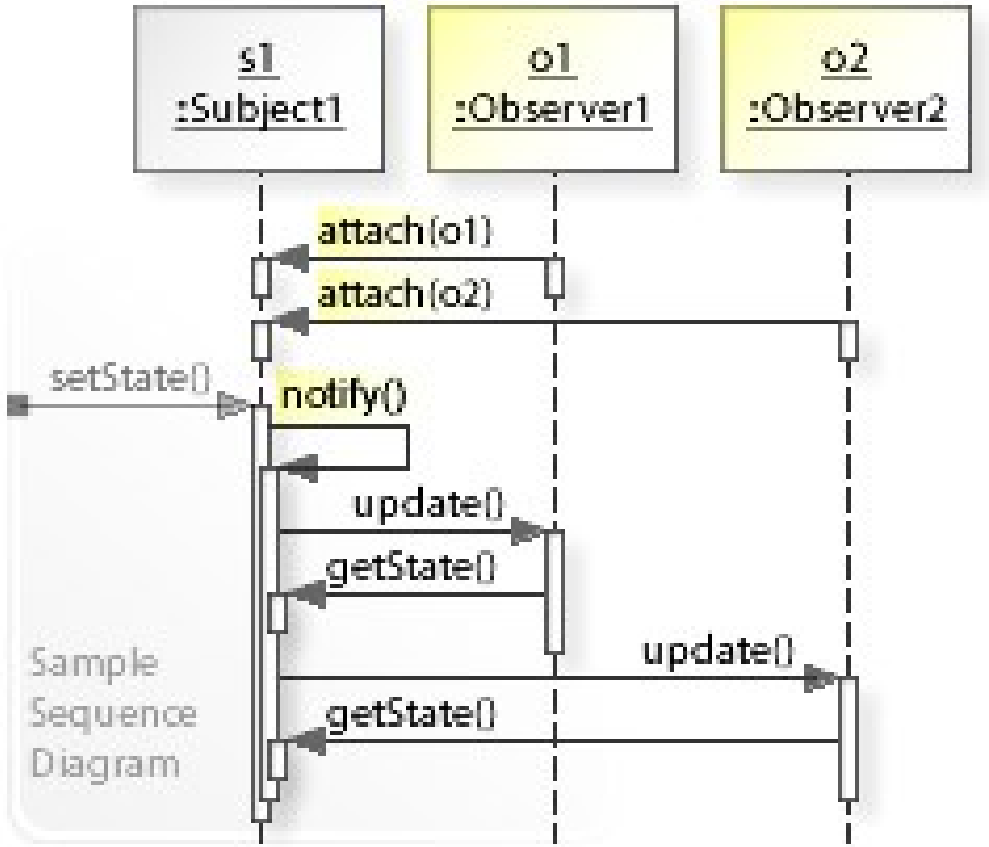
Adapter/Wrapper

```
interface Shape {
        float getSurface();
}

class Rectangle implements Shape {
…
    public float getSurface() {
        return length * height;
    }
}

class Disc implements Shape {
…
    public float getSurface() {
        return 3.14f * radius * radius;
    }
}
```

```
class Triangle {
…
    public double getArea() {
        return base * height / 2;
    }
}

class TriangleShapeAdapter implements Shape {
    Triangle triangle;
    public float getSurface() {
        return Float.valueOf((float) triangle.getArea());
    }
}

float computeTotalSurface(List<Shape> shapes) {
        return shapes.stream().map(
                s -> s.getSurface()).reduce(0f, Float::sum);
}
```

Observer





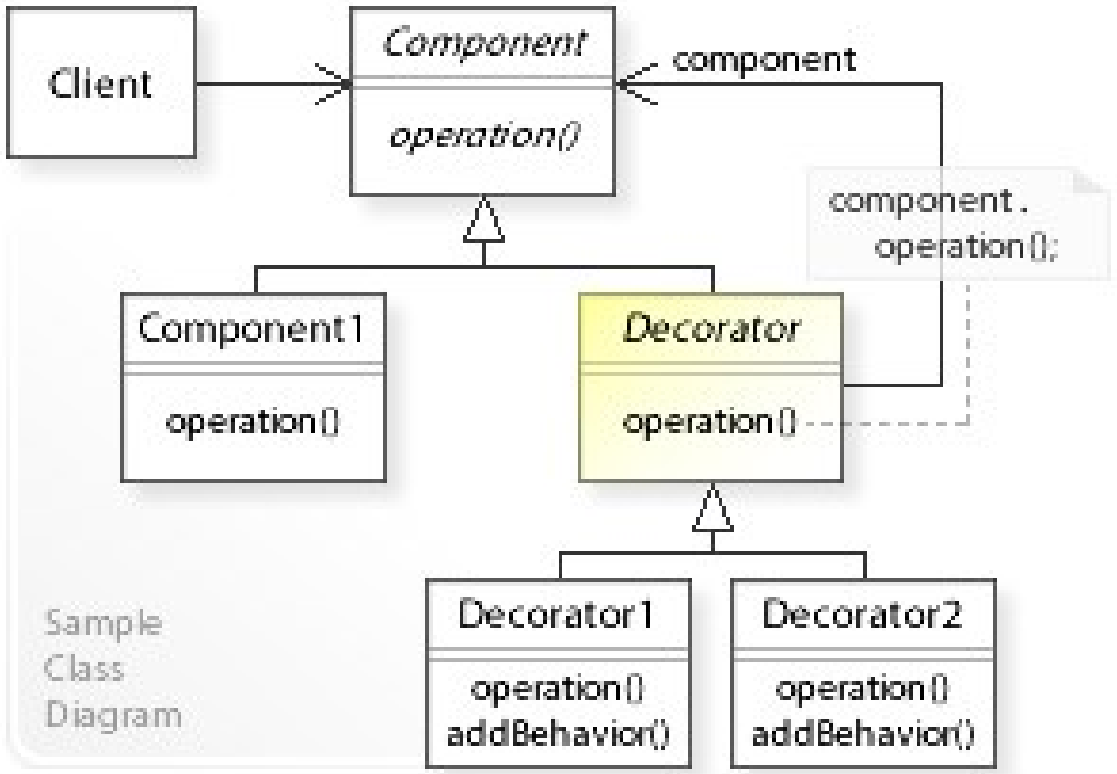
Source Wikipedia : https://en.wikipedia.org/wiki/File:W3sDesign_Observer_Design_Pattern_UML.jpg

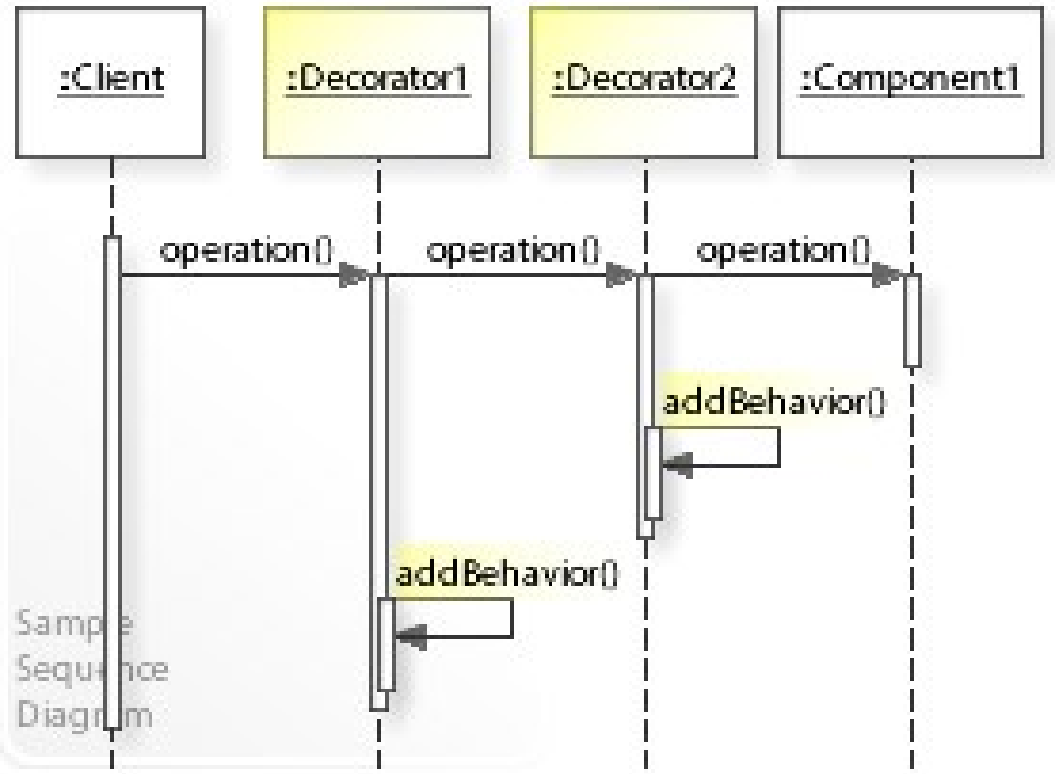
Observer

```
public void example() {
    System.out.println("Enter Text: ");
    EventSource eventSource = new EventSource();

    eventSource.addObserver(event -> {
    System.out.println("Received response: " + event);
    });

    eventSource.doSomethingUseful();
}

public interface Observer {
        void update(String event);
}
```

```
class EventSource {

private final List<Observer> observers =
                new ArrayList<>();

    private void notifyObservers(String event) {
        observers.forEach(o -> o.update(event));
    }

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void doSomethingUseful() {
        notifyObservers("new useful update");
    }
}
```

Decorator





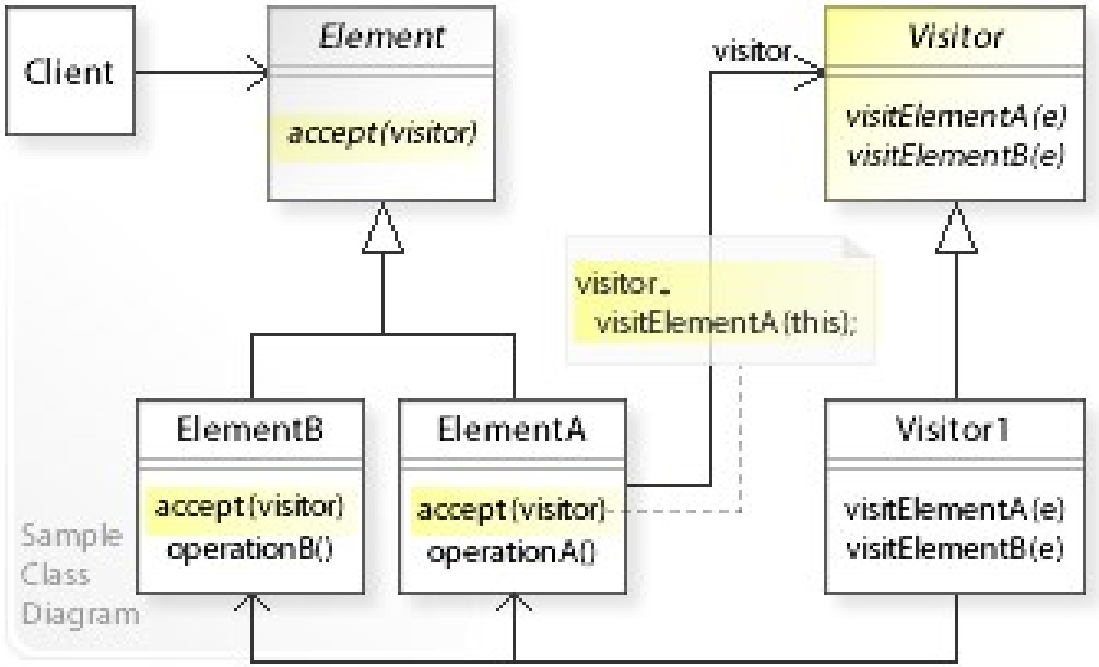
Source Wikipedia : https://en.wikipedia.org/wiki/File:W3sDesign_Decorator_Design_Pattern_UML.jpg

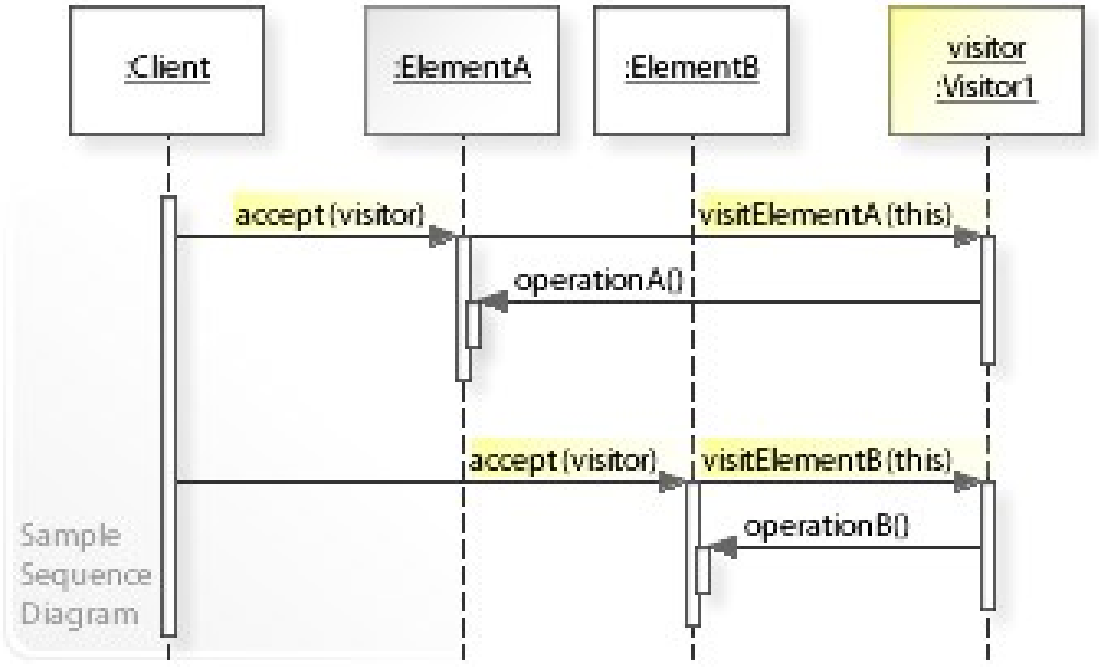
Decorator

```
public interface House {
    void sleep();
}

public class MyHouse implements House {
    public void sleep() {
        System.out.println("sleep quietly");
    }
}

public class SecuredHouseDecorator implements House {
    private House house;

    public SecuredHouseDecorator(House house) {
        this.house = house;
    }

    public void sleep() {
        enableAlarm();
        house.sleep();
    }
    private void enableAlarm() {}
}
```

```
public class CleanedHouseDecorator implements House {
    private House house;

    public CleanedHouseDecorator(House house) {
        this.house = house;
    }
    public void sleep() {
        clean();
        house.sleep();
    }
    private void clean() {}
}

public void goToSleepInACleanAndSecuredHouse() {
    House mySecuredAndCleanedHouse =
        new CleanedHouseDecorator(
                new SecuredHouseDecorator(
                        new MyHouse()));
}
```

Visitor





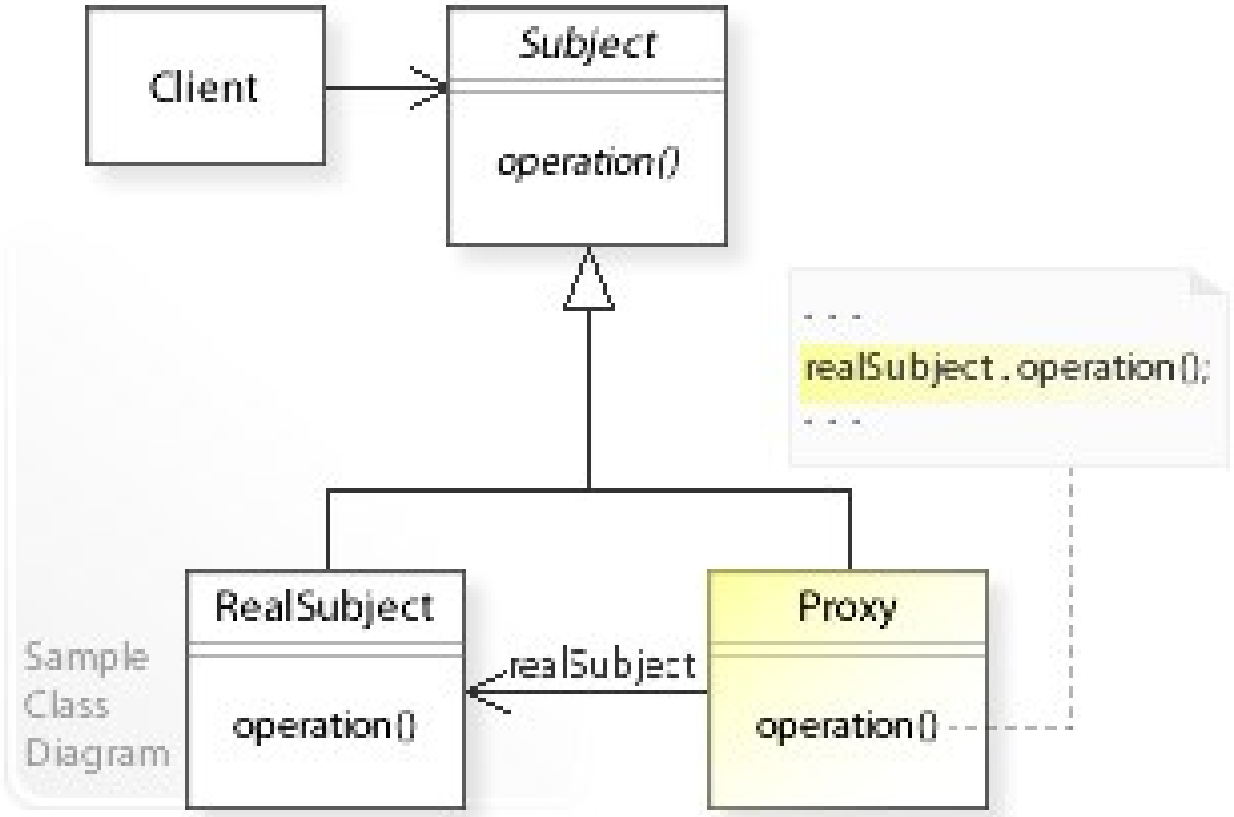
Source Wikipedia : https://en.wikipedia.org/wiki/File:W3sDesign_Visitor_Design_Pattern_UML.jpg

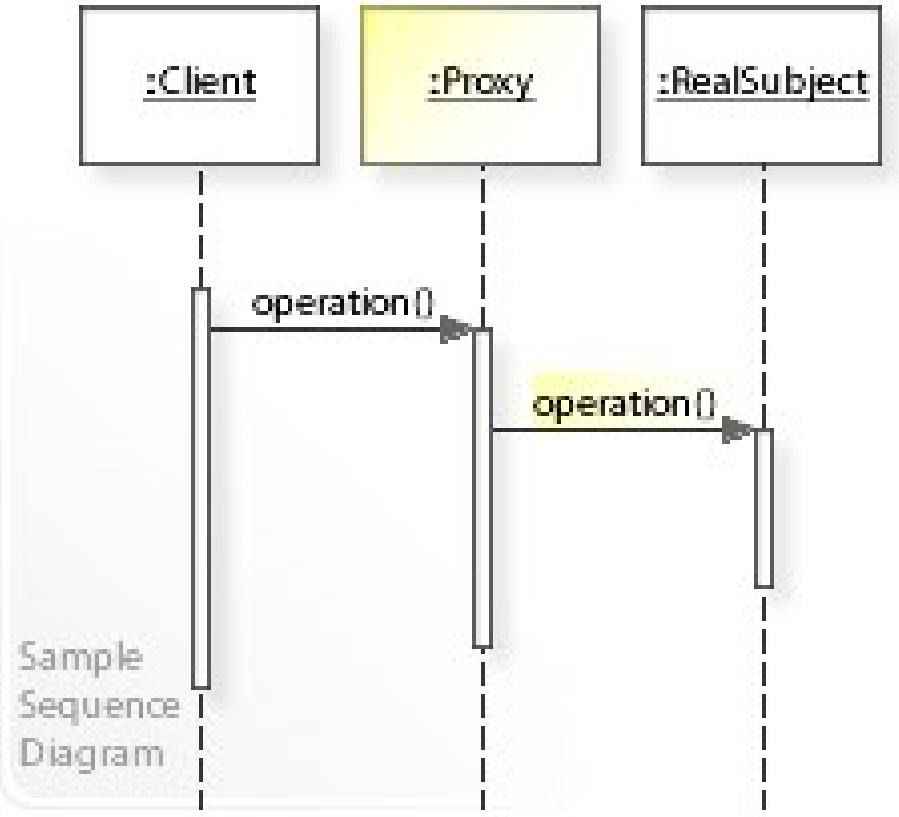
Visitor

```
interface Visitable {
     public float accept(ShoppingCartVisitor visitor);
}

class Book implements Visitable {
     public float price;

     public Book(int cost) {
     this.price = cost;
     }
     public float accept(ShoppingCartVisitor visitor) {
     return visitor.visit(this);
     }
}

class Fruit implements Visitable {
      …
     public int getPricePerKg() { return pricePerKg; }
     public int getWeight() { return weight; }

     public float accept(ShoppingCartVisitor visitor) {
     return visitor.visit(this);
     }
}
```

```
interface ShoppingCartVisitor {
float visit(Book book);
float visit(Fruit fruit);
}

class ShoppingCartVisitorImpl implements ShoppingCartVisitor
{
public float visit(Book book) {
float cost = 0;
if (book.price > 50) {
cost = book.price * .9f;
} else
cost = book.price;
return cost;
}
public float visit(Fruit fruit) {
float cost = fruit.getPricePerKg() * fruit.getWeight();
return cost;
}
}

void testVisitor() {
Visitable[] items = new Visitable[] { new Book(20), new
Book(100), new Fruit(10, 2),
new Fruit(5, 5) };
ShoppingCartVisitor visitor = new ShoppingCartVisitorImpl();
float sum = Arrays.stream(items).map(v ->
v.accept(visitor)).reduce(0f, Float::sum);
}
```

Proxy





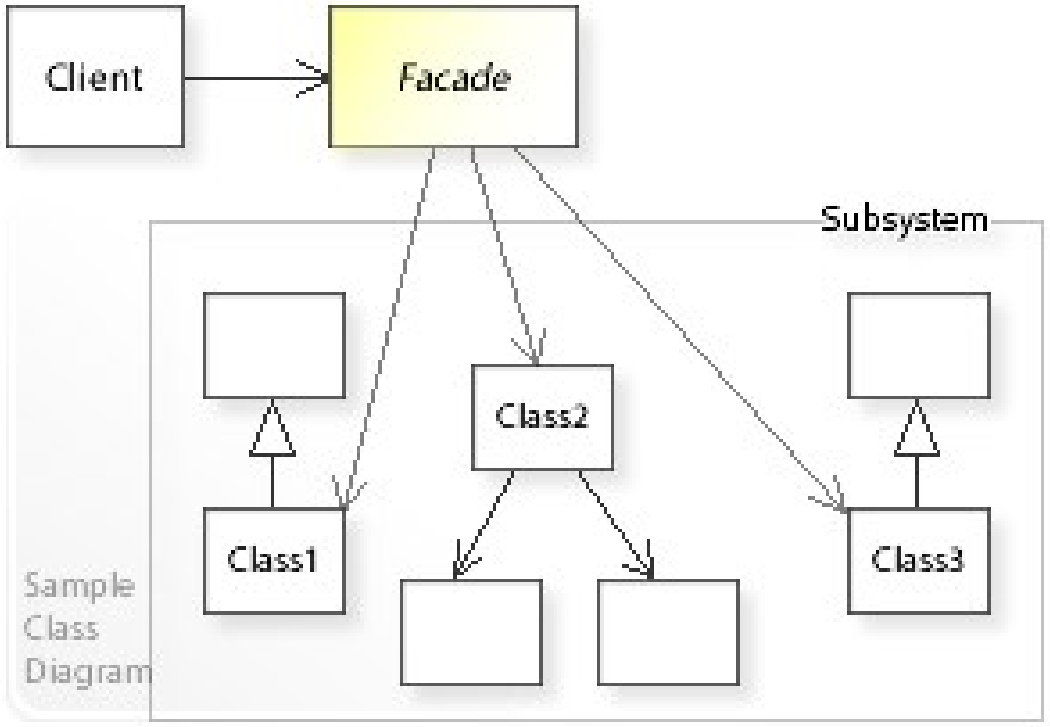
Source Wikipedia https://en.wikipedia.org/wiki/File:W3sDesign_Proxy_Design_Pattern_UML.jpg

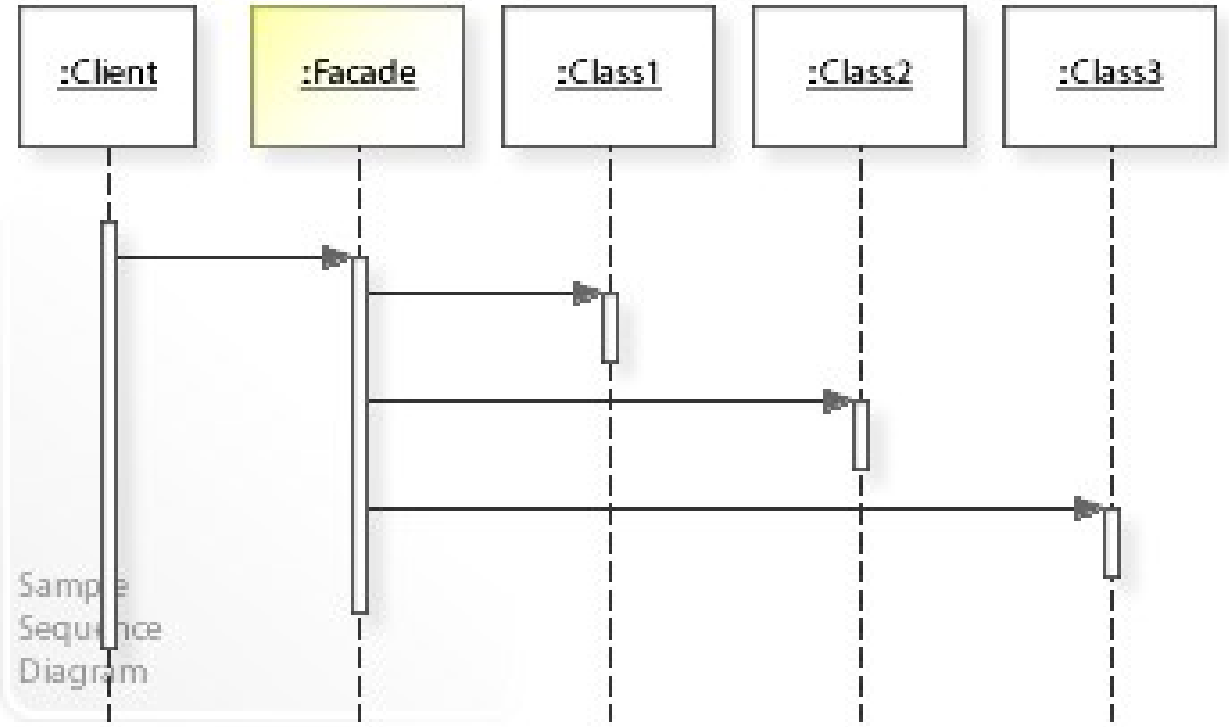
Proxy

```
interface Image {
    Image load(String url);
}

class RealImage implements Image {
    public Image load(String url) {
        // Lot's of heavy stuff
        return null;
    }
}
```

```
class ImageCacheProxy implements Image {
    Map<String, Image> cache = new HashMap<>();

    @Override
    public Image load(String url) {
        Image i = cache.get(url);
        if (cache.get(url) == null) {
            i = new RealImage().load(url);
            cache.put(url, i);
        }
        return i;
    }
}
```

Facade





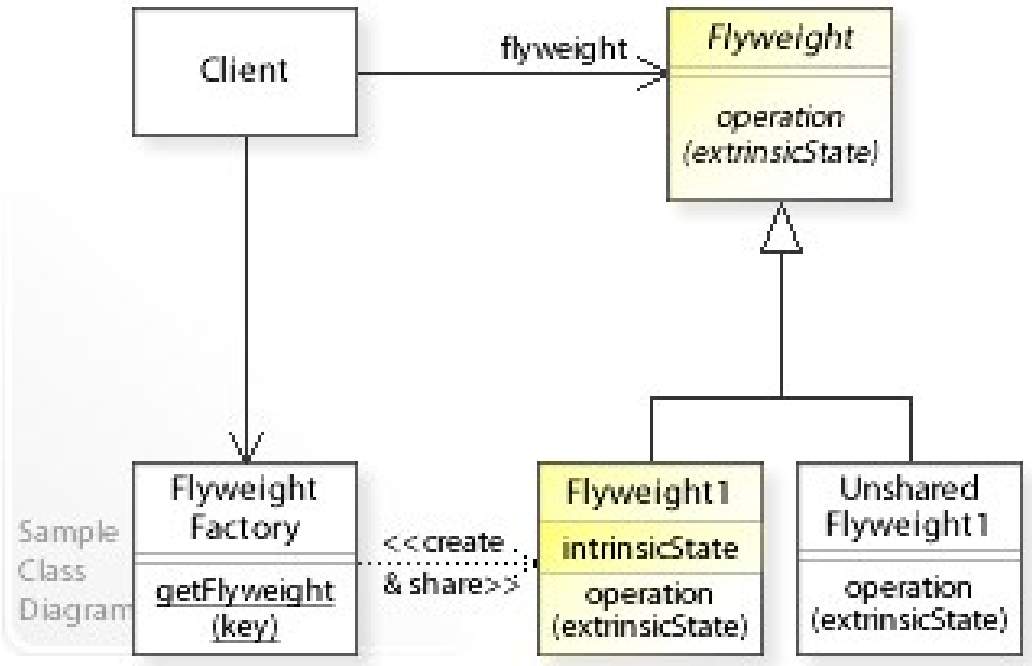
Source Wikipedia : https://en.wikipedia.org/wiki/File:W3sDesign_Facade_Design_Pattern_UML.jpg

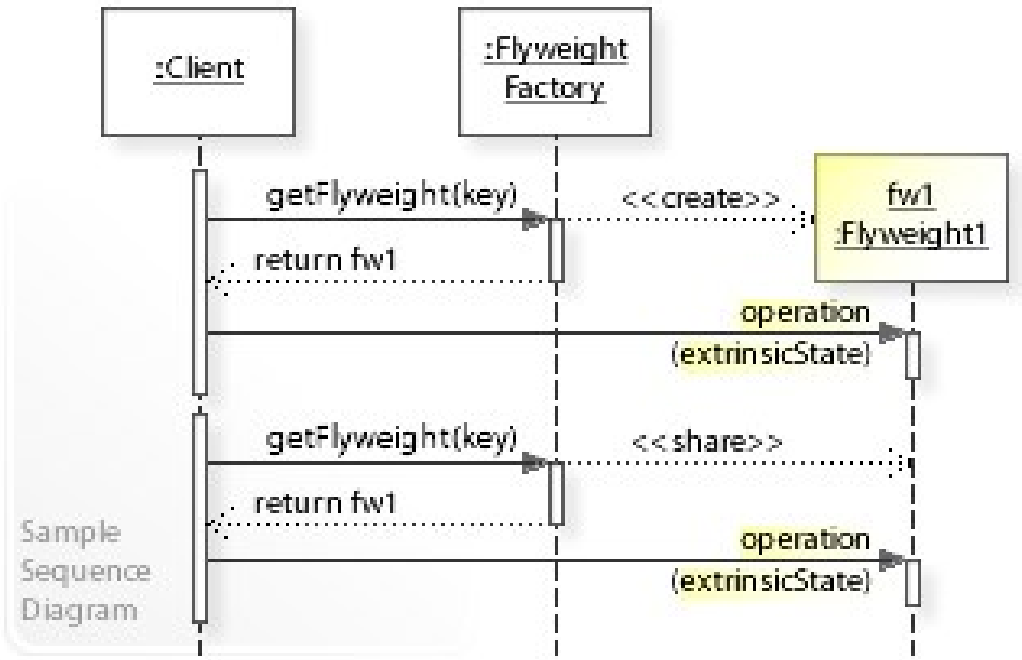
Facade

```
class CPU {
    public void freeze() { ... }
    public void jump(long position) { ... }
    public void execute() { ... }
}

class HardDrive {
    public byte[] read(long lba, int size) { ... }
}

class Memory {
    public void load(long position, byte[] data) { ... }
}


void useComputer() {
     ComputerFacade computer = new ComputerFacade();
    computer.start();
}
```

```
class ComputerFacade {
    private final CPU processor;
    private final Memory ram;
    private final HardDrive hd;

    public ComputerFacade() {
        this.processor = new CPU();
        this.ram = new Memory();
        this.hd = new HardDrive();
    }

    public void start() {
        processor.freeze();
        ram.load(BOOT_ADDRESS, hd.read(BOOT_SECTOR,
                    SECTOR_SIZE));
        processor.jump(BOOT_ADDRESS);
        processor.execute();
    }
```

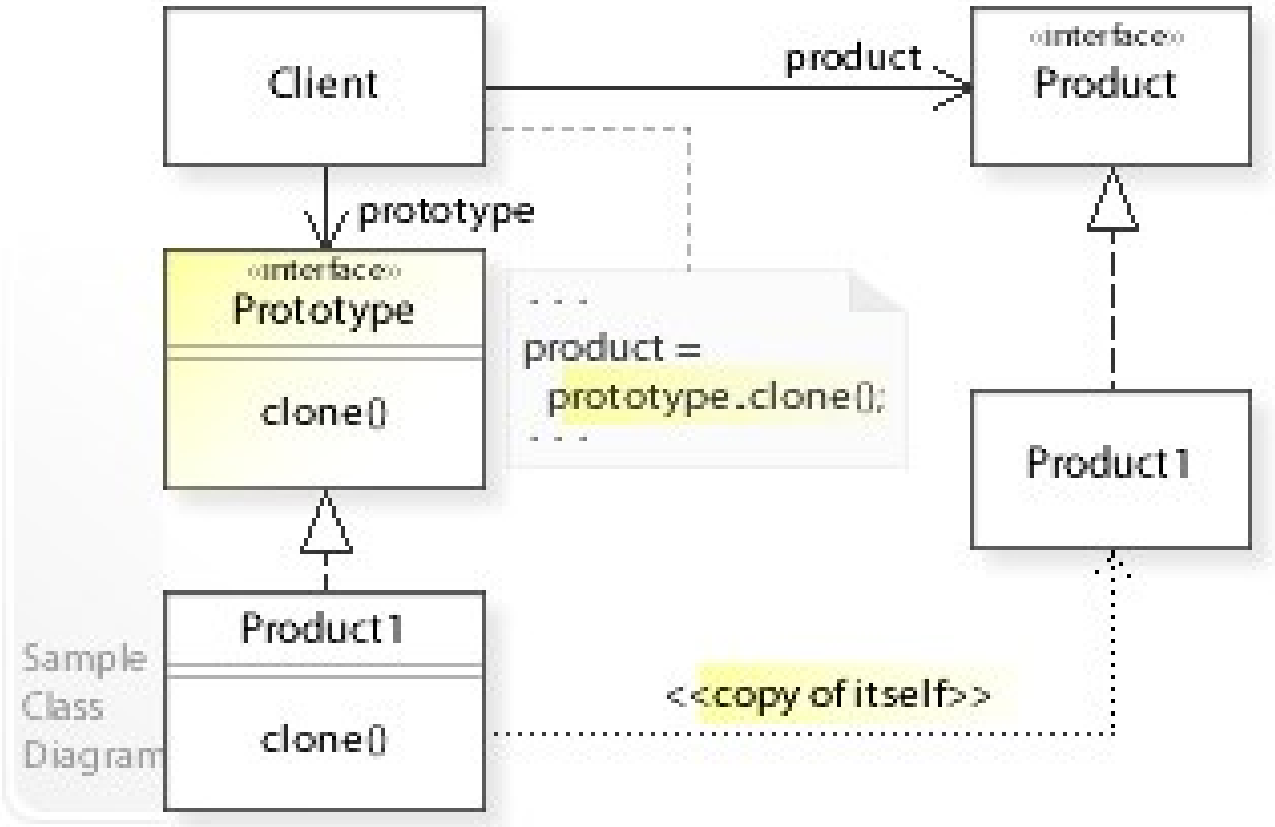
Flyweight

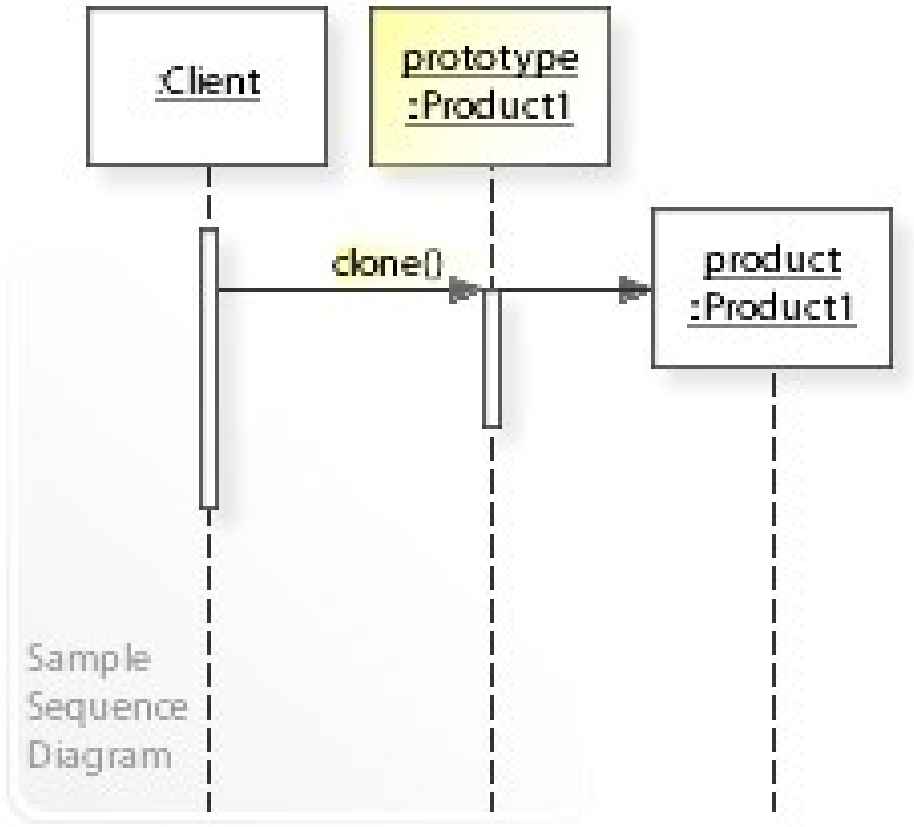




Source Wikipedia : https://en.wikipedia.org/wiki/File:W3sDesign_Flyweight_Design_Pattern_UML.jpg

Prototype





Source Wikipedia : https://en.wikipedia.org/wiki/File:W3sDesign_Prototype_Design_Pattern_UML.jpg