COIS3040 Lecture 5

Singleton

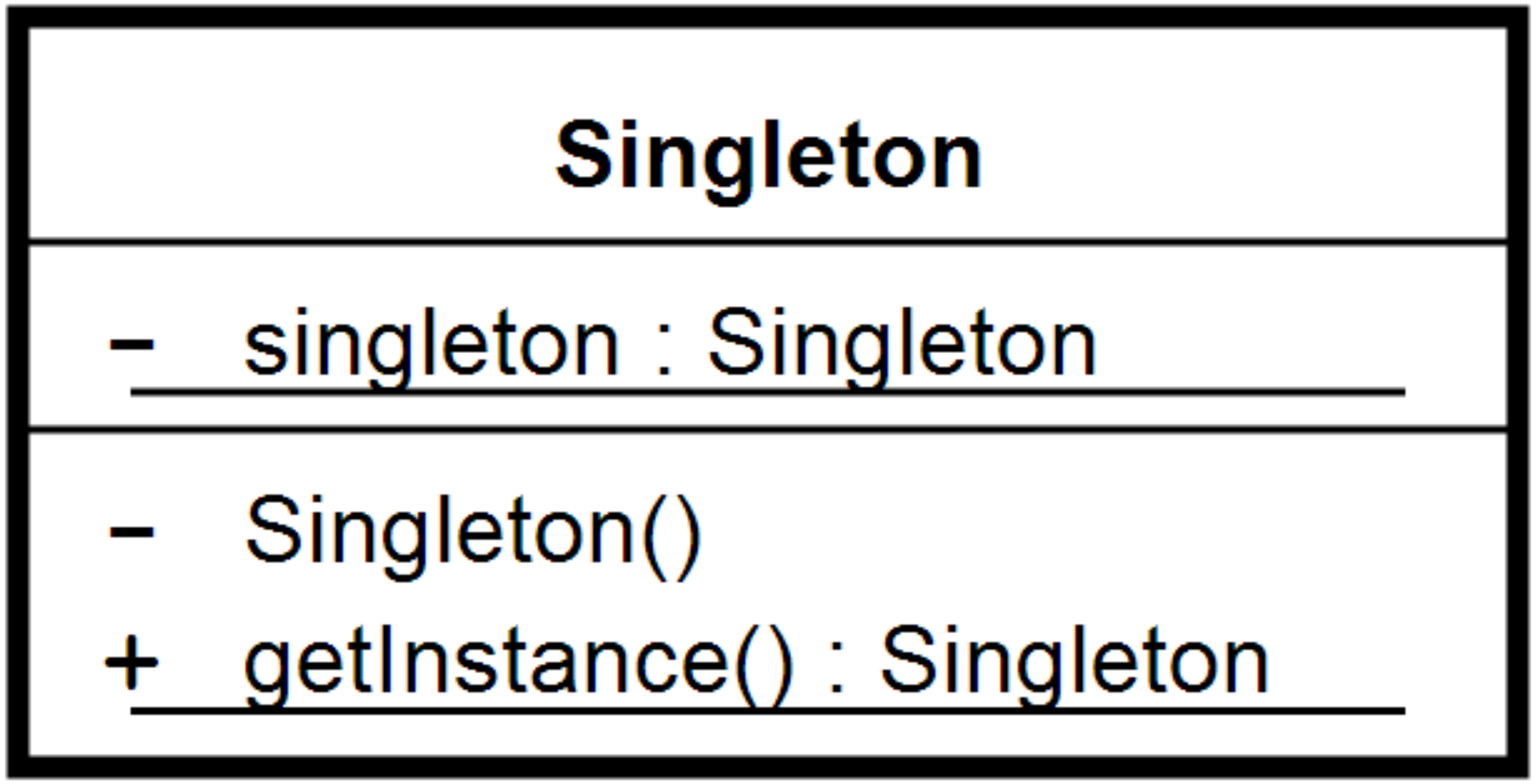
- It's important for some classes to have exactly one instance.
- More than one instance will result in incorrect program behaviour
- More than one instance will result in the overuse of resources
- More than one instance will result in inconsistent results
- There is a need for a global point of access

Singleton

- Example: There must be one instance of the printer spooler to accessed by all clients.
- This usually happens when you want to share a global resource.
- The singleton pattern ensures that there is only one point of entry and only one instance is created.

Singleton

- How to do that?



Singleton

```
public final class Singleton {
    private static final Singleton INSTANCE = new Singleton();

    private Singleton() {}

    public static Singleton getInstance() {
        return INSTANCE;
    }
}
```

Singleton —Eager initialization

static private data element

```
public final class Singleton {
    private static final Singleton INSTANCE = new Singleton();

    private Singleton() {}

    public static Singleton getInstance() {
        return INSTANCE;
    }
}
```

private constructor

public static getter

Singleton—Lazy instantiation

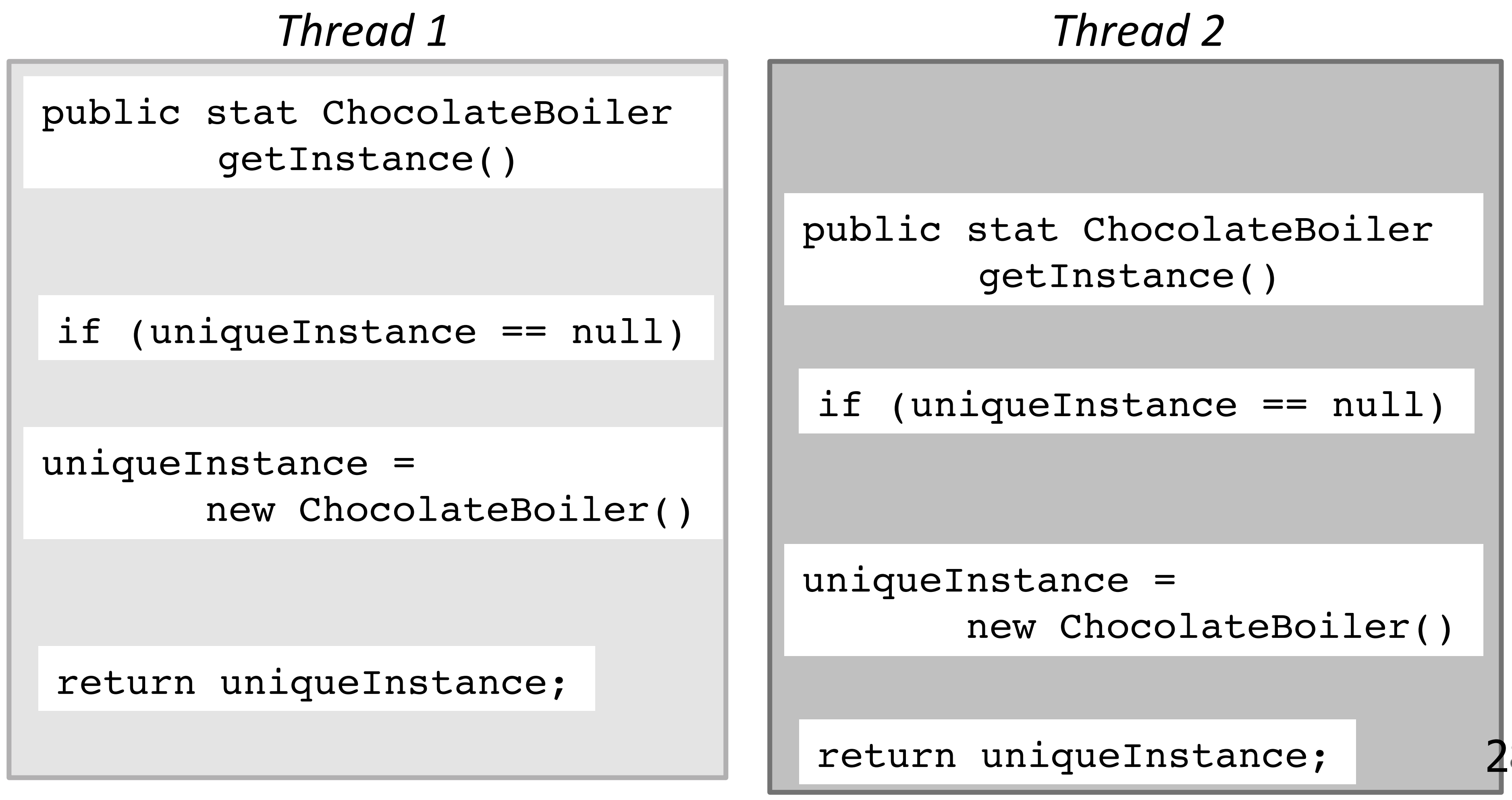
initialize with null

```
public final class Singleton {
    private static Singleton instance = null;

    private Singleton() {}

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

lazy instantiation

Singleton and Multithreading

When 2 threads are calling getInstance,
you will have two instances



Thread 1

```
public stat ChocolateBoiler
        getInstance()

if (uniqueInstance == null)

uniqueInstance =
       new ChocolateBoiler()

return uniqueInstance;
```

Thread 2

```
public stat ChocolateBoiler
        getInstance()

if (uniqueInstance == null)

uniqueInstance =
       new ChocolateBoiler()

return uniqueInstance;
```

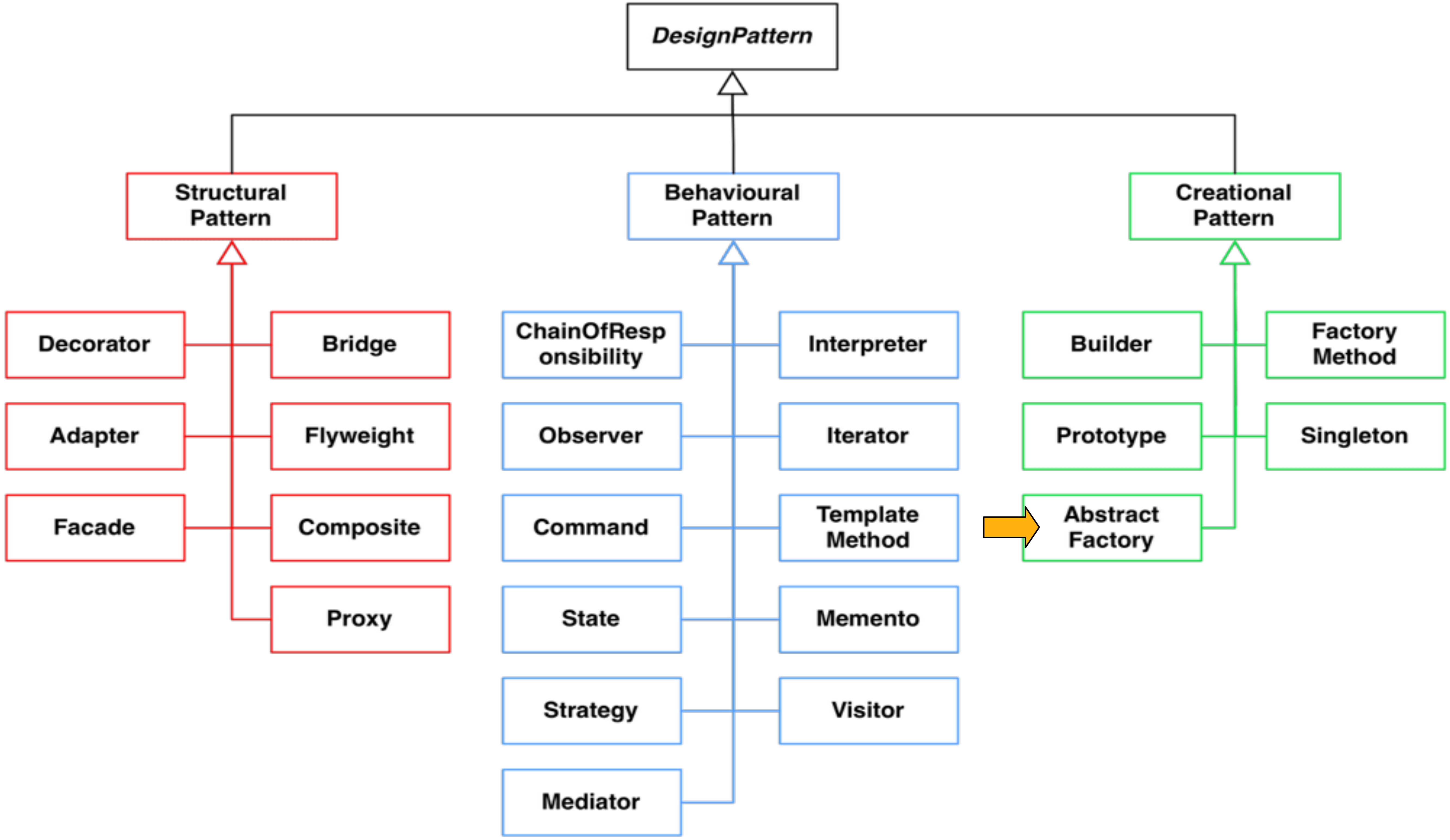
Singleton and Multithreading

Solution1: Use synchronized ..

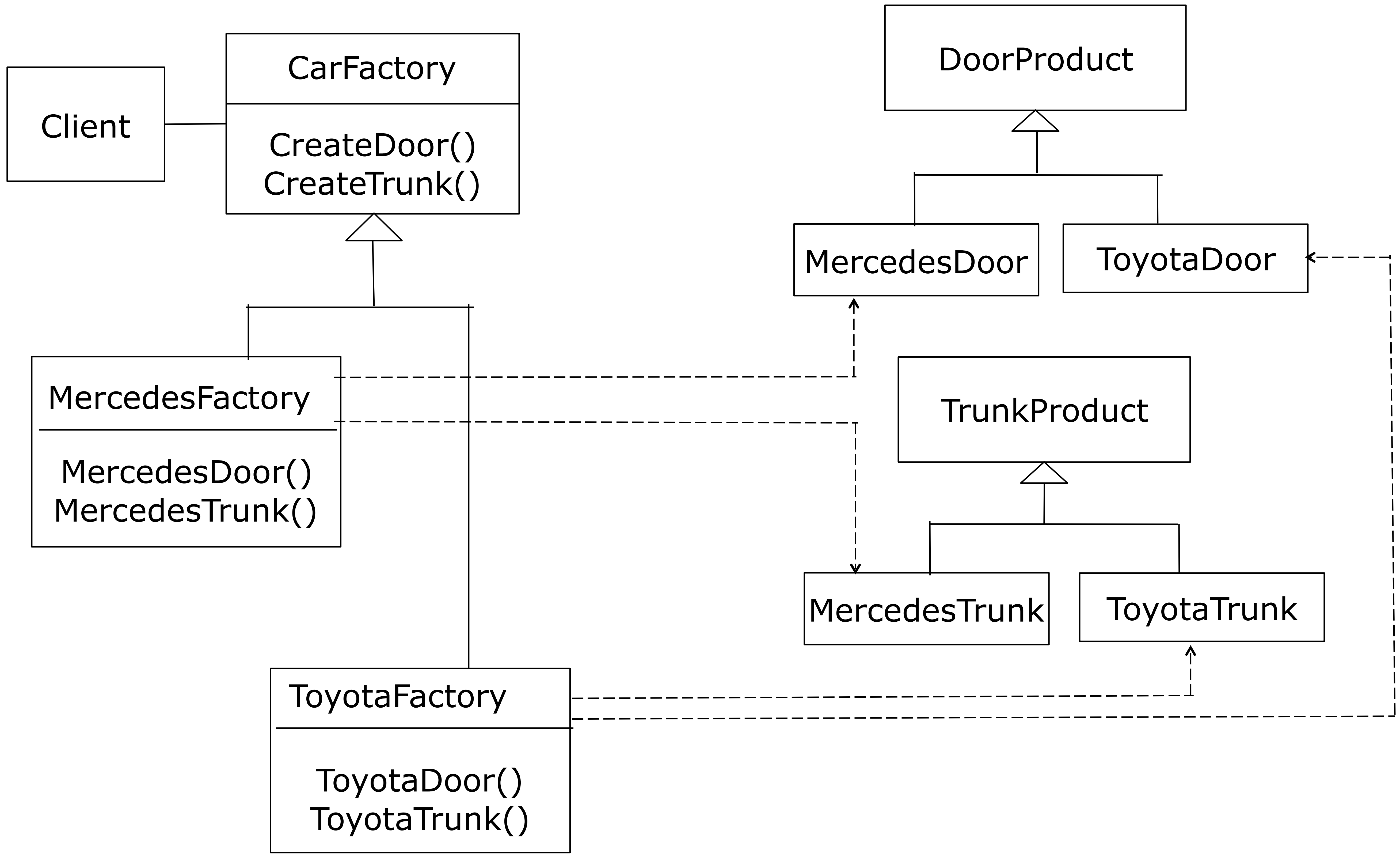
```
public static synchronized Singleton getInstance()
{... }
```

Solution2: Use eagerly created Singleton

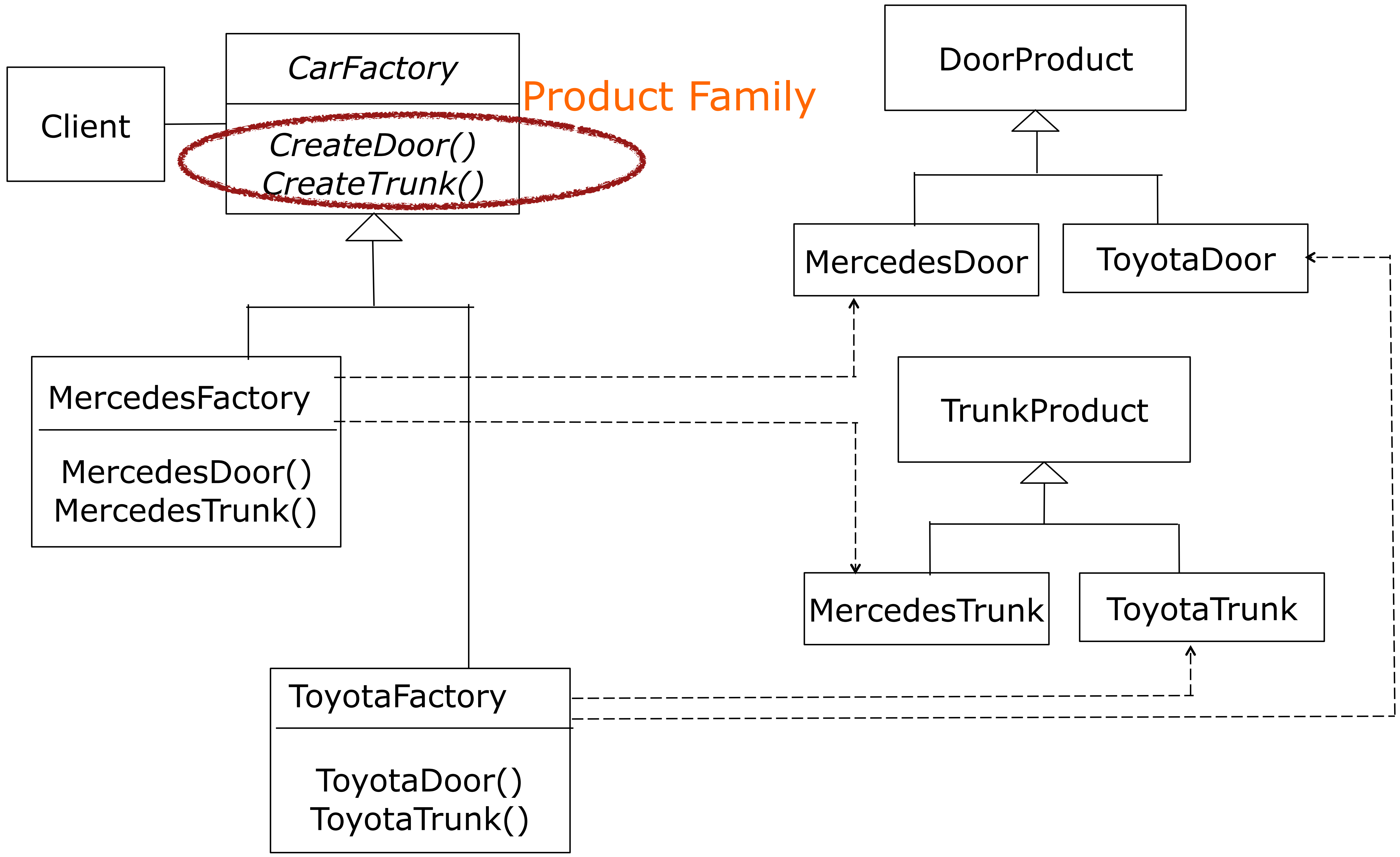
Taxonomy of Design Patterns



Abstract Factory



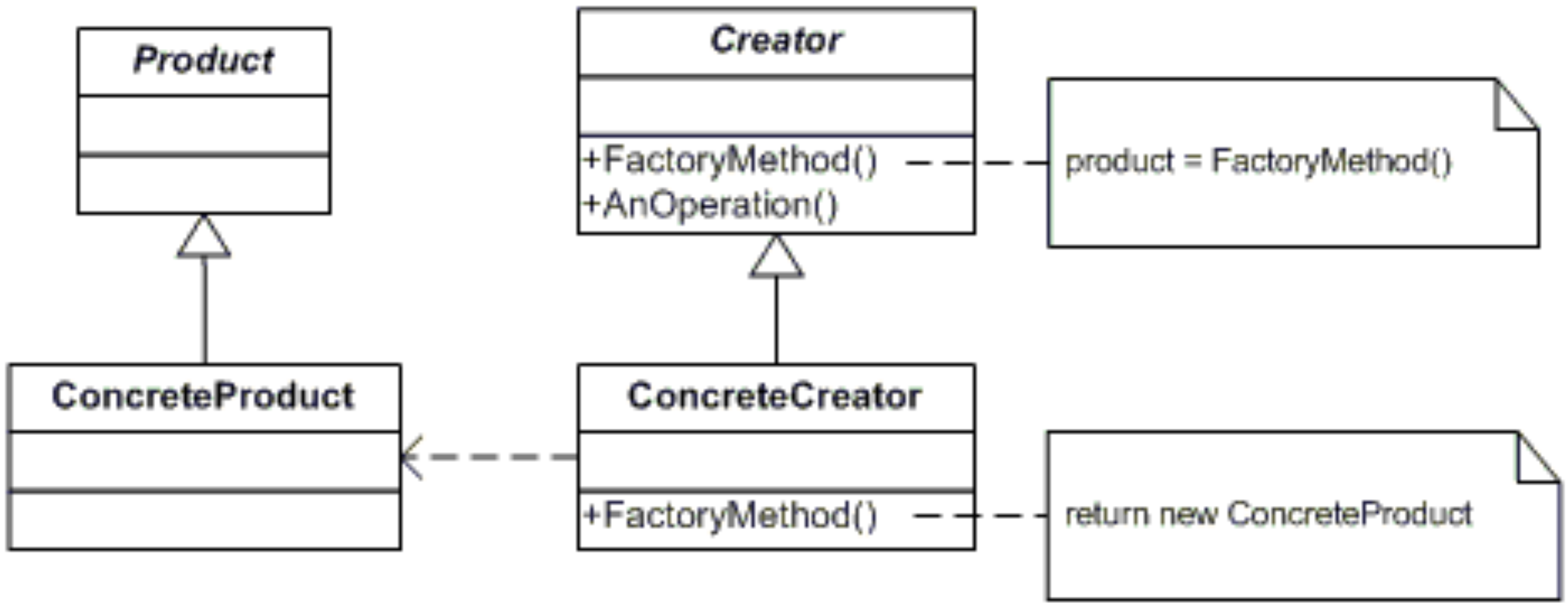
Abstract Factory



Abstract Factory

- The methods in the Abstract Factory are product type dependent, so if we add another product, we need to change the interface of the base class.
- In a way we restrict the product combination that a client can create.

Factory Method Pattern



Factory Method Pattern

- Contains one method to produce one type of product related to its type.
- A class Creator can't anticipate the class of objects it must create.
- The Creator class wants its subclasses to specify the objects it creates.