Software Design 1

Software Engineering
Prof. Maged Elaasar

Learning Objectives

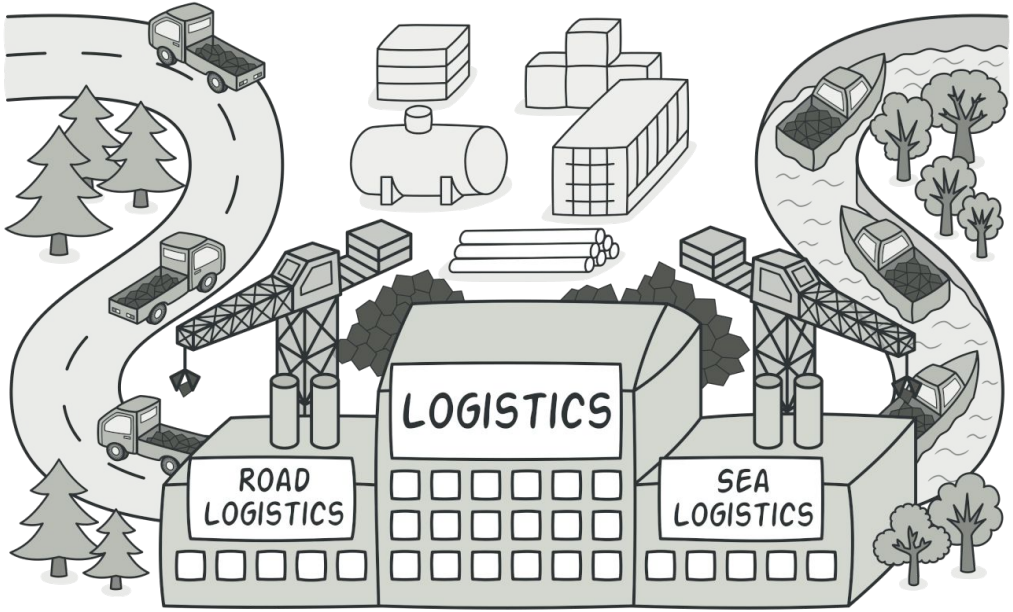
- GoF Creational Patterns
  - Factory Method pattern
  - Abstract Factory pattern
  - Singleton pattern

Factory Method Pattern

Factory Method Pattern

Problem

When an object can be created in different ways but used in a standardized way



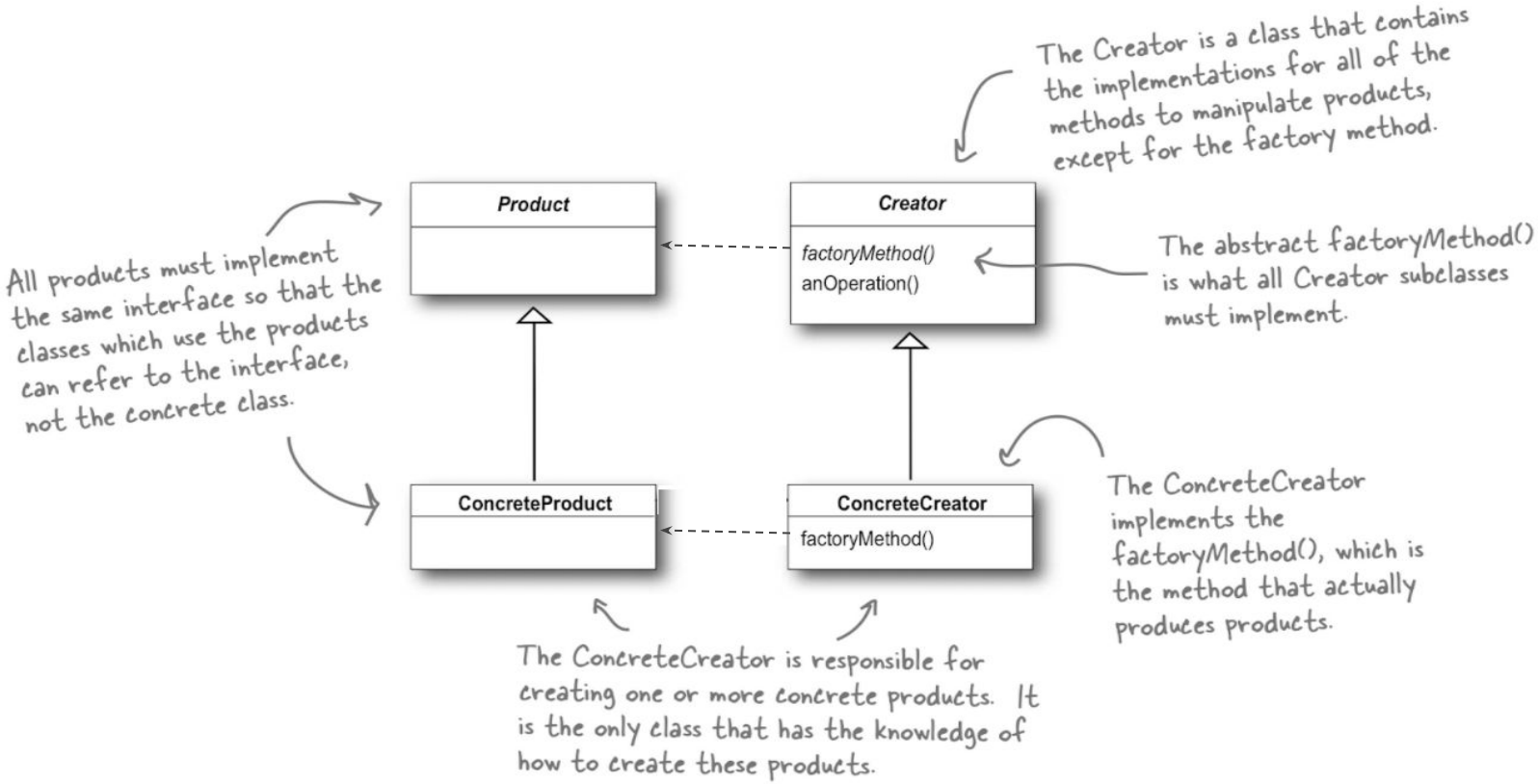
Example: Pizza Store

```
class PizzaStore {
   Pizza orderPizza(String type) {
      if (type.equals("cheese")) {
         pizza = new CheesePizza();
      } else if type.equals("greek")) {
      pizza = new GreekPizza();
      } else if type.equals("pepperoni")) {
      pizza = new PepperoniPizza();
      }
      pizza.prepare();
      pizza.bake();
      pizza.cut();
      pizza.box()
      return pizza;
   }
}
```

Part that varies.

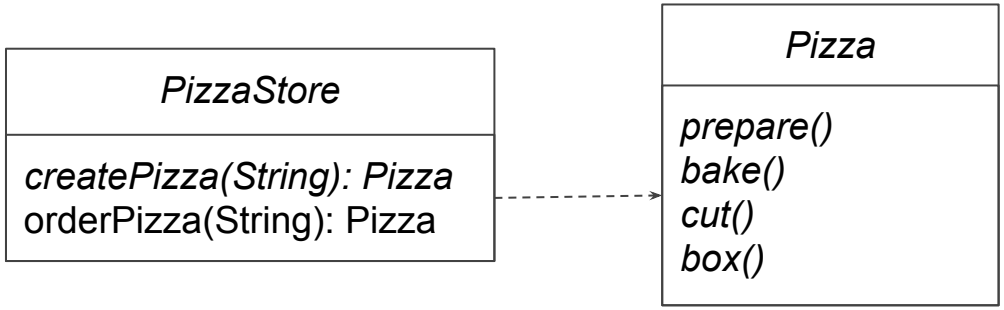
Part that remains constant

Factory Method Pattern



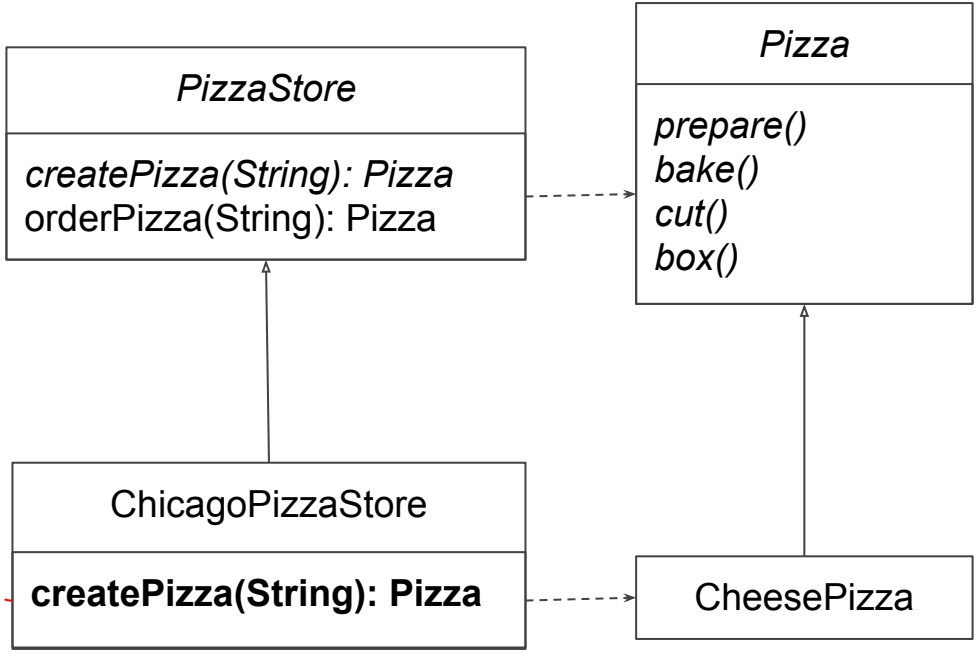
Example: Pizza Factory Method

```
public abstract class PizzaStore {

  protected abstract Pizza createPizza(String item);

  public Pizza orderPizza(String type) {
     Pizza pizza = createPizza(type);
     pizza.prepare();
     pizza.bake();
     pizza.cut();
     pizza.box();
     return pizza;
  }

}
```



Example: Pizza Factory Method

```
class ChicagoPizzaStore extends PizzaStore {
    protected Pizza createPizza(String type) {
        Pizza pizza = null;
        if (type.equals("cheese")) {
            pizza = new CheesePizza();
        } else if (type.equals("pepperoni")) {
            pizza = new PepperoniPizza();
        } else if (type.equals("clam")) {
            pizza = new ClamPizza();
        } else if (type.equals("veggie")) {
            pizza = new VeggiePizza();
        }
        return pizza;
    }
}
```



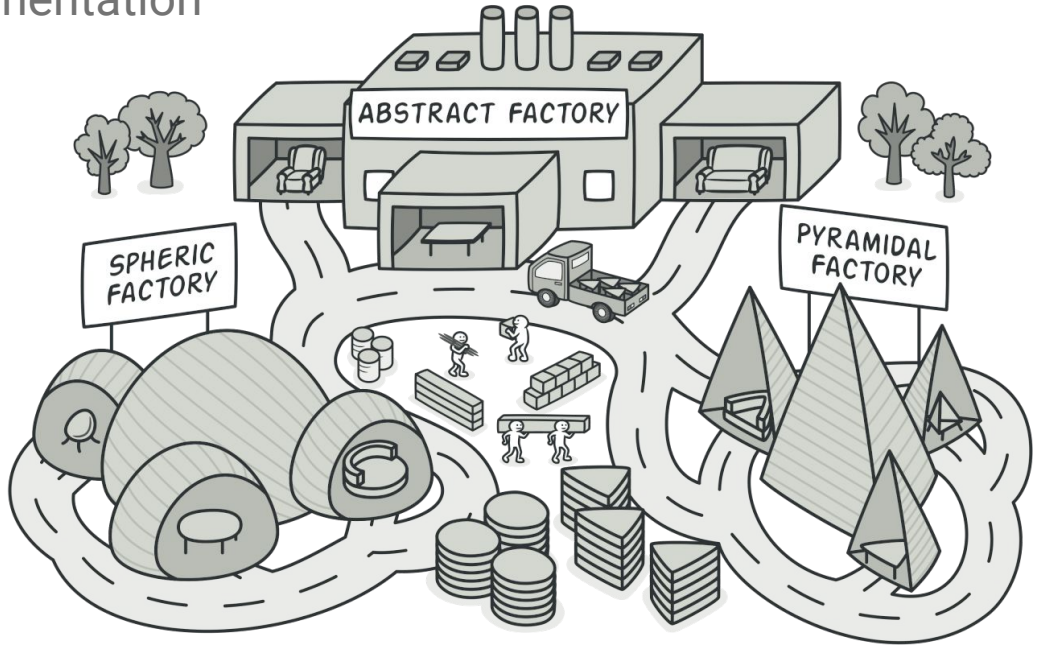

```
PizzaStore store = new ChicagoPizzaStore();
Pizza pizza = store.orderPizza("cheese");
```

Abstract Factory Pattern

Abstract Factory Pattern

Problem

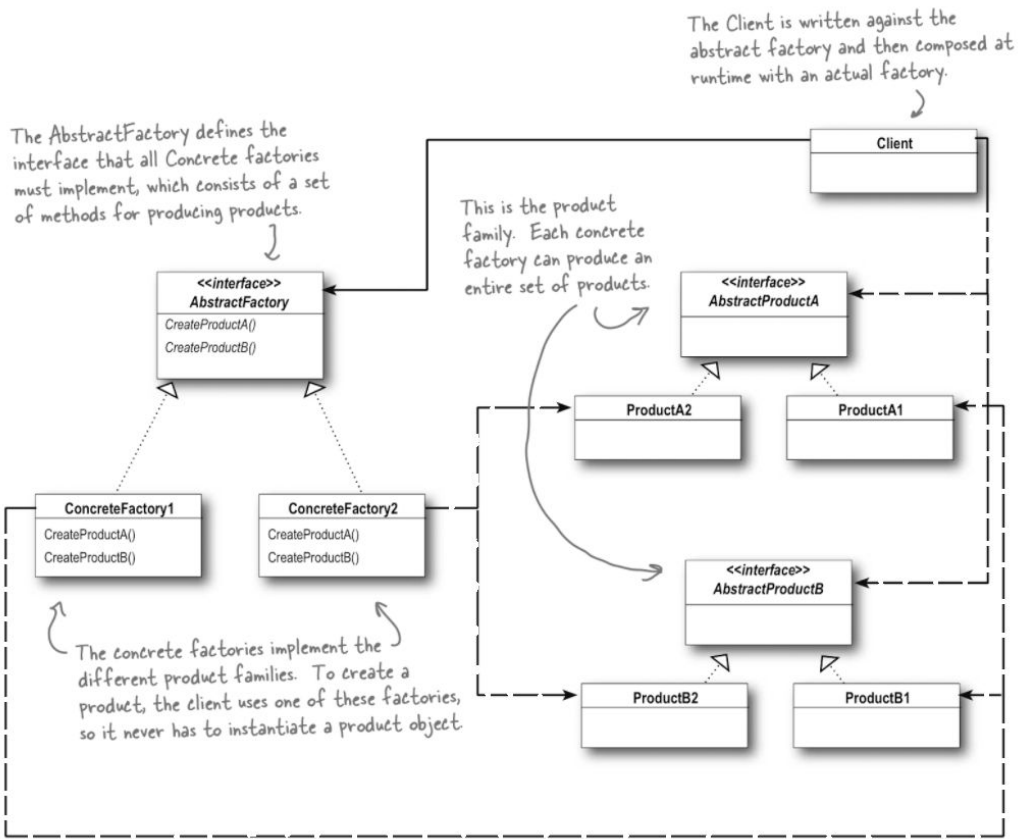
When a family of related objects need to be created in a consistent way without specifying their implementation



Example: Controlling Pizza Quality

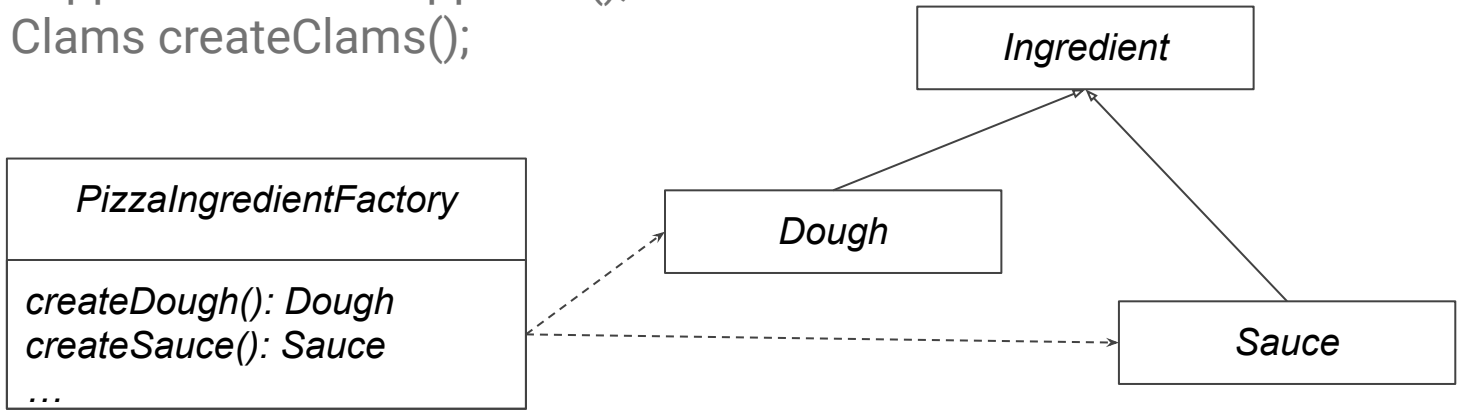
- Some of your franchises are substituting inferior ingredients to increase profit
- Time to enter the pizza ingredient business
  - You'll make all the ingredients yourself and ship them to your franchises
- You have the same product families (e.g., dough, sauce, cheese, veggies, meats, etc.) but different implementations (e.g., thin vs. thick or mozzarella vs. reggiano) based on each region

Abstract Factory Pattern



Example: Pizza Ingredient Factory

```
interface PizzaIngredientFactory {
    public Dough createDough();
    public Sauce createSauce();
    public Cheese createCheese();
    public Veggies[] createVeggies();
    public Pepperoni createPepperoni();
    public Clams createClams();
}
```

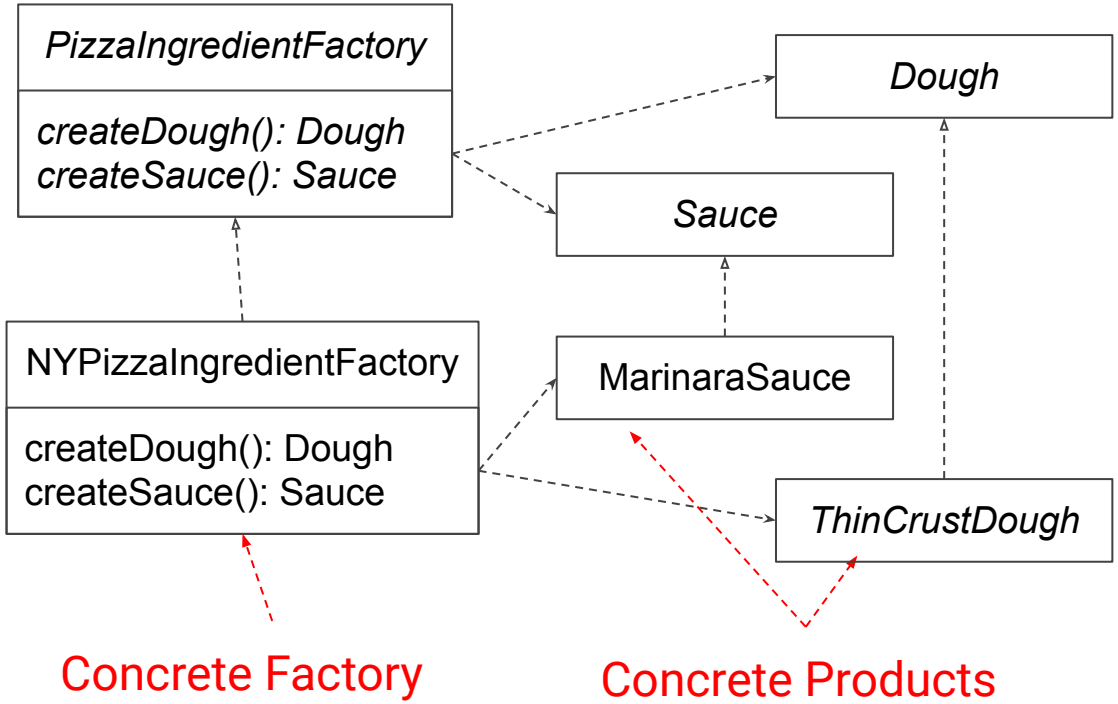


Abstract Factory

Abstract Products

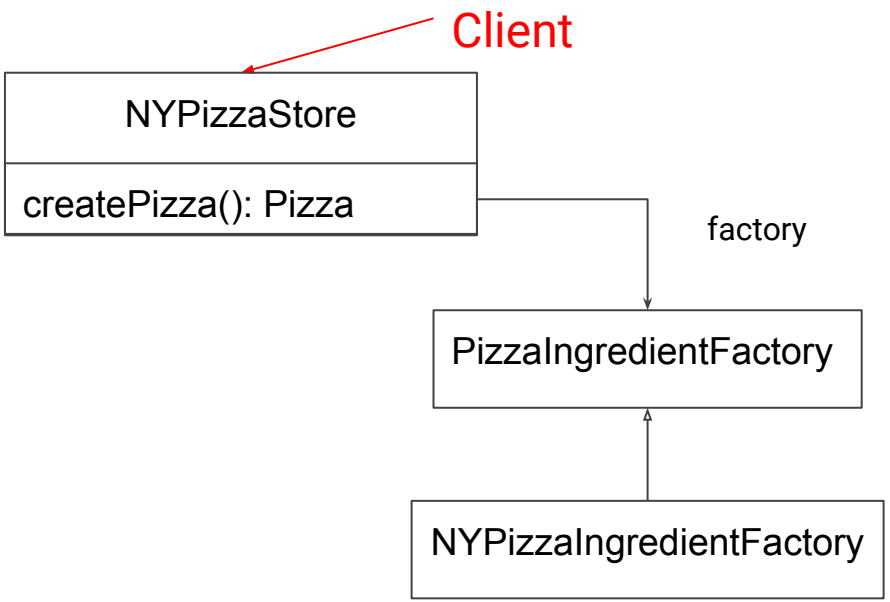
Example: Pizza Ingredient Factory

```
class NYPizzaIngredientFactory
   implements PizzaIngredientFactory {

   public Dough createDough() {
      return new ThinCrustDough();
   }
   public Sauce createSauce() {
      return new MarinaraSauce();
   }
   …
}

class ThinCrustDough implements Dough { }
class MarinaraSauce implements Sauce { }
```



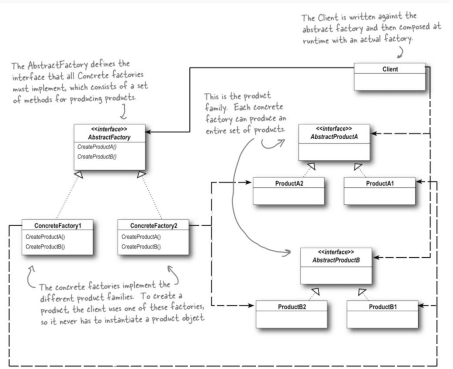
Example: Pizza Ingredient Factory

```
public class NYPizzaStore extends PizzaStore {
   private PizzaIngredientFactory factory =
      new NYPizzaIngredientFactory();

   protected Pizza createPizza(String item) {
      Pizza pizza = new Pizza();
      if (item.equals("cheese")) {
         Sauce sauce = factory.createSauce();
         pizza = pizza.addIngredient(sauce);
         Cheese cheese = factory.createCheese();
         pizza = pizza.addIngredient(cheese);
      } else if (item.equals("veggie")) {
         Sauce sauce = factory.createSauce();
         pizza = pizza.addIngredient(sauce);
         Cilantro cilantro = factory.createCilantro();
         pizza = pizza.addIngredient(cilantro);
       } // more of the same…
       return pizza;
    }
}
```

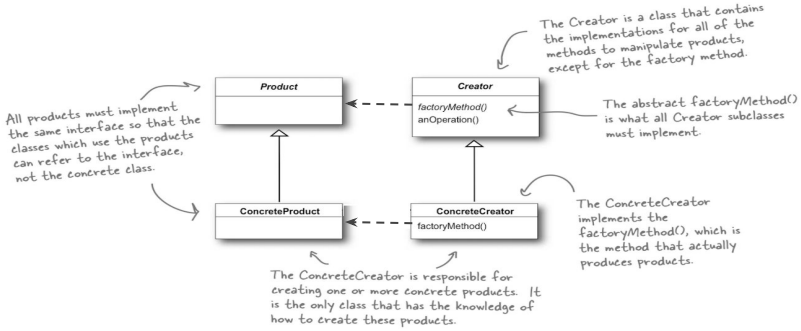


Abstract Factory vs. Factory Method

- Represented as a class
- Factory is dedicated to creating products
- The created products implement different interfaces but belong to the same family



- Represented as a method
- Exists in a class with other methods that calls it
- The created products implement the same interface

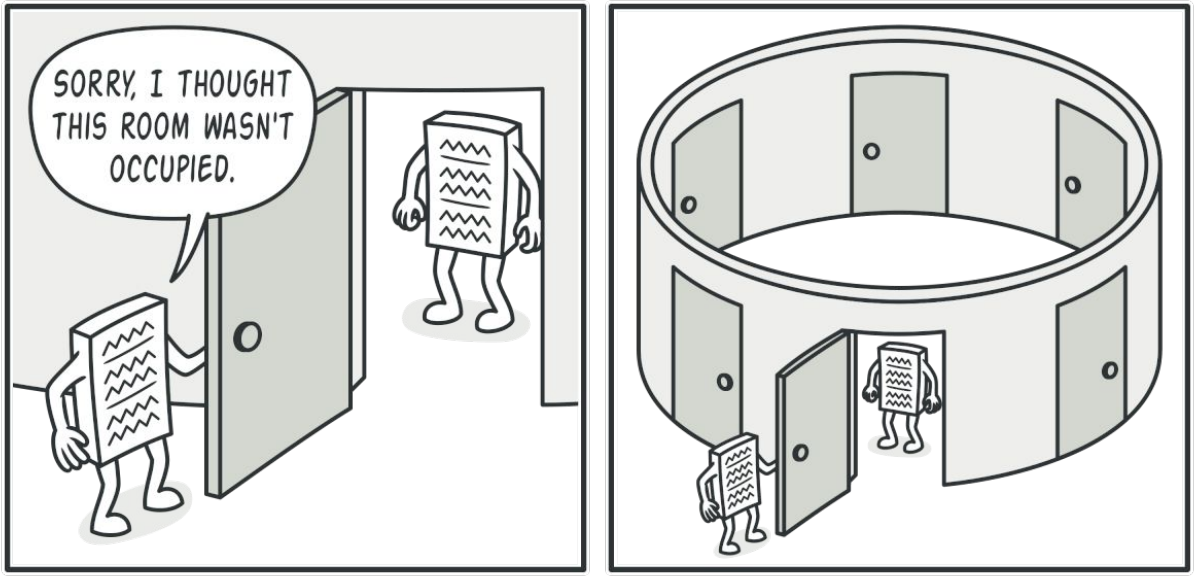


Singleton Pattern

Singleton Pattern

Problem

When you need one, and only one, instance of a type



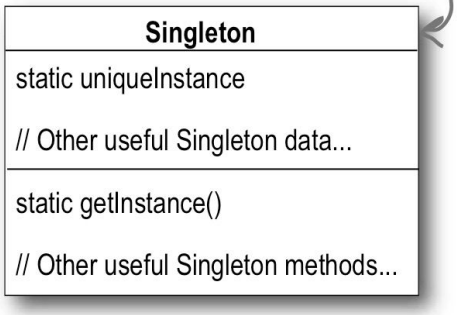
Motivation for Singleton

- Some objects should only be instantiated once:
  - Thread pools, caches, logging objects, device drivers, etc.

- Instantiating more than one of such objects may create problems
  - incorrect program behavior, resource overuse, inconsistent results

Singleton Pattern

The getInstance() method is static, which means it's a class method, so you can conveniently access this method from anywhere in your code using Singleton.getInstance(). That's just as easy as accessing a global variable, but we get benefits like lazy instantiation from the Singleton.



The uniqueInstance class variable holds our one and only instance of Singleton.

A class implementing the Singleton Pattern is more than a Singleton; it is a general purpose class with its own set of data and methods.

Example: Singleton in Java (Eager Instantiation)

```
class Singleton {
    private static Singleton uniqueInstance = new Singleton();
    private Singleton() {}
    public static Singleton getInstance() {
        return uniqueInstance;
    }
    // ...
}
```

The only created instance

private constructor

public Accessor method

Example: Singleton in Java (Lazy Instantiation)

```
class Singleton {
    private static Singleton uniqueInstance;
    private Singleton() {}
    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
    // ...
}
```

The instance is not initially created

The instance gets created lazily

Creational Patterns Quiz

References

- Freeman, E., Freenman, E., “Head First Design Pattern.” O’Rielly, 2004.
- https://home.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/rao.pdf
- Software Design Patterns