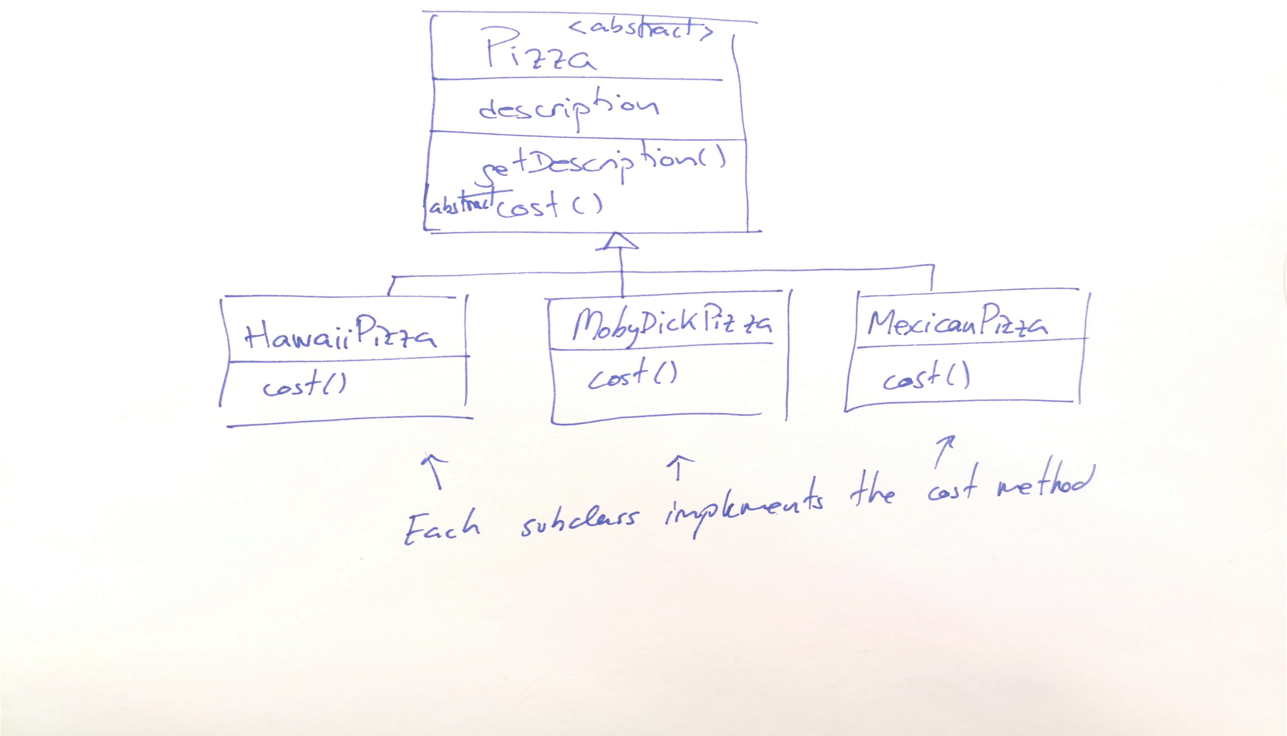
Welcome to PizzaShut

PizzaShut pizzas have made a name for itself as the fastest pizza shop around Uppsala. If you have seen one in a local corner, bike for 5 minutes and you will find another one.

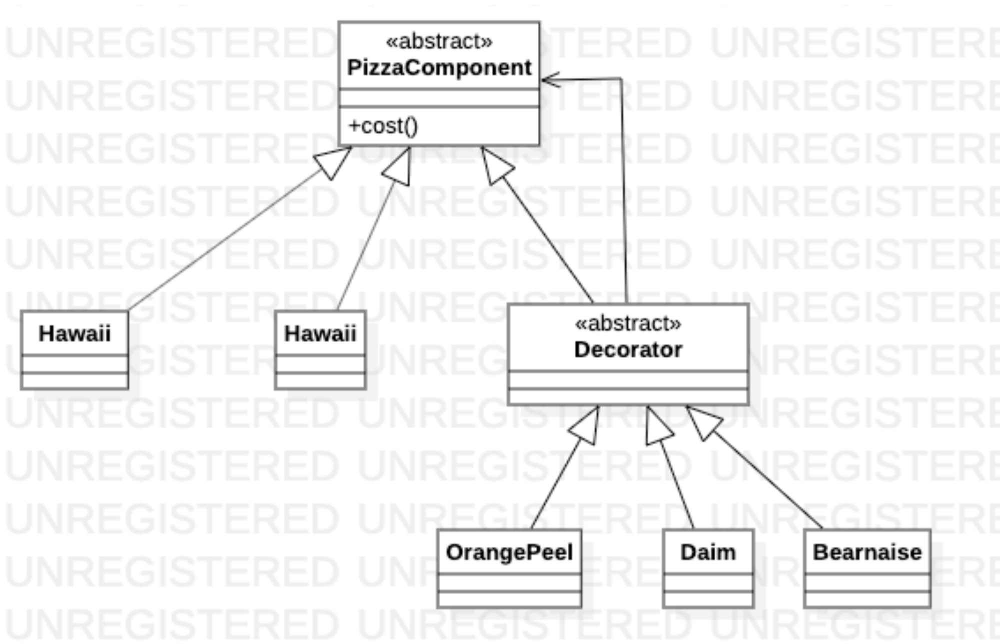
Because they have grown so quickly, they are having problems with the design of their ordering system and matching the amount of toppings that one can add to the pizzas.

Their pizza model looks as:



In addition to the included pizza ingredients, you can ask for several extra and fancy ingredients like peel of kaki, orange or lemon, chocolate, Daim pieces, etc, and top it off with a sauce, such as bearnaise or hollandaise sauce. PizzaShut will charge a bit for each of these, and a bearnaise sauce is obviously more expensive than a hollandaise sauce.

- Try as a first attempt to create everything using subclasses.
- Second attempt: add all the attributes needed and a maximum of 4 ingredients.
- Use the decorator pattern to build a pizza.


«abstract»
PizzaComponent
+cost()
Hawaii
Hawaii
«abstract»
Decorator
OrangePeel
Daim
Bearnaise

Part II: Pizza Creation

The PizzaShut system takes a string as input for the name of the selected pizza. The name of the pizza is already a valid one. How can we map the name of a pizza to the actual construction of the pizza? (implicit design pattern named Simple Factory)

Discuss how to create a pizza from its name.

Know that we know how to create a pizza, PizzaShut has a really clear model on building a pizza. All the pizzas go though some preparation, baking, cut, and ingredients. For this, we extend the Pizza abstract class with the corresponding methods, all of which return a `String`.

Discuss how each pizza is built according to its expectation (ingredients, sauces, etc).

Solution:

As per the diagram, we separate the `PizzaComponent` from the Hawaii and Mexican pizza, so that the abstract class `Pizza` has attributes (we could have done this as well in the diagram from before).

Instead of adding more functionality to a `PizzaComponent`, such as `buildDough()`, `buildSauce()`, etc, we create a `PizzaBuilder` abstract class that has that functionality. The Hawaii and Mexican pizza inherit from it and can therefore implement those features.

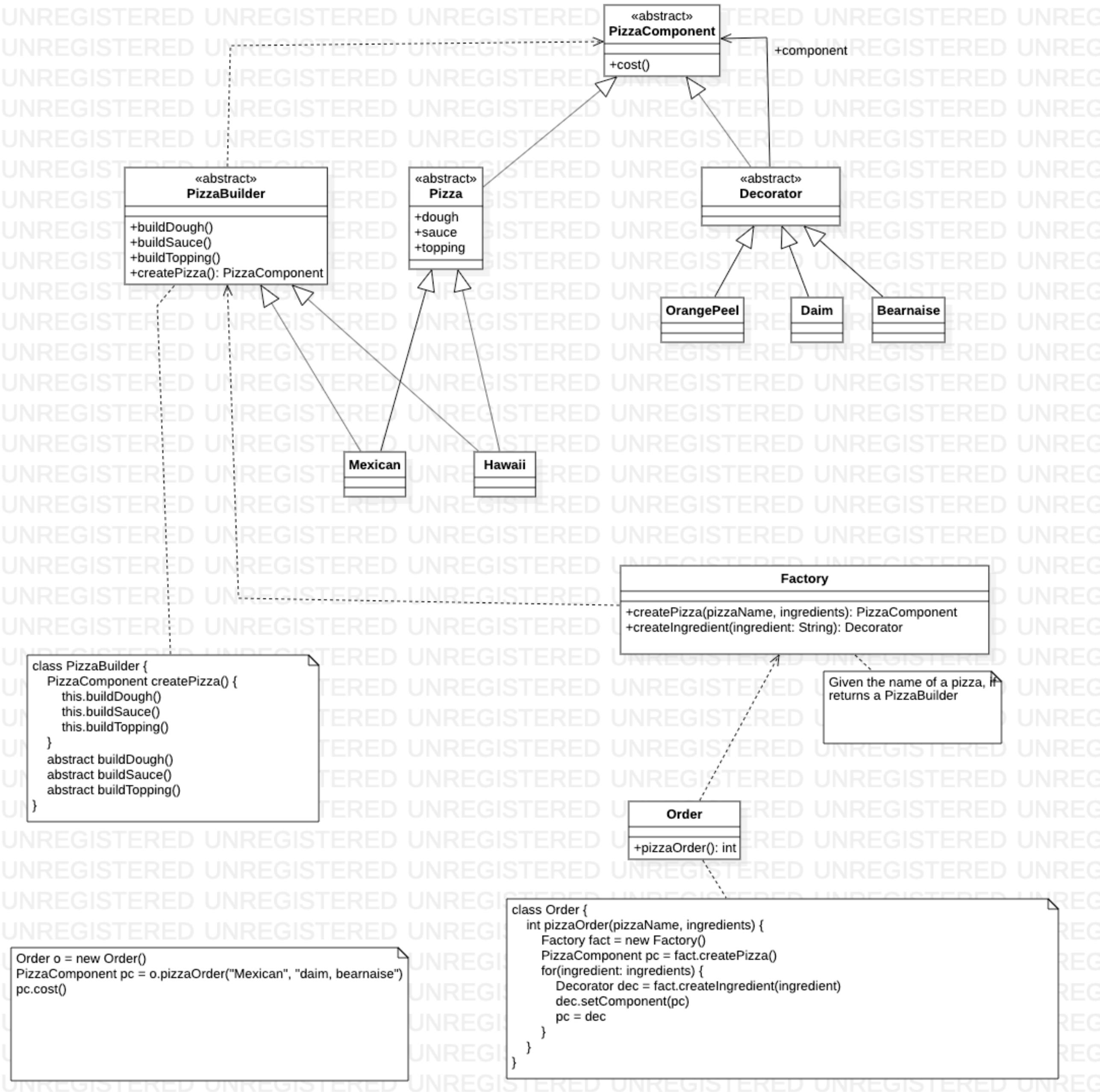
The `PizzaFactory` uses a `PizzaBuilder` in its createPizza method `(that is, new Mexican())` and returns a `PizzaComponent (from method createPizza())` once the basic pizza style is build (only contains default toppings). The factory also knows how to create ingredients which are not initialised. As in the lecture, the code would resemble:

```
public PizzaFactory{

    PizzaComponent createPizza(String pizzaName){

        PizzaComponent pc;

        if pizzaName == "Mexican"

            return new Mexican().createPizza()

        } else if () {

           …

        }

    }


   PizzaComponent createIngredient(String ingredient){ ... }

}
```

The `createPizza` method from above calls on the `createPizza` from the instance of a Mexican pizza, which calls the default pizza creation on the `PizzaBuilder` (code in the class diagram as well).

```
class Mexican extends Pizza, PizzaBuilder
    // missing implementation of abstract methods
    ...
    PizzaComponent createPizza(){
        super().createPizza()
    }
}

class PizzaBuilder{
  PizzaComponent createPizza() {
   this.buildDough()
    this.buildSauce()
    this.buildToppings()
  }
}
```

Given an `Order`, we create a pizza from its name, using the `Factory` pattern from above. Notice how the method `pizzaOrder` in `Order` decouples the creation of a `Pizza` from its ingredients by first creating the pizza with its factory, and then adding the ingredients on top of the `Pizza`. This is only possible because a factory returns a `PizzaComponent`, which is later reused using the decorator pattern to add extra ingredients. The class diagram contains some extra code for clarification.


«abstract»
PizzaComponent
+cost()
+component
«abstract»
PizzaBuilder
+buildDough()
+buildSauce()
+buildTopping()
+createPizza(): PizzaComponent
«abstract»
Pizza
+dough
+sauce
+topping
«abstract»
Decorator
OrangePeel
Daim
Bearnaise
Mexican
Hawaii
Factory
+createPizza(pizzaName, ingredients): PizzaComponent
+createIngredient(ingredient: String): Decorator
class PizzaBuilder {
PizzaComponent createPizza() {
this.buildDough()
this.buildSauce()
this.buildTopping()
}
abstract buildDough()
abstract buildSauce()
abstract buildTopping()
}
Given the name of a pizza, it returns a PizzaBuilder
Order
+pizzaOrder(): int
class Order {
int pizzaOrder(pizzaName, ingredients) {
Factory fact = new Factory()
PizzaComponent pc = fact.createPizza()
for(ingredient: ingredients) {
Decorator dec = fact.createIngredient(ingredient)
dec.setComponent(pc)
pc = dec
}
}
}
Order o = new Order()
PizzaComponent pc = o.pizzaOrder("Mexican", "daim, bearnaise")
pc.cost()