UPPSALA
UNIVERSITET

# 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.

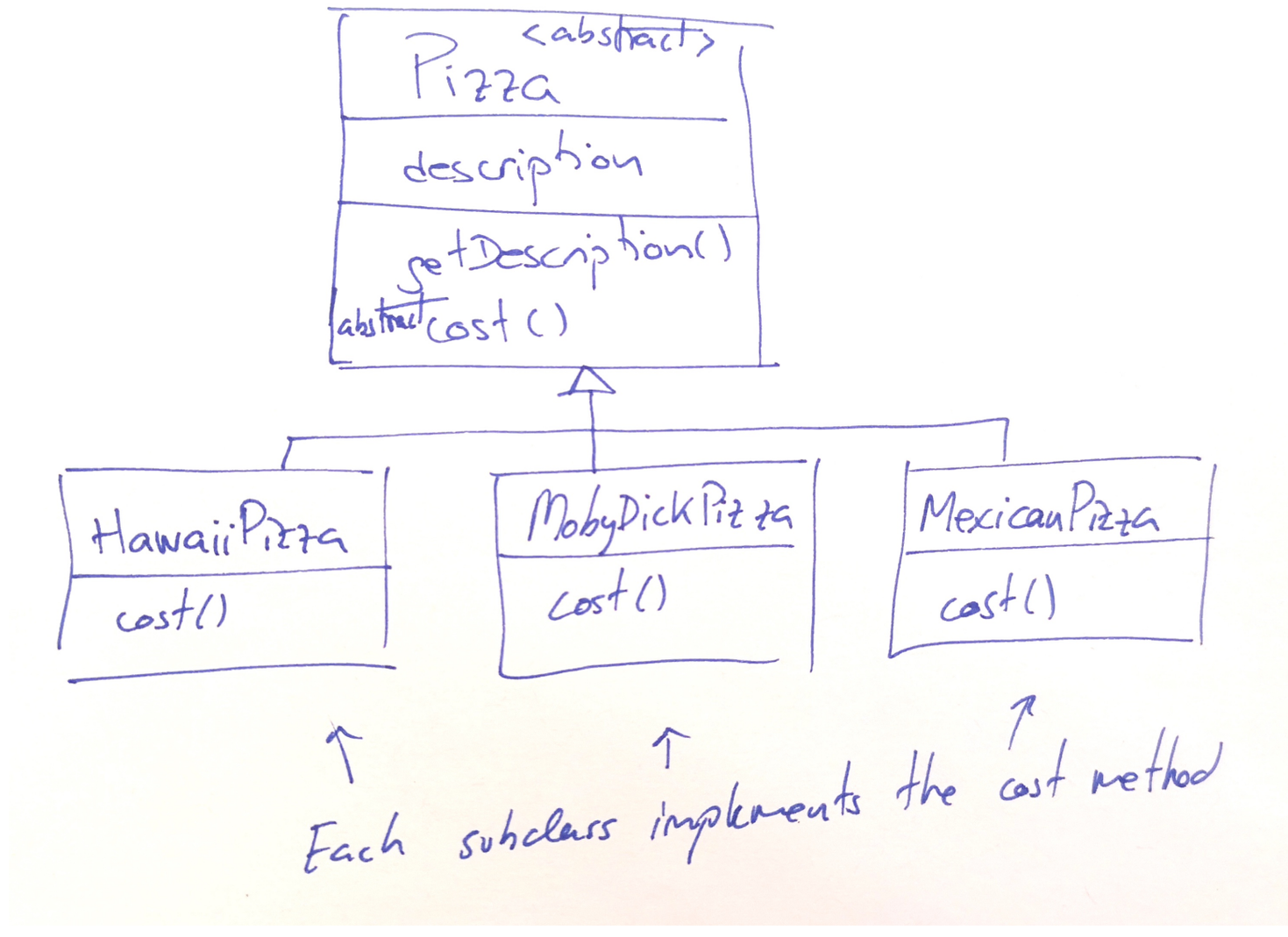
UPPSALA
UNIVERSITET

# PizzaShut

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.

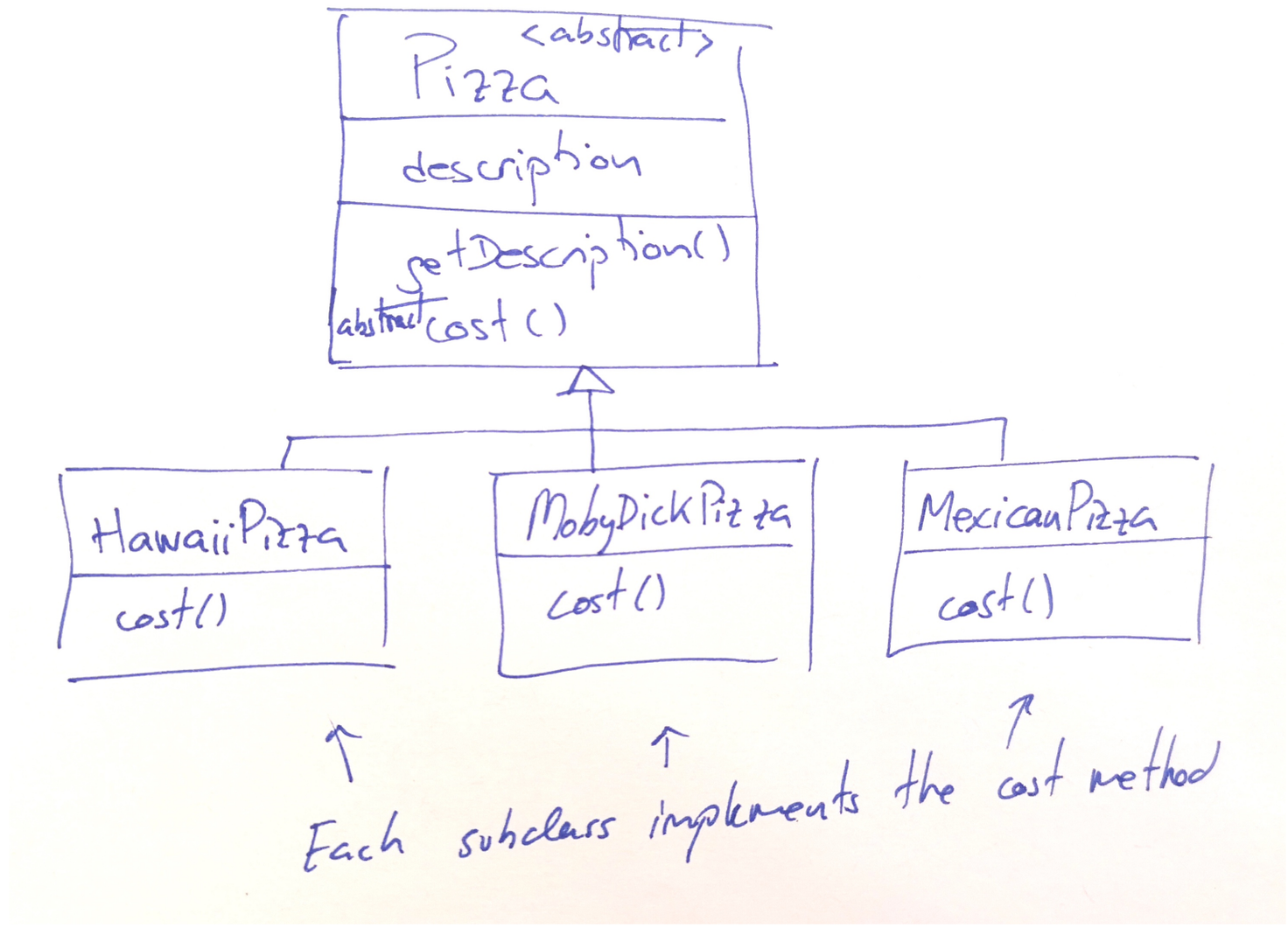
UPPSALA
UNIVERSITET

# PizzaShut



UPPSALA
UNIVERSITET

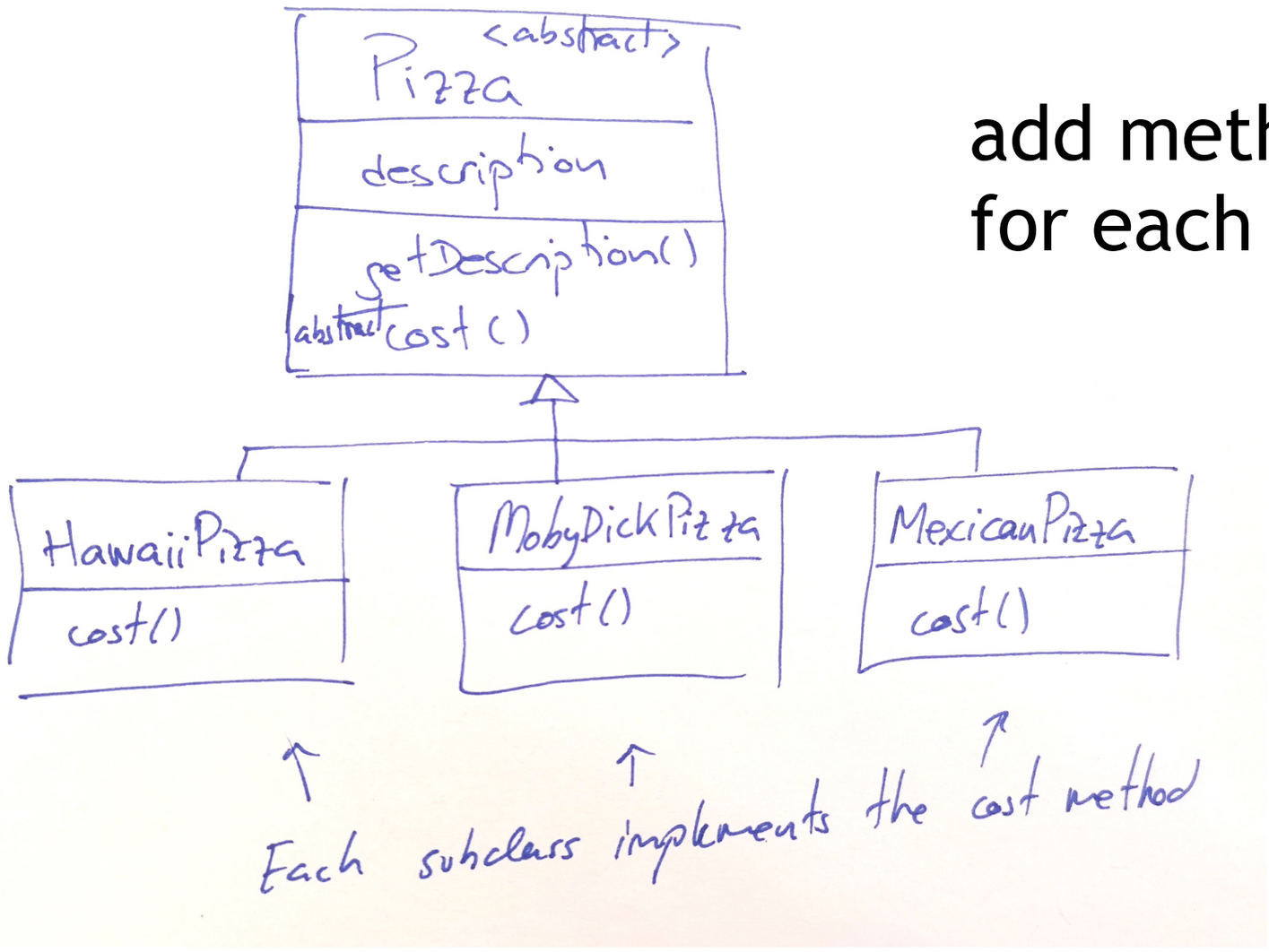
# Try as a first attempt to create everything using subclasses.



UPPSALA
UNIVERSITET

Second attempt:

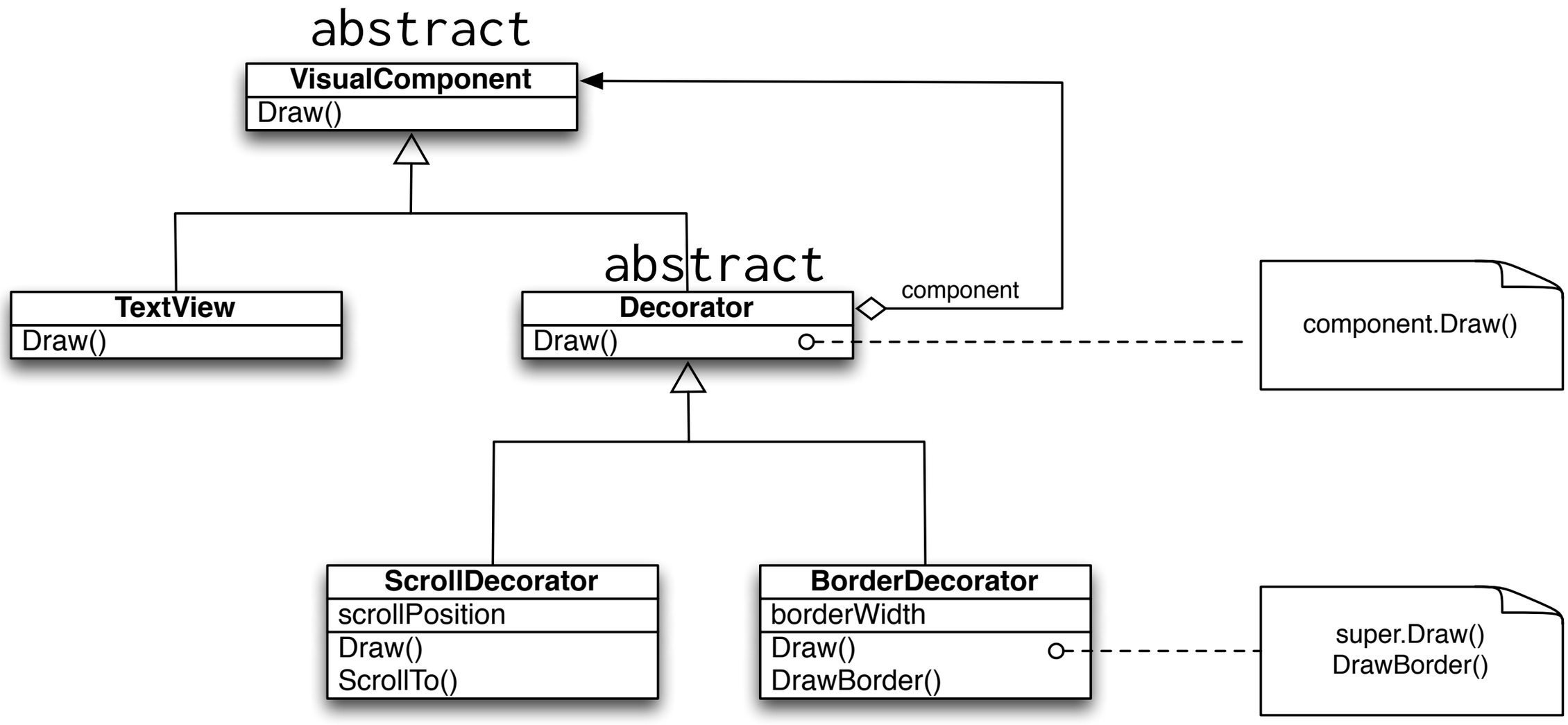
add all the attributes needed in the supper class and a maximum of 4 ingredients.



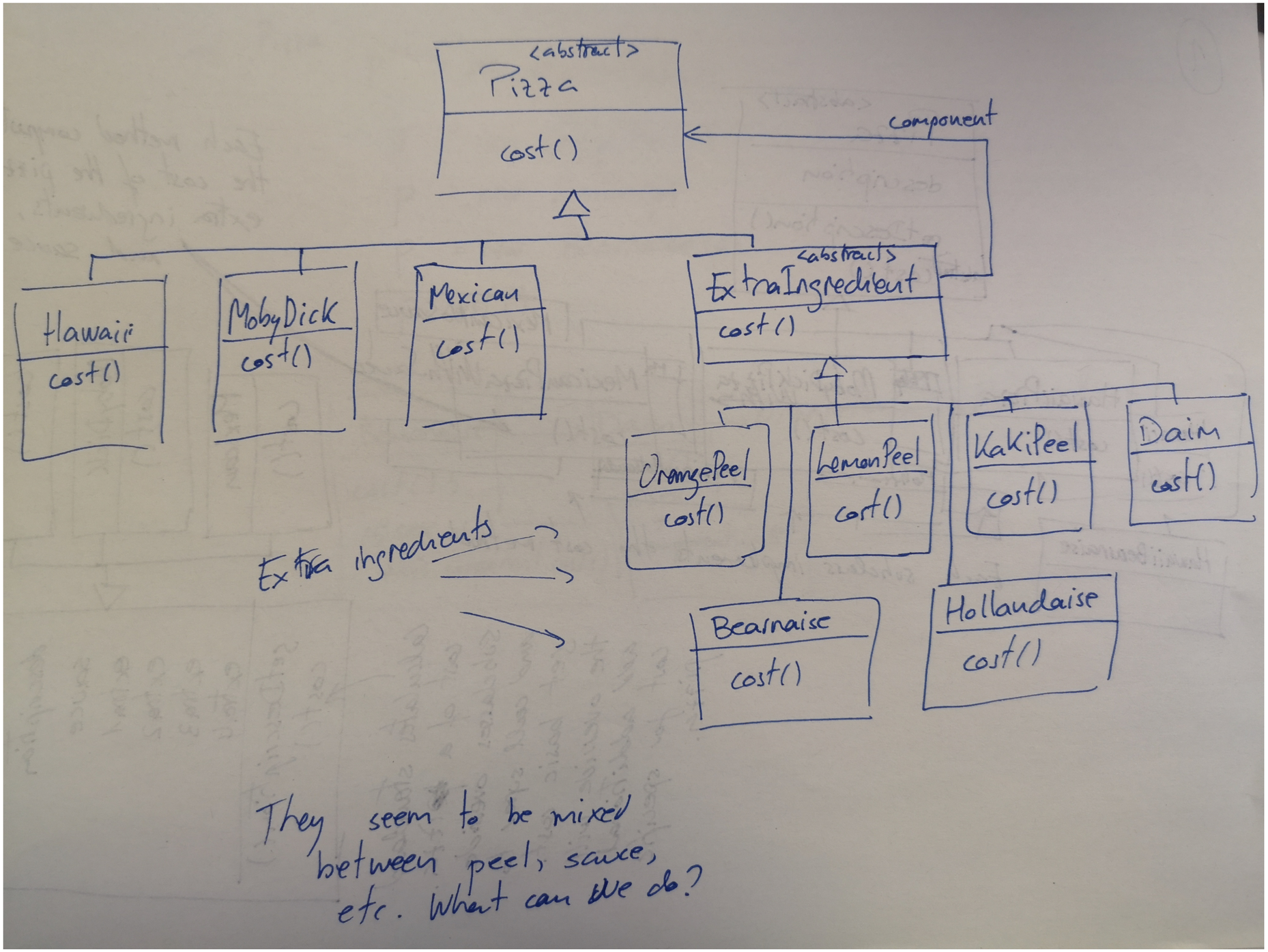

add methods and attributes for each ingredient

UPPSALA
UNIVERSITET

# Use the decorator pattern to build a pizza.



UPPSALA
UNIVERSITET



UPPSALA
UNIVERSITET

# PizzaShut: 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.

UPPSALA
UNIVERSITET

# PizzaShut: Pizza Creation

Know that we know how to create a pizza, PizzaShut has a really clear model on building a pizza. All the pizzas go though:

```
preparation(): String,
baking(): String,
cut(): String,
ingredients(): String.
```

For this, we extend the Pizza abstract class with the corresponding methods. Discuss how each pizza is built according to its expectation (ingredients, sauces, etc). Which design pattern?

# BUILDER

separate the construction of a complex objects from its representation so that the same construction process can create different representations.

# BUILDER

separate the construction of a complex objects from its representation so that the same construction process can create different representations.

Builder =

VAPIANO®
PASTA | PIZZA | BAR

# BUILDER

```
class Pizza {
  private String dough = "";
  private String sauce = "";
  private String topping = "";

  public void setDough(String dough)     {
      this.dough = dough;
  }
  public void setSauce(String sauce)     {
      this.sauce = sauce;
  }
  public void setTopping(String topping) {
    this.topping = topping;
  }
}

/* "Abstract Builder" */
abstract class PizzaBuilder {
  protected Pizza pizza;

  public Pizza getPizza() { return pizza; }
  public void createNewPizzaProduct() { pizza = new Pizza(); }

  public abstract void buildDough();
  public abstract void buildSauce();
  public abstract void buildTopping();
}
```

Recipe

# BUILDER

```
class Pizza {
  private String dough = "";
  private String sauce = "";
  private String topping = "";

  public void setDough(String dough)     {
      this.dough = dough;
  }
  public void setSauce(String sauce)     {
      this.sauce = sauce;
  }
  public void setTopping(String topping) {
    this.topping = topping;
  }
}

/* "Abstract Builder" */
abstract class PizzaBuilder {
  protected Pizza pizza;

  public Pizza getPizza() { return pizza; }
  public void createNewPizzaProduct() { pizza = new Pizza(); }

  public abstract void buildDough();
  public abstract void buildSauce();
  public abstract void buildTopping();
}
```

Recipe

```
/* "ConcreteBuilder" */
class HawaiianPizzaBuilder extends PizzaBuilder {
  public void buildDough()   { pizza.setDough("cross"); }
  public void buildSauce()   { pizza.setSauce("mild"); }
  public void buildTopping() {
      pizza.setTopping("ham+pineapple");
  }
}
```

```
/* "ConcreteBuilder" */
class SpicyPizzaBuilder extends PizzaBuilder {
  // specialised version, similar to HawaiianPizzaBuilder
}
```

# LDER

```
/* "Director" */
class Waiter {
  private PizzaBuilder pizzaBuilder;

  public void setPizzaBuilder(PizzaBuilder pb) {
    pizzaBuilder = pb;
  }
  public Pizza getPizza() {
    return pizzaBuilder.getPizza();
  }

  public void constructPizza() {
    pizzaBuilder.createNewPizzaProduct();
    pizzaBuilder.buildDough();
    pizzaBuilder.buildSauce();
    pizzaBuilder.buildTopping();
  }
}
```

```
/* "ConcreteBuilder" */
class HawaiianPizzaBuilder extends PizzaBuilder {
  public void buildDough()   { pizza.setDough("cross"); }
  public void buildSauce()   { pizza.setSauce("mild"); }
  public void buildTopping() {
      pizza.setTopping("ham+pineapple");
  }
}

/* "ConcreteBuilder" */
class SpicyPizzaBuilder extends PizzaBuilder {
  // specialised version, similar to HawaiianPizzaBuilder
}
```

Recipe

```
/* "Abstract Builder" */
abstract class PizzaBuilder {
  protected Pizza pizza;

  public Pizza getPizza() { return pizza; }
  public void createNewPizzaProduct() { pizza = new Pizza(); }

  public abstract void buildDough();
  public abstract void buildSauce();
  public abstract void buildTopping();
}
```

# LDER

```
/* "Director" */
class Waiter {
  private PizzaBuilder pizzaBuilder;

  public void setPizzaBuilder(PizzaBuilder pb) {
    pizzaBuilder = pb;
  }
  public Pizza getPizza() {
    return pizzaBuilder.getPizza();
  }

  public void constructPizza() {
    pizzaBuilder.createNewPizzaProduct();
    pizzaBuilder.buildDough();
    pizzaBuilder.buildSauce();
    pizzaBuilder.buildTopping();
  }
}
```

```
/* "ConcreteBuilder" */
class HawaiianPizzaBuilder extends PizzaBuilder {
  public void buildDough()   { pizza.setDough("cross"); }
  public void buildSauce()   { pizza.setSauce("mild"); }
  public void buildTopping() {
      pizza.setTopping("ham+pineapple");
  }
}

/* "ConcreteBuilder" */
class SpicyPizzaBuilder extends PizzaBuilder {
  // specialised version, similar to HawaiianPizzaBuilder
}
```

```
/* "Abstract Builder" */
abstract class PizzaBuilder {
  protected Pizza pizza;

  public Pizza getPizza() { return pizza; }
  public void createNewPizzaProduct() { pizz

  public abstract void buildDough();
  public abstract void buildSauce();
  public abstract void buildTopping();
}
```

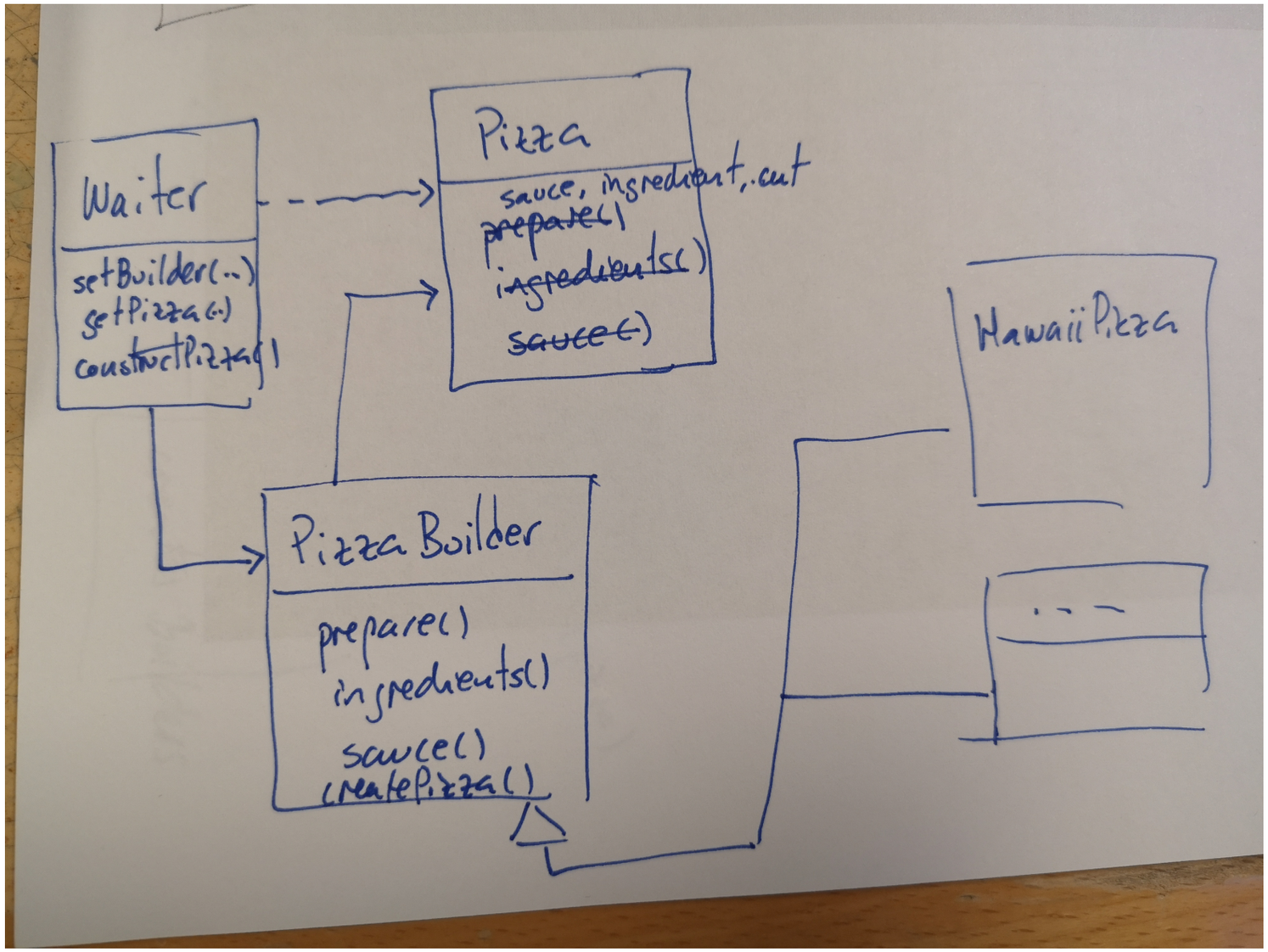
Recipe

```
class BuilderExample {
  public static void main(String[] args) {
    Waiter waiter = new Waiter();
    PizzaBuilder hawaiian_pizzabuilder = new HawaiianPizzaBuilder();
    PizzaBuilder spicy_pizzabuilder = new SpicyPizzaBuilder();

    waiter.setPizzaBuilder( hawaiian_pizzabuilder );
    waiter.constructPizza();

    Pizza pizza = waiter.getPizza();
  }
}
```

UPPSALA
UNIVERSITET

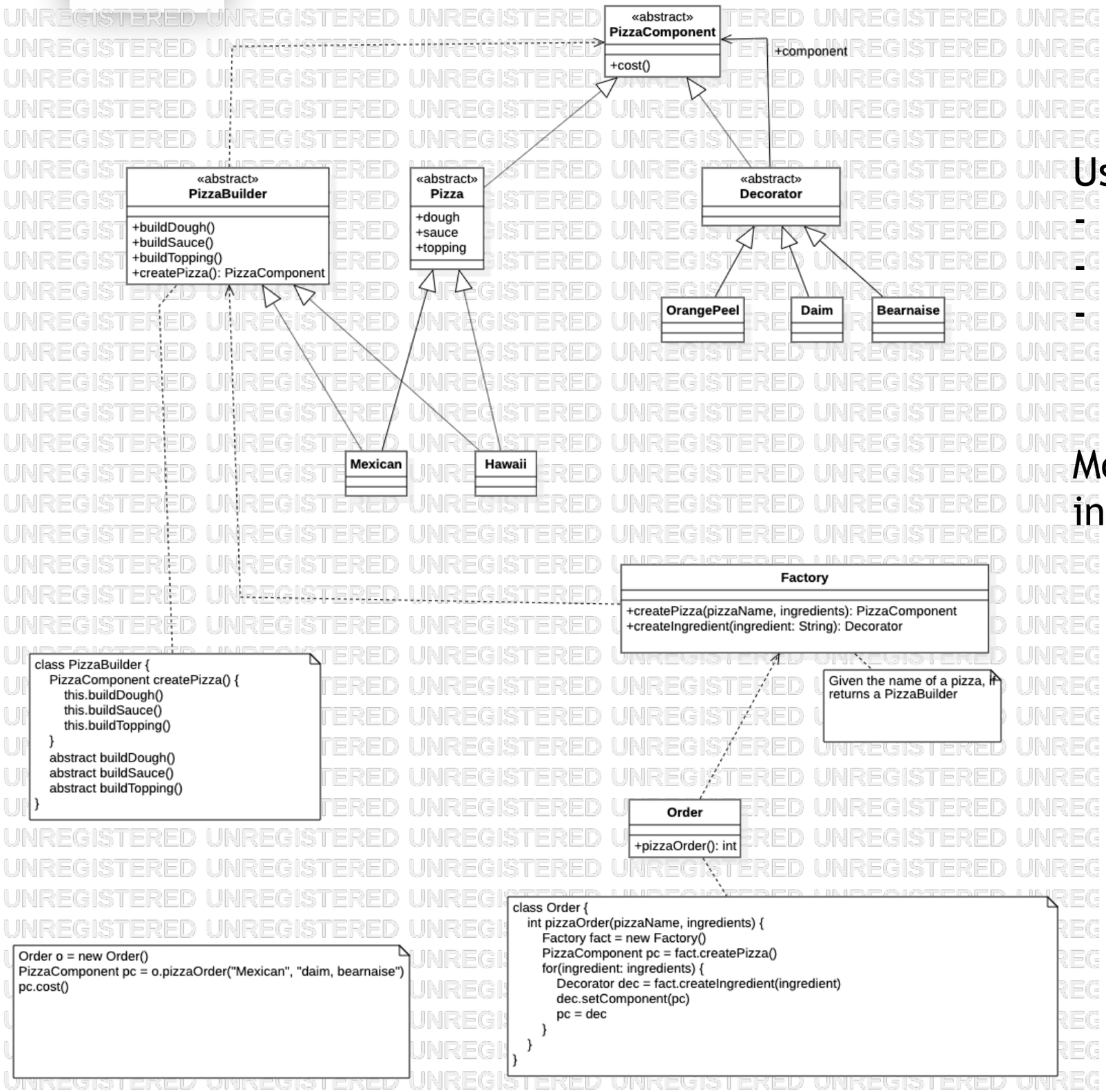
# PizzaShut: Pizza Creation

```
preparation(): String,
baking(): String,
cut(): String,
ingredients(): String
```



UPPSALA UNIVERSITET

# PizzaShut: Pizza Creation




Uses the following design patterns:

- Decorator,
- Factory, and
- Builder

More details in the document in studentportalen, Seminar1.pdf

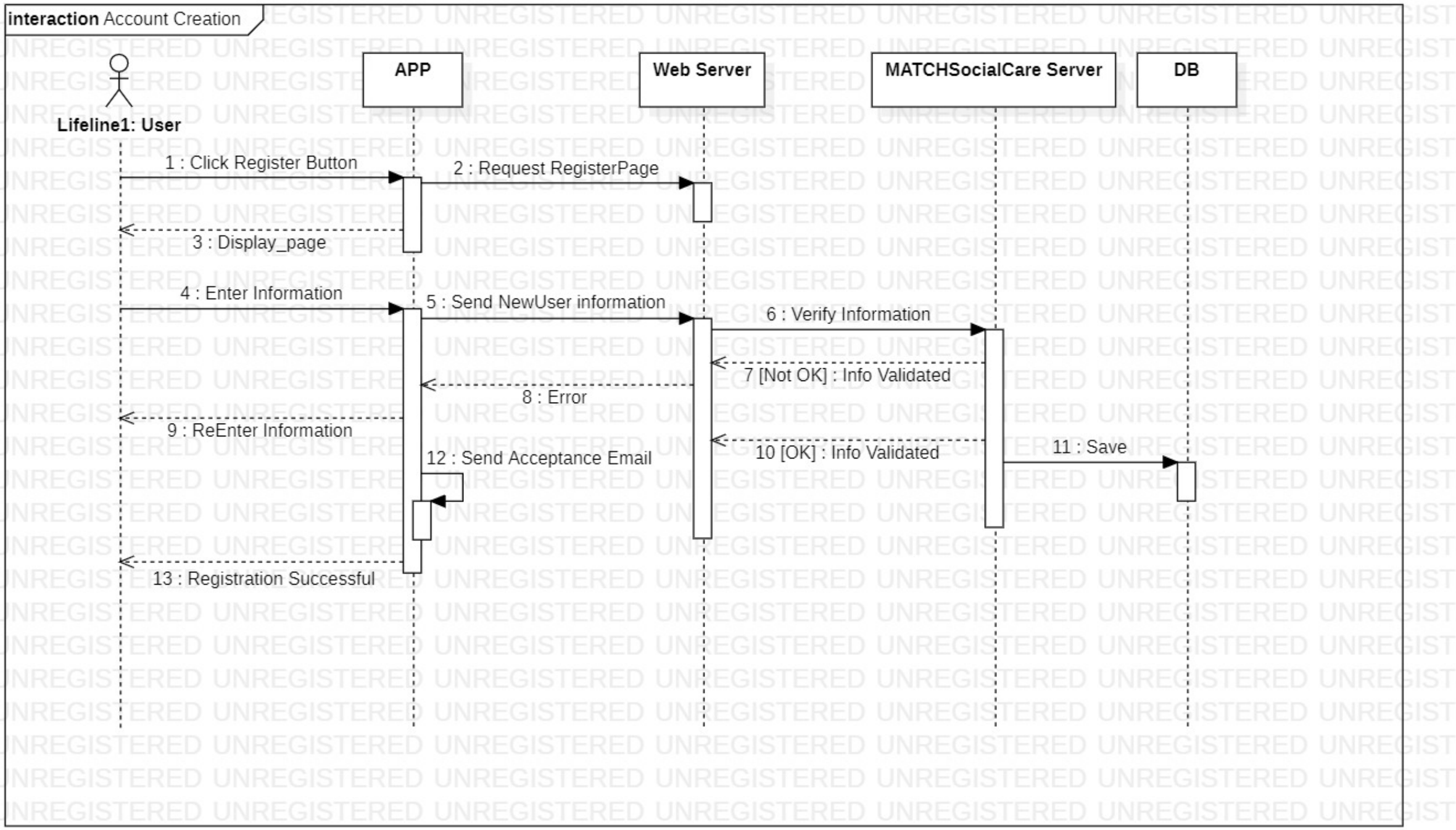
UPPSALA
UNIVERSITET

# SEMINAR 2

0. Achievements (questions?)
1. Common Pitfalls
2. PizzaShut is back!

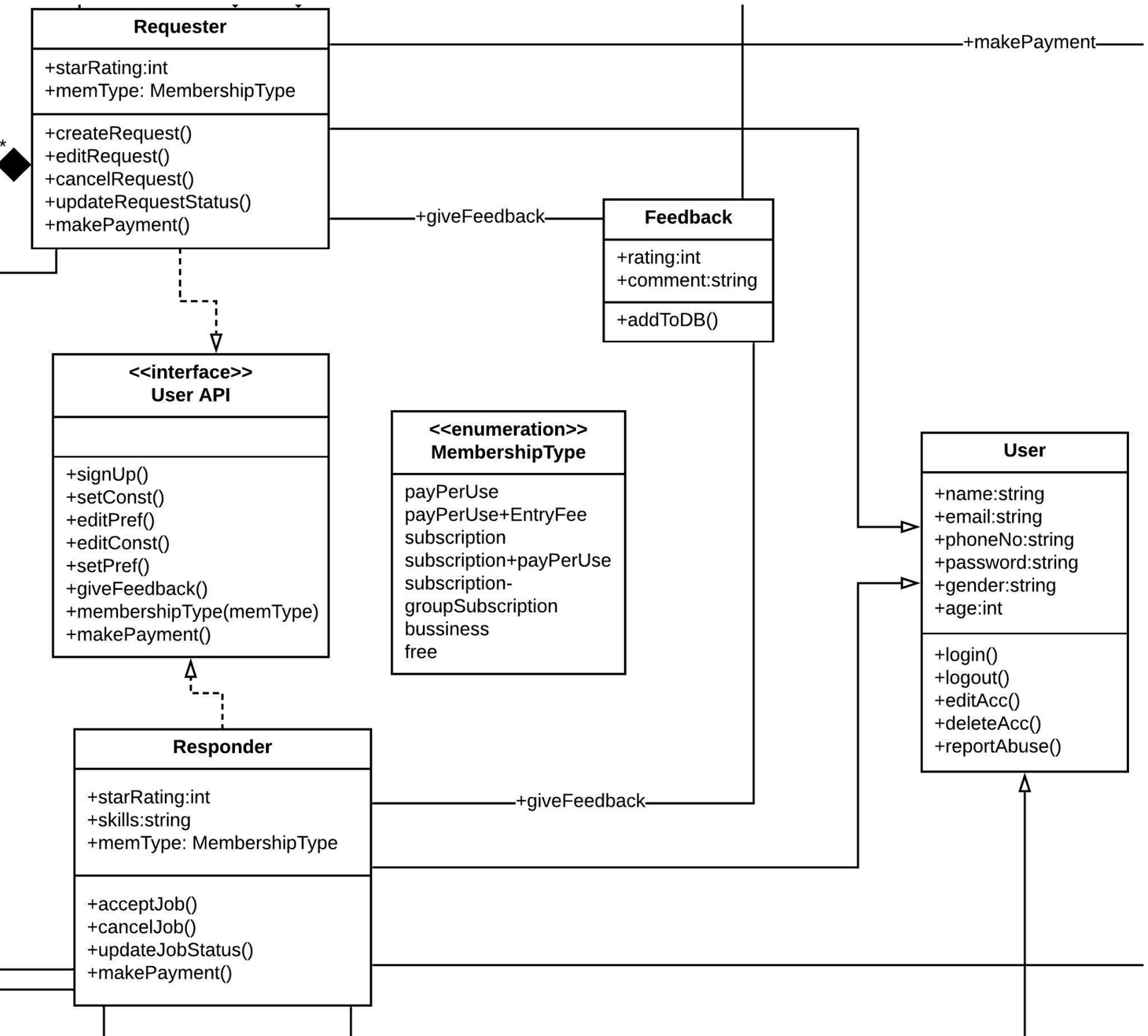
UPPSALA UNIVERSITET

# Behavioural Pitfalls



UPPSALA
UNIVERSITET

# Class Diagram Pitfalls



UPPSALA
UNIVERSITET

# PizzaNotification

The PizzaShut store system has implemented a pick-up method where clients get a device and a number after ordering a pizza. When the pizza is ready, the device shows your number. This system is not ideal but works just fine, even when they spam everyone and makes them check their number.

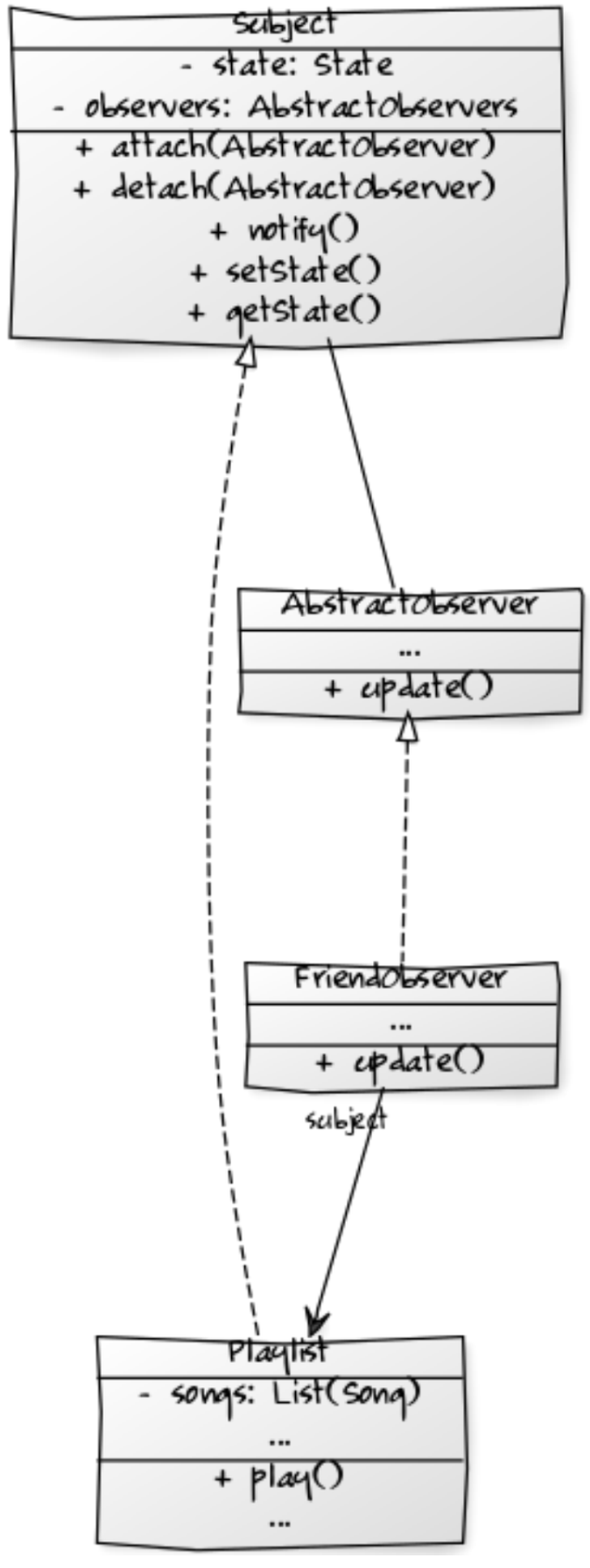
Discuss how would you design such a system?

UPPSALA
UNIVERSITET

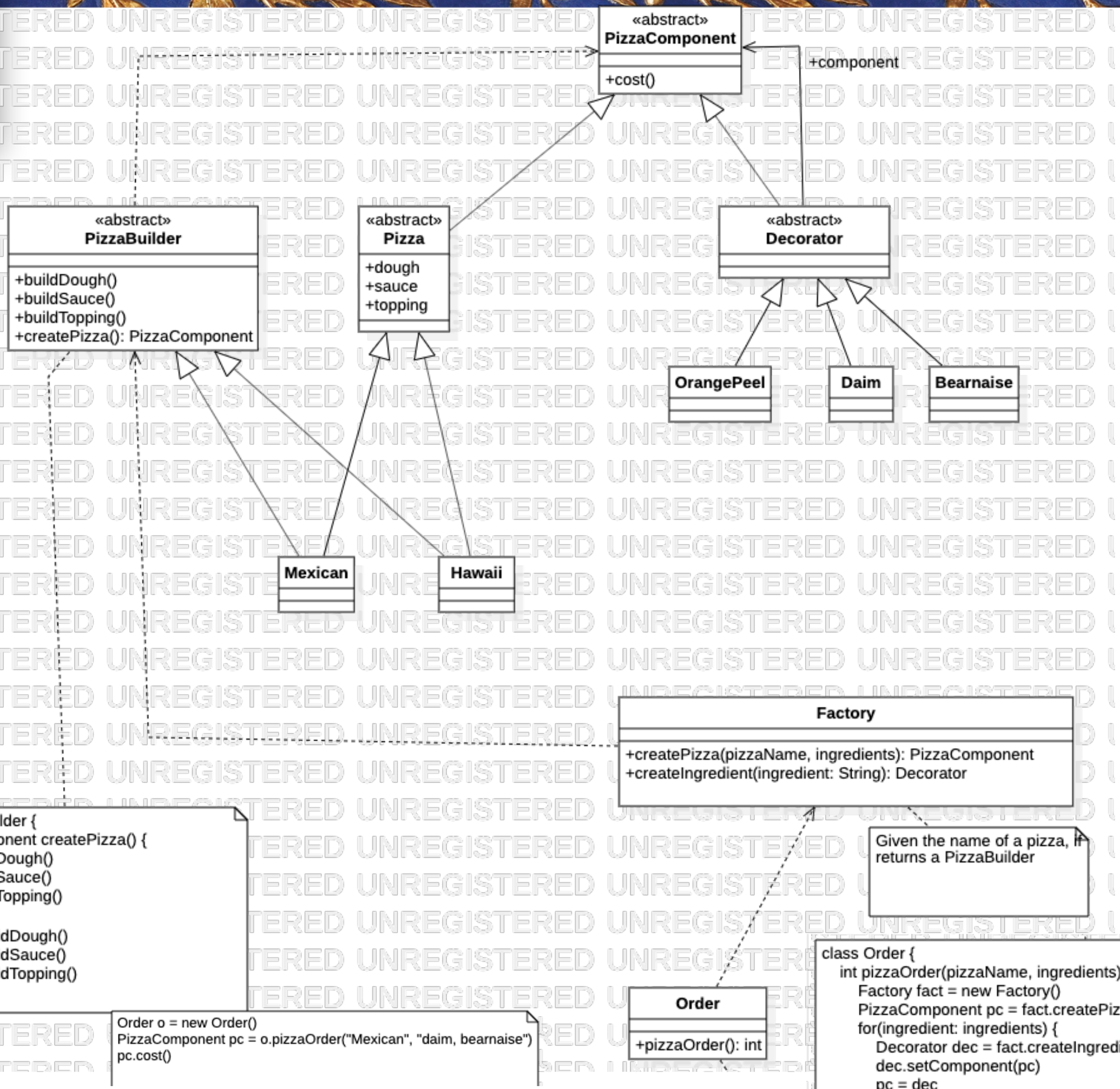
# PizzaNotification

The PizzaShut store system has implemented a pick-up method where clients get a device and a number after ordering a pizza. When the pizza is ready, the device shows your number. This system is not ideal but works just fine, even when they spam everyone and makes them check their name.

Discuss how would you design such a system?



UPPSALA UNIVERSITET



UPPSALA
UNIVERSITET

# Thank you

UPPSALA
UNIVERSITET

# Pizza Language


The PizzaShut wants to take out of the market to Fooodora and InlinePizza. Their previous ordering system relied on receiving data in certain order and could not be customised to the needs of its clients, e.g. `Order("mexican+extra-jalapeños+no-cheese")`

To kill the competition, they adopt Cook, a programming language customised for cooking. The order now receives  programs and interprets them as recipes in the PizzaBuilder.

UPPSALA UNIVERSITET

# Pizza Language

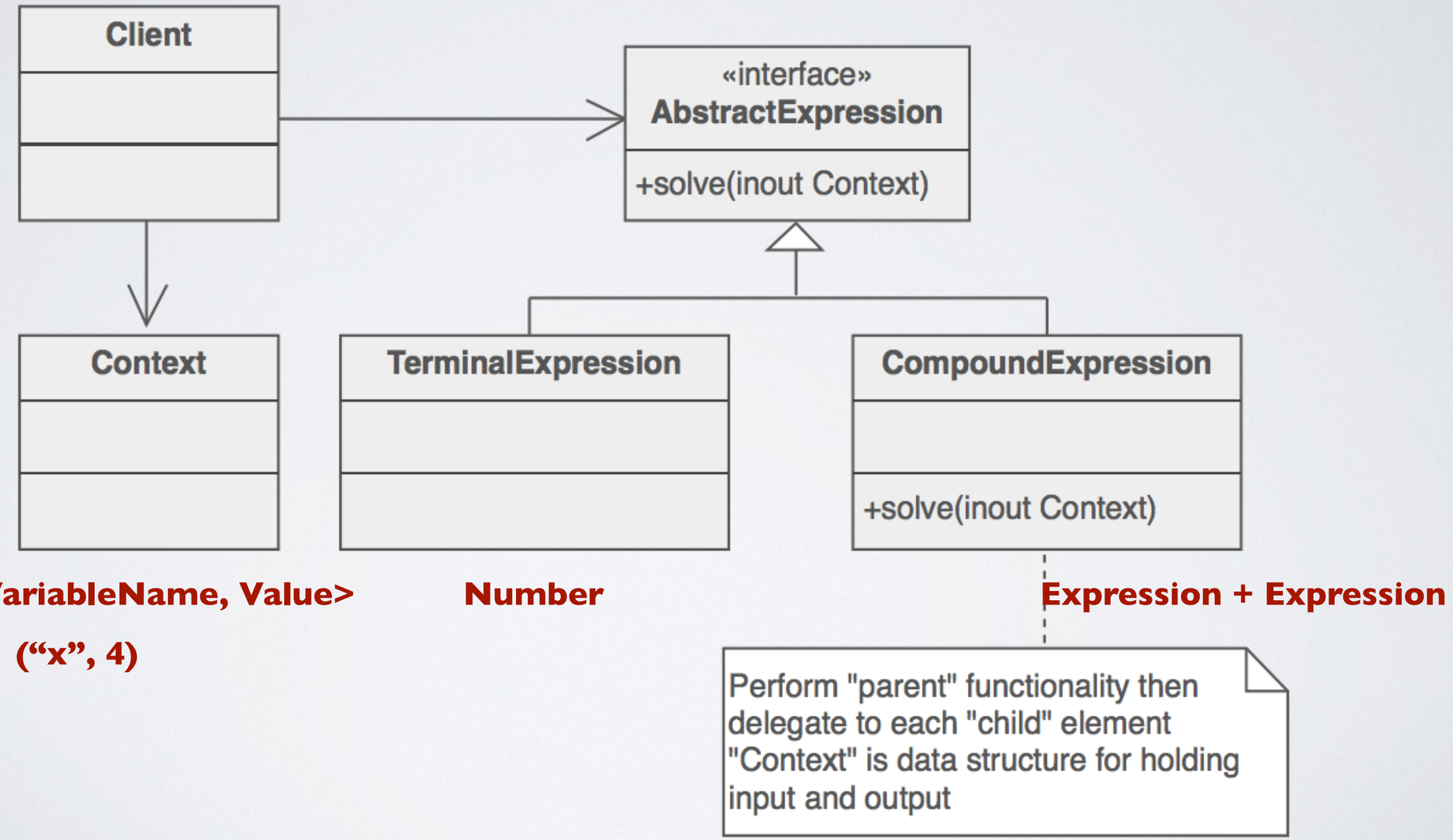
```
info {
  title: "Chocolate Chip Pizza"
  description: "Best chocolate chip pizza ever"
  time: 45
  servings: 3
}

ingredients{
  granulated-sugar:0.75c
  brown-sugar:0.75c;
  butter:1c <prep: "soften">
  flour:2.25c <ext: "all-purpose"> <brand: "GoldMetal">
  salt:0.5tsp
  nuts:1c <prep: "fine chop">
  chocolate-chips: 12oz <ext: “semi-sweet">
  cheese: taco-cheese
  sauce: tomate-with-sugar
}
```

```
directions{
  oven: heat until 375F
  dough:
    mix (sugar, butter, vanilla, egg)
    stir (flour, baking-soda, salt)
    place <size:tsp> <dist: 2in> on
          cooking-sheet <ext: ungreased>

cookies:
  bake dough on oven until
       (time:8 or color:light-brown)
  cool <time:min>;
  place on wire-rack
  cool
}

pizza: …
```

# INTERPRETER

Interpreter – Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences from the language.

# INTERPRETER



# INTERPRETER

```
interface Expression {
    public int interpret(Map<String,Expression> variables);
}
```

```
class Plus implements Expression {
    Expression leftOperand;
    Expression rightOperand;
    public Plus(Expression left, Expression right) {
        leftOperand = left;
        rightOperand = right;
    }

    public int interpret(Map<String,Expression> variables)  {
        return leftOperand.interpret(variables) + rightOperand.interpret(variables);
    }
}
```

# INTERPRETER

```
interface Expression {
    public int interpret(Map<String,Expression> variables);
}


class Number implements Expression {
    private int number;
    public Number(int number)       { this.number = number; }
    public int interpret(Map<String,Expression> variables)  { return number; }
}


class Variable implements Expression {
  private String name;
  public Variable(String name)       { this.name = name; }
  public int interpret(Map<String,Expression> variables)  {
   return !variables.get(name) ? 0 : variables.get(name).interpret(variables);
  }
}
```

UPPSALA UNIVERSITET

# Pizza Language

```
info {
  title: "Chocolate Chip Pizza"
  description: "Best chocolate chip pizza ever"
  time: 45
  servings: 3
}

ingredients{
  granulated-sugar:0.75c
  brown-sugar:0.75c;
  butter:1c <prep: "soften">
  flour:2.25c <ext: "all-purpose"> <brand: "GoldMetal">
  salt:0.5tsp
  nuts:1c <prep: "fine chop">
  chocolate-chips: 12oz <ext: “semi-sweet">
  cheese: taco-cheese
  sauce: tomate-with-sugar
}
```

```
directions{
  oven: heat until 375F
  dough:
    mix (sugar, butter, vanilla, egg)
    stir (flour, baking-soda, salt)
    place <size:tsp> <dist: 2in> on
          cooking-sheet <ext: ungreased>

cookies:
  bake dough on oven until
       (time:8 or color:light-brown)
  cool <time:min>;
    place on wire-rack
    cool
}

sauce { … }

pizza-ingredients { … }
```

UPPSALA UNIVERSITET

# Pizza Language

```
info {
  title: "Chocolate Chip Pizza"
  description: "Best chocolate chip pizza ever"
  time: 45
  servings: 3
}

ingredients{
  granulated-sugar:0.75c
  brown-sugar:0.75c;
  butter:1c <prep: "soften">
  flour:2.25c <ext: "all-purpose"> <brand: "GoldMetal">
  salt:0.5tsp
  nuts:1c <prep: "fine chop">
  chocolate-chips: 12oz <ext: “semi-sweet">
  cheese: taco-cheese
  sauce: tomate-with-sugar
}
```

```
directions{
  oven: heat until 375F
  dough:
    mix (sugar, butter, vanilla, egg)
    stir (flour, baking-soda, salt)
    place <size:tsp> <dist: 2in> on
          cooking-sheet <ext: ungreased>

cookies:
  bake dough on oven until
       (time:8 or color:light-brown)
  cool <time:min>;
  place on wire-rack
  cool
}

sauce { … }

pizza-ingredients { … }
```

## Interpreter Patter:

UPPSALA UNIVERSITET

# Pizza Language

```
info {
  title: "Chocolate Chip Pizza"
  description: "Best chocolate chip pizza ever"
  time: 45
  servings: 3
}

ingredients{
  granulated-sugar:0.75c
  brown-sugar:0.75c;
  butter:1c <prep: "soften">
  flour:2.25c <ext: "all-purpose"> <brand: "GoldMetal">
  salt:0.5tsp
  nuts:1c <prep: "fine chop">
  chocolate-chips: 12oz <ext: “semi-sweet">
  cheese: taco-cheese
  sauce: tomate-with-sugar
}
```

```
directions{
  oven: heat until 375F
  dough:
    mix (sugar, butter, vanilla, egg)
    stir (flour, baking-soda, salt)
    place <size:tsp> <dist: 2in> on
          cooking-sheet <ext: ungreased>

cookies:
  bake dough on oven until
       (time:8 or color:light-brown)
  cool <time:min>;
  place on wire-rack
  cool
}

sauce { … }

pizza-ingredients { … }
```

## Interpreter Patter:

1. Localise elements that form an Abstract Syntax Tree (ABS)

UPPSALA UNIVERSITET

# Pizza Language

```
info {
  title: "Chocolate Chip Pizza"
  description: "Best chocolate chip pizza ever"
  time: 45
  servings: 3
}

ingredients{
  granulated-sugar:0.75c
  brown-sugar:0.75c;
  butter:1c <prep: "soften">
  flour:2.25c <ext: "all-purpose"> <brand: "GoldMetal">
  salt:0.5tsp
  nuts:1c <prep: "fine chop">
  chocolate-chips: 12oz <ext: “semi-sweet">
  cheese: taco-cheese
  sauce: tomate-with-sugar
}
```

```
directions{
  oven: heat until 375F
  dough:
    mix (sugar, butter, vanilla, egg)
    stir (flour, baking-soda, salt)
    place <size:tsp> <dist: 2in> on
          cooking-sheet <ext: ungreased>

cookies:
  bake dough on oven until
       (time:8 or color:light-brown)
  cool <time:min>;
    place on wire-rack
    cool
}

sauce { … }

pizza-ingredients { … }
```

## Interpreter Patter:

1. Localise elements that form an Abstract Syntax Tree (ABS)
2. Tell me how to create an ABS (give an example for the recipe)

UPPSALA UNIVERSITET

# Pizza Language

```
info {
  title: "Chocolate Chip Pizza"
  description: "Best chocolate chip pizza ever"
  time: 45
  servings: 3
}

ingredients{
  granulated-sugar:0.75c
  brown-sugar:0.75c;
  butter:1c <prep: "soften">
  flour:2.25c <ext: "all-purpose"> <brand: "GoldMetal">
  salt:0.5tsp
  nuts:1c <prep: "fine chop">
  chocolate-chips: 12oz <ext: “semi-sweet">
  cheese: taco-cheese
  sauce: tomate-with-sugar
}
```

```
directions{
  oven: heat until 375F
  dough:
    mix (sugar, butter, vanilla, egg)
    stir (flour, baking-soda, salt)
    place <size:tsp> <dist: 2in> on
          cooking-sheet <ext: ungreased>

cookies:
  bake dough on oven until
       (time:8 or color:light-brown)
  cool <time:min>;
    place on wire-rack
    cool
}

sauce { … }

pizza-ingredients { … }
```

## Interpreter Patter:

1. Localise elements that form an Abstract Syntax Tree (ABS)
2. Tell me how to create an ABS (give an example for the recipe)
3. How to add it to the system?

UPPSALA UNIVERSITET

# Pizza Language

```
info {
  title: "Chocolate Chip Pizza"
  description: "Best chocolate chip pizza ever"
  time: 45
  servings: 3
}

ingredients{
  granulated-sugar:0.75c
  brown-sugar:0.75c;
  butter:1c <prep: "soften">
  flour:2.25c <ext: "all-purpose"> <brand: "GoldMetal">
  salt:0.5tsp
  nuts:1c <prep: "fine chop">
  chocolate-chips: 12oz <ext: “semi-sweet">
  cheese: taco-cheese
  sauce: tomate-with-sugar
}
```

```
directions{
  oven: heat until 375F
  dough:
    mix (sugar, butter, vanilla, egg)
    stir (flour, baking-soda, salt)
    place <size:tsp> <dist: 2in> on
          cooking-sheet <ext: ungreased>

cookies:
  bake dough on oven until
       (time:8 or color:light-brown)
  cool <time:min>;
  place on wire-rack
  cool
}

sauce { … }

pizza-ingredients { … }
```

## Interpreter Patter:

1. Localise elements that form an Abstract Syntax Tree (ABS)
2. Tell me how to create an ABS (give an example for the recipe)
3. How to add it to the system?
4. Place it in the builder

UPPSALA
UNIVERSITET

«abstract»
**PizzaComponent**
+cost()

+component

«abstract»
**PizzaBuilder**
+buildDough()
+buildSauce()
+buildTopping()
+createPizza(): PizzaComponent

«abstract»
**Pizza**
+dough
+sauce
+topping

«abstract»
**Decorator**

**Ingredient**
+name
+quantity

«abstract»
**Recipe**
+interpret()

**OrangePeel**

**Daim**

**Bearnaise**

**Mexican**

**Hawaii**

**CookBuilder**
+recipe
+interpret()

**Butter**

**Flour**

**Salt**

**Mix**
+ingred1
+ingred2

**Stir**
+ingred1
+ingred2

**Factory**
+createPizza(pizzaName, ingredients): PizzaComponent
+createIngredient(ingredient: String): Decorator
+createPizza(cook)

Given the name of a pizza, it returns a PizzaBuilder

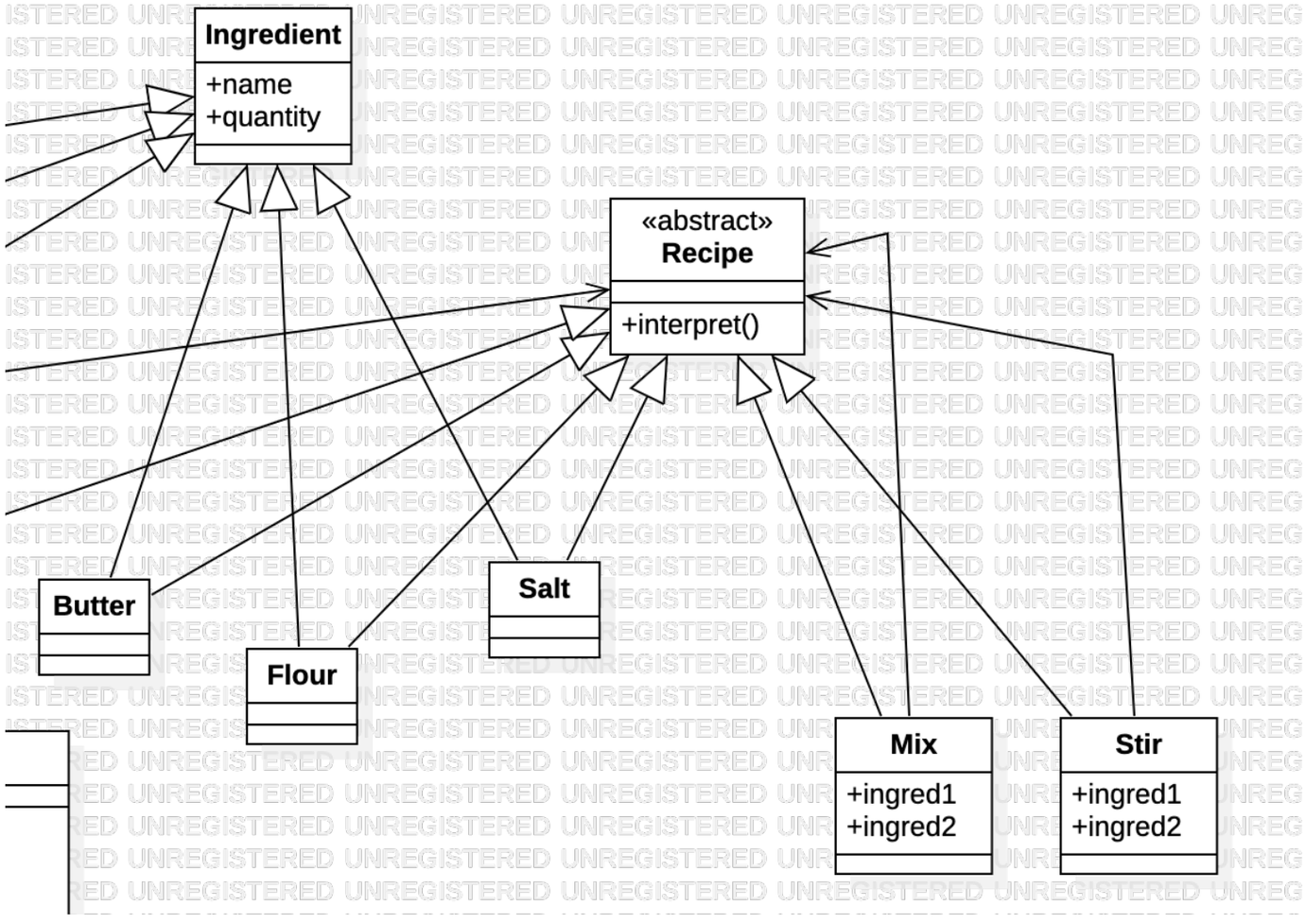
**Order**
+pizzaOrder(): int
+pizzaCook(recipe: Recipe)

izzaBuilder {
aComponent createPizza() {
is.buildDough()
is.buildSauce()
is.buildTopping()

ract buildDough()
ract buildSauce()
ract buildTopping()

UPPSALA
UNIVERSITET

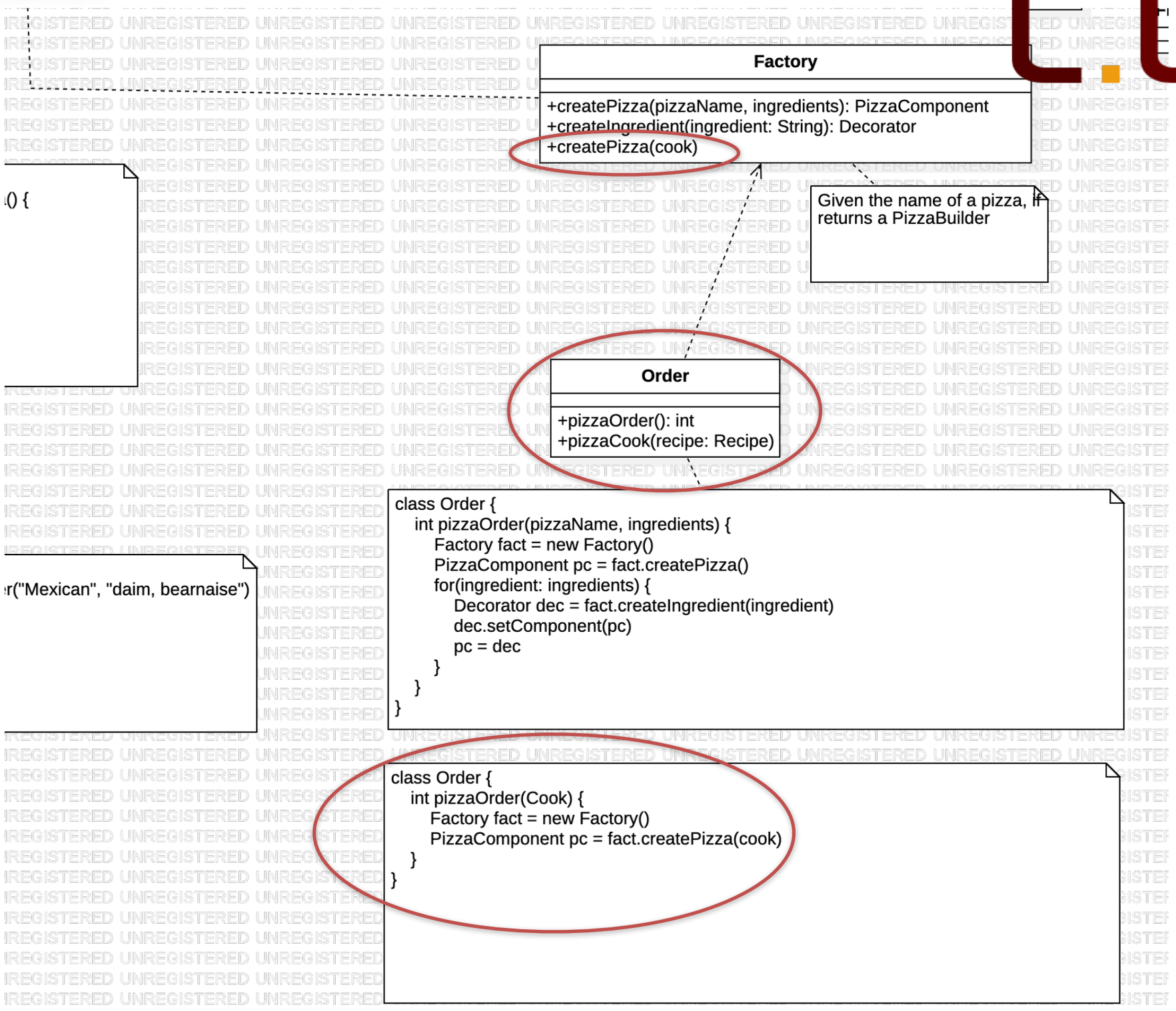
C.O.O.K




Recipe:

- made of ingredients
- represent language, encoded as ABS

UPPSALA
UNIVERSITET



UPPSALA
UNIVERSITET

C.O.O.K