Tuesday 9/30/14 4156

Factory pattern

```
Pizza orderPizza() {
    Pizza pizza = new Pizza();

    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}

    Pizza orderPizza(String type){ }
        Pizza pizza)

        if (type.equals("cheese"){
            pizza = new CheesePizza()}
        } else if (type.equals("greek"){
            pizza = new GreekPizza();
        } else if (type.equals("pepperoni"){
            pizza = new PepperoniPizza();
        }
```

need to change this code for every new (or removed) kind of pizza
— not closed for modification

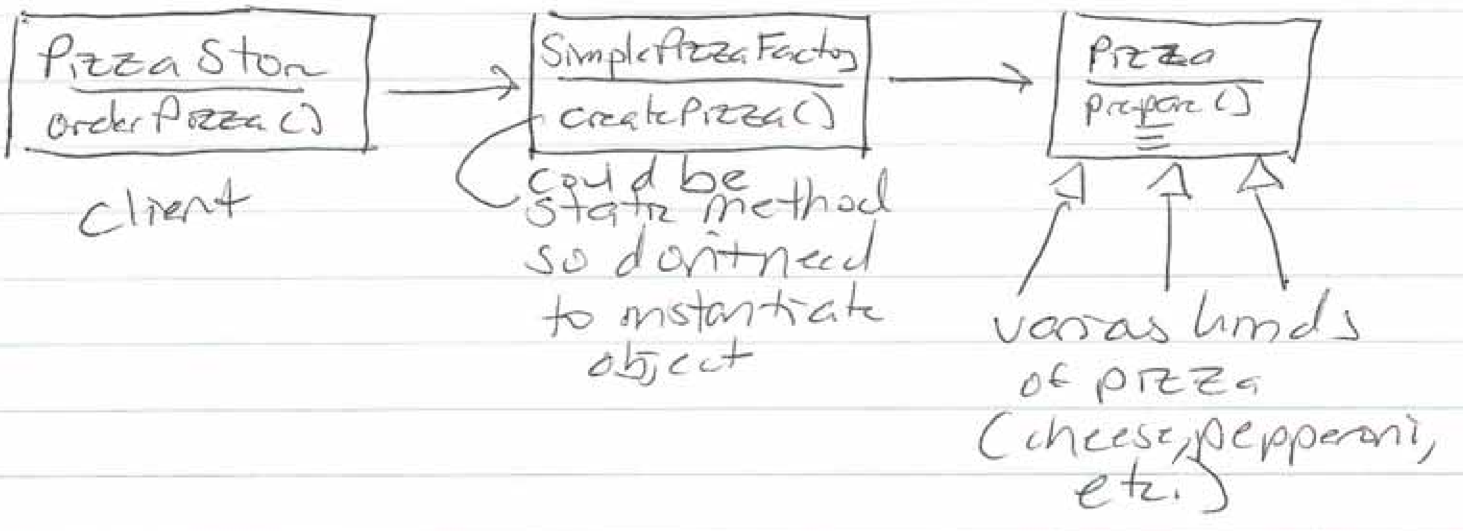
Tuesday 9/30/14 4156

we need to encapsulate object creation

Simple Pizza Factory

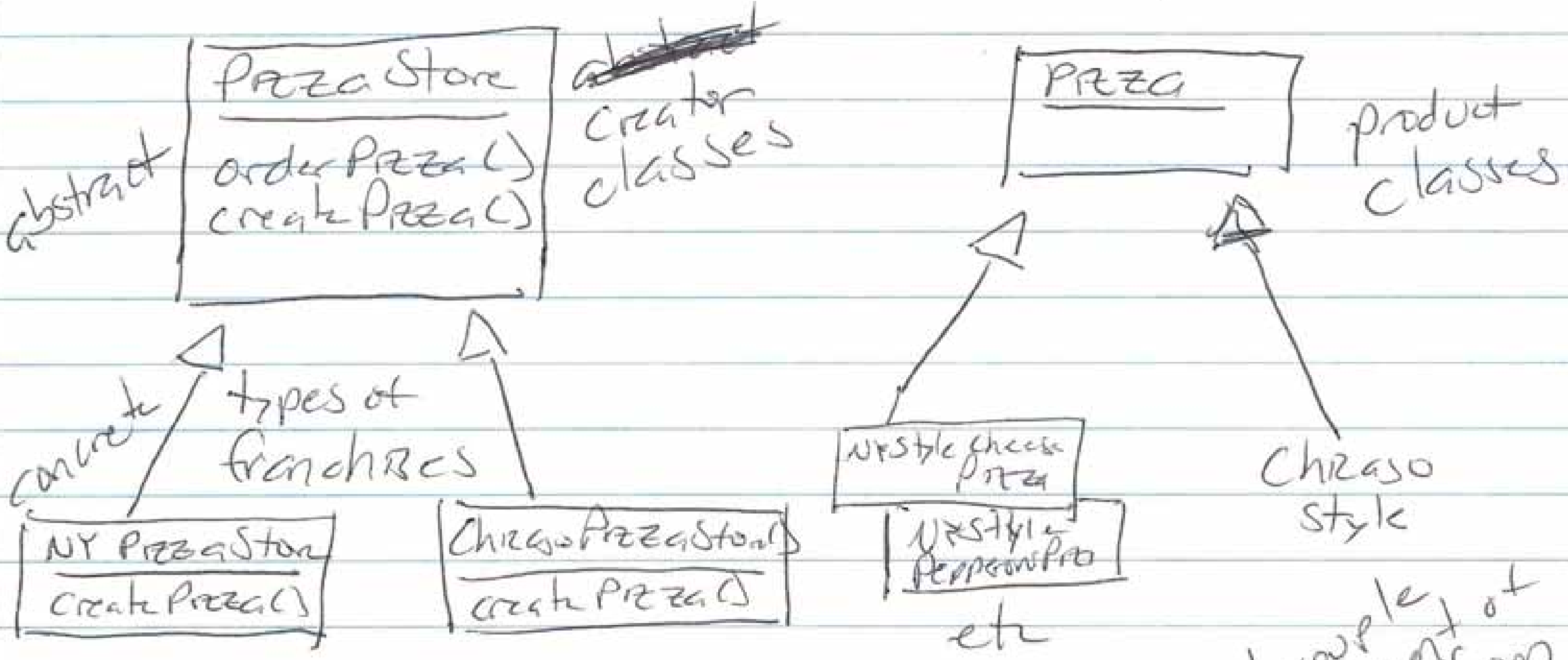
```
public class SimplePizzaFactory {
    public Pizza createPizza(String type) {
        Pizza pizza = null;

        if  ≡          ← taken from
                         orderPizza()
        else ≡

        return pizza;
    } }


public class PizzaStore {
    SimplePizzaFactory factory;

    public pizzaStore(SimplePizzaFactory) {
        this.factory = factory;
    }

    public Pizza orderPizza(type) {
        Pizza pizza;
        pizza = factory.createPizza(typed);
        ≡
    }
```

Tuesday 9/30/14 4156



PizzaStore
orderPizza()
client
SimplePizzaFactory
createPizza()
could be static method so don't need to instantiate object
Pizza
prepare()
various kinds of pizza (cheese, pepperoni, etc.)

this simple factory mechanism is NOT on the official list of design patterns - instead the factory method pattern



parallel class hierarchies

decouple implement of product from its use

factory method pattern defines an interface for creating an object, but lets subclasses decide which class to instantiate

Tuesday 9/30/14 4156

the term factory is broadly used whenever there's a separate class or method responsible for construction & which particular object is created depends on the subclass chosen at runtime

Abstract Factory is related concept for when we need a family of products - each ~~with~~ member of the family with its own set of subclasses
- dependent objects

book motivates by extending the pizza franchise idea to consider families of ingredients

e.g., cheese pizza
Chicago style - plum tomato sauce, mozzarella cheese, parmesan cheese, oregano spices, thick crust dough
NY style - marinara sauce, reggiano cheese, garlic spices, thin crust dough
each has sauce, cheese, spices but choose which specific ingredients based on pizza style

Tuesday 9/30/14

```
public interface
    Pizza Ingredient Factory {

      public Dough createDough();
      public Sauce createSauce();
      public Cheese createCheese();
        etc.
 }
```

```
public class NY Pizza IngredientFactory
    implements Pizza Ingredient Factory {

      implements each create method
      using NY style ingredients
```

another subclass for Chicago style,
  California style, etc.

```
public class NY PizzaStore extends
      PizzaStore {
   protected Pizza createPizza (String item) {
      Pizza  pizza = null;
      Pizza IngredientFactory =
           new NYPizzaIngredient Factory();
      if (item.equals = "cheese")} {
          pizza = new Cheese Pizza (IngredientFactory);
    etc.
```

Tuesday 9/30/14 4156

Abstract Factory gives interface for creating family of dependent products that need to "match"
- decouples client code from actual factory

implement variety of factories for different contexts - client can be composed w/ actual factory at runtime

Abstract Factory uses object composition & addresses set of dependent objects
Factory Method uses inheritance & addresses choice of one object among subclasses

Tuesday 9/30/14 4156

singleton pattern - ensures a class has only one instance, & provides a global point of access to it

examples - thread pool, cache, log, registry, device driver

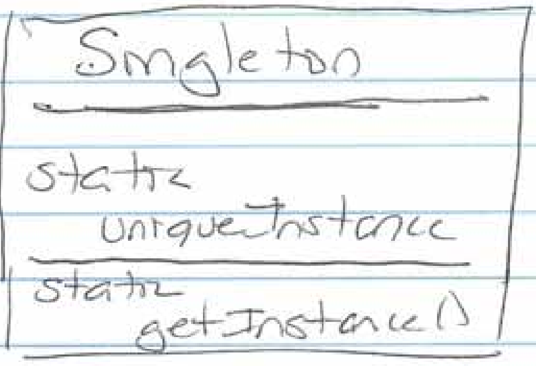
instantiating more than one would cause errors - need convention for ensuring no more than one is ever instantiated (without necessarily instantiating that one as part of system startup)

trick - no public constructor
declared private

static method getInstance()
check if instance already exists
if yes, return it
if no, create & return it (lazy)

prevents any other class from creating a new instance on its own
class manages single instance of self

Tuesday 9/30/14 4156



access via

Singleton.getInstance()

```
public class Singleton {
   private static Singleton uniqueInstance;
     =

   private Singleton() { }                why??

   public static synchronized
         SingletongetInstance() {
      if (uniqueInstance == null) {
         uniqueInstance = new Singleton();
      }
      return uniqueInstance;
   }
     =
}
```

synchronization is expensive, if getInstance() is called often may want to initialize eagerly - guaranteed threadsafe if static initializer on class load