Encapsulating object creation

So now we know we'd be better off moving the object creation out of the orderPizza() method. But how? Well, what we're going to do is take the creation code and move it out into another object that is only going to be concerned with creating pizzas.

```
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();
}
```

```
Pizza orderPizza(String type) {
        Pizza pizza;

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

First we pull the object creation code out of the orderPizza() method.

What's going to go here?

Then we place that code in an object that is only going to worry about how to create pizzas. If any other object needs a pizza created, this is the object to come to.

We've got a name for this new object: we call it a Factory.

Factories handle the details of object creation. Once we have a SimplePizzaFactory, our orderPizza() method becomes a client of that object. Anytime it needs a pizza, it asks the pizza factory to make one. Gone are the days when the orderPizza() method needs to know about Greek versus Clam pizzas. Now the orderPizza() method just cares that it gets a pizza that implements the Pizza interface so that it can call prepare(), bake(), cut(), and box().

We've still got a few details to fill in here; for instance, what does the orderPizza() method replace its creation code with? Let's implement a simple factory for the pizza store and find out...