Simple Factory Pattern

- Simple factory generates an instance of an object/service for client without exposing any implementation to the client.
- Refers to the newly created object through a common interface
- Formally a factory is a function or method that returns objects of a varying prototype or class from some method call, which is assumed to be "new".
- One family of Objects.

```
class Main {
  public static void main(String[] args) {
    PizzaGetter pg = new PizzaGetter();
    Pizza p = pg.getPizza();
    System.out.println(p+" has: "+p.toppings);

  }
}

class PizzaGetter{
    Pizza getPizza(){
    return new CheesePizza();
 }
}

class Pizza{
  String toppings;
}

class CheesePizza extends Pizza{
  CheesePizza(){
    toppings = "cheese";
  }
  void melt(){
    System.out.println("sizzle");
  }
}

// result:
// CheesePizza@001 has: cheese
```

Possible issue:

```
p.melt(); // will NOT work
```

`p` is a `Pizza` type with referenece to `CheesePizza` type object, and `melt()` is

inside CheesePizza (have a Pizza interface to a CheesePizza object).

the type of the reference determines the questions you can ask

---

Factory Method

- Defines an interface for creating objects, but let subclasses to decide which class to instantiate
- Refers the newly created object through a common interface.

  The Factory Method Pattern defines an interface for creating an object, but lets subclasses decide which class to instantiate.

  Factory Method lets a class defer instantiation to subclasses.

```
class Main {
  public static void main(String[] args) {
    PizzaGetter pg = new DeluxePizzaGetter();
    Pizza p = pg.getPizza();
    System.out.println(p);
 }
}

// Factory Method Pattern
abstract class PizzaGetter{
  Pizza getPizza(){
    return subclassMakePizza();//abs below
  }
  // factory method
  abstract Pizza subclassMakePizza(); // <---- factory method
}

class DeluxePizzaGetter extends PizzaGetter{
  Pizza subclassMakePizza(){          // <---- factory method
    return new DeluxeCheese();
  }
}

class Pizza{
  String toppings;
}

class CheesePizza extends Pizza{
```

```
  CheesePizza(){
    toppings = "cheese";
  }
}

class DeluxeCheese extends CheesePizza{
  DeluxeCheese(){
    toppings = " lots of cheese!";
  }
}

//  returns:
//  DeluxeCheese@001
```

Simple Factory VS Factory Method

Factory Method subclasses look a lot like Simple Factory, however Simple Factory is used once, while with Factory Method you are creating a framework that let's the subclasses decide which implementation will be used.

For example, the subclassMakePizza() method in the Factory Method provides a general framework for creating pizzas that relies on a factory method to actually create the concrete classes that go into making a pizza. By subclassing the DeluxaPizzaGetter class, you decide what concrete products go into making the pizza that subclassMakePizza() returns. Compare that with Simple Factory, which gives you a way to encapsulate object creation, but doesn't give you the flexibility of the Factory Method because there is no way to vary the products you're creating.

Abstract Factory

- Abstract Factory offers the interface for creating a family of related objects, without explicitly specifying their concrete classes.
- Can create a family of related objects that have different parent class or interface.
- A factory of factories.
- An Abstract Factory gives us an interface for creating a family of products. By writing code that uses this interface, we decouple our code from the actual factory that creates the products. That allows us to implement a variety of factories that produce products meant for different contexts –

such as different regions, different operating systems, or different look and feels.
- provides an abstract type for creating a family of products. Subclasses of this type define how those products are produced. To use the factory, you instantiate one and pass it into some code that is written against the abstract type.
- group together a set of related products

```
abstract class Pizza {
  PizzaIngredientFactory ingredientFactory;
  String name;

  Dough dough;
  Sauce sauce;
  Cheese cheese;

  // prepare is abstract to collect ingredients from factory
  abstract void prepare();


  void setName(String name) {
    this.name = name;
  }

  String getName() {
    return name;
  }

}

public class CheesePizza extends Pizza {

  PizzaIngredientFactory ingredientFactory;

  // each CheesePizza gets passed a factory
  public CheesePizza(PizzaIngredientFactory ingredientFactory) {
    this.ingredientFactory = ingredientFactory;
  }

  void prepare() {
    System.out.println("Preparing " + name);
    dough = ingredientFactory.createDough();
    sauce = ingredientFactory.createSauce();
    cheese  = ingredientFactory.createCheese();
  }
```

```
}

class NYPizzaStore extends PizzaStore {

  protected Pizza createPizza(String item) {
    Pizza pizza = null;
    ingredientFactory = new NYPizzaIngredientFactory();

    if (item.equals("cheese")) {

      pizza = new CheesePizza(ingredientFactory);
      pizza.setName("New York Style Cheese Pizza");

    }
    return pizza;
  }
}

class PizzaTestDrive {

  public static void main(String[] args) {

    PizzaStore nyStore = new NYPizzaStore();

    Pizza pizza1 = nyStore.orderPizza("cheese");
    System.out.println( pizza1 );

  }
}
```

1) create instance of NYPizzaStore
2) instance calls orderPizza()
3) createPizza() is then called, and the ingredientFactory is instantiated and passed to the pizza
4) prepare() is called which calls the factory to cerate the ingredient
5) orderPizza()

Factory Method VS Abstract Factory

- Factory Method is a single method, and an Abstract Factory is an object.
- Factory Method uses classes to create through inheritence, while Abstract Factory uses objects composition.

- To create objects using Factory Method, you need to extend a class and override a factory method.
- Factory Method is used to create one product only but Abstract Factory is about creating families of related or dependent products.
- Abstract Factory can be implemented by multiple Factory Methods.

Factories Summary

- All factories encapsulate object creation.
- Simple Factory (not design pattern) is a simple way to decouple your clients from concrete classes
- Factory Method relies on inheritance: object creation is delegated to subclasses which implement the factory method to create objects.
- Abstract Factory relies on object composition: object creation is implemented in methods exposed in the factory interface.
- All factory patterns promote loose coupling by reducing the dependency of your application on concrete classes.
- The intent of Factory Method is to allow a class to defer instantiation to its subclasses.
- The intent of Abstract Factory is to create families of related objects without having to depend on their concrete classes.
- The Dependency Inversion Principle guides us to avoid dependencies on concrete types and to strive for abstractions.
- Factories are a powerful technique for coding to abstractions, not concrete classes