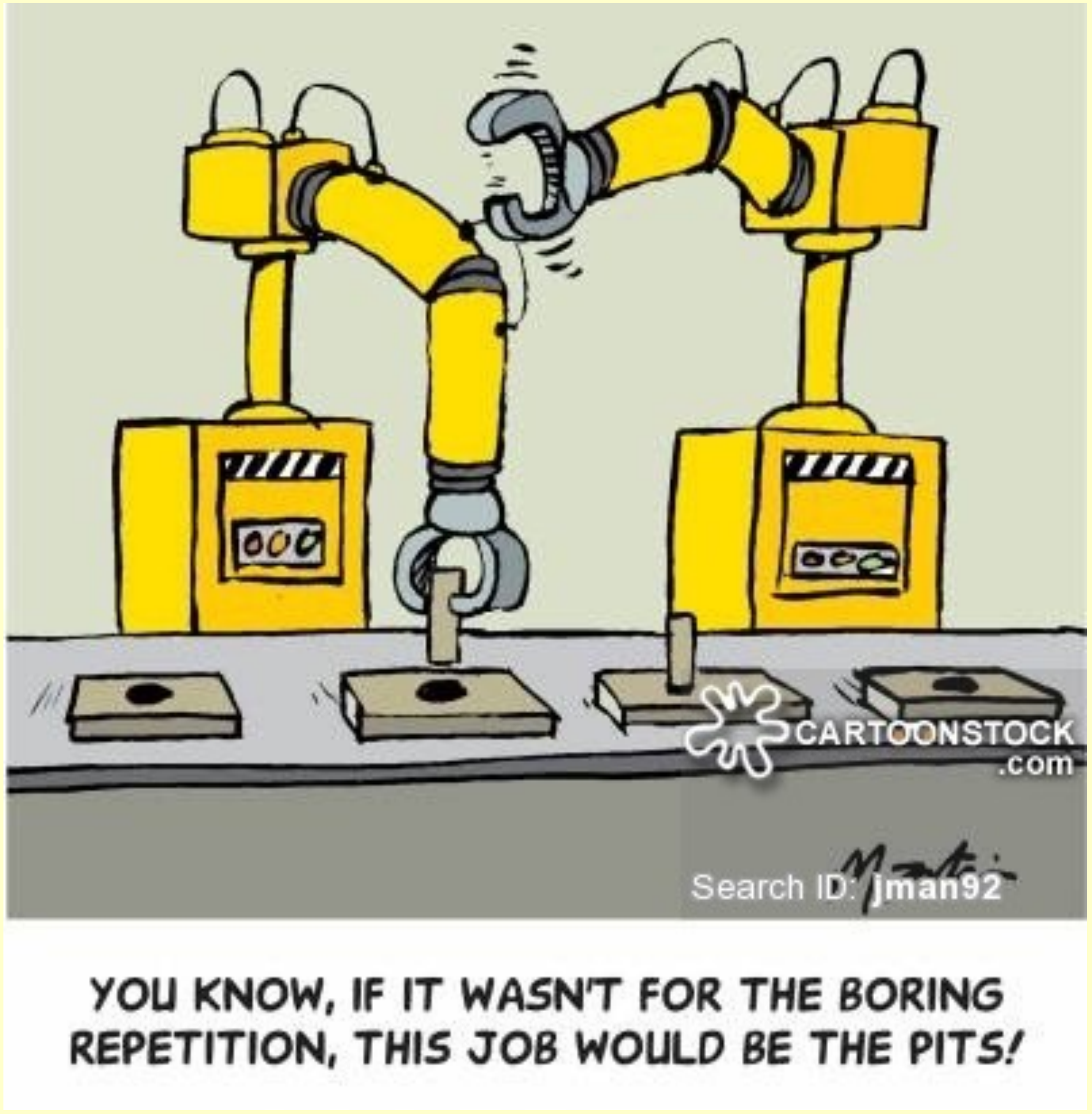
CSC 7203 : Advanced Object Oriented Development

J Paul Gibson, D311

paul.gibson@telecom-sudparis.eu
~gibson/Teaching/CSC7203/

Factory Pattern

…/~gibson/Teaching/CSC7203/CSC7203-AdvancedOO-L3-Factory.pdf


CARTOONSTOCK
.com
Search ID: jman92
YOU KNOW, IF IT WASN'T FOR THE BORING REPETITION, THIS JOB WOULD BE THE PITS!

Factory Pattern

See - http://sourcemaking.com/design_patterns/factory_method

•Intent

•Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
•Defining a "virtual" constructor.

•Problem

•A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.

NOTE: The implementation of Factory Method discussed in the largely overlaps with that of Abstract Factory.

Factory Pattern

See - http://sourcemaking.com/design_patterns/factory_method

Relation to other patterns

Abstract Factory classes are often implemented with Factory Methods, but they can be implemented using Prototype.

Factory Methods are usually called within Template Methods.

Factory Method: creation through inheritance. Prototype: creation through delegation.

Often, designs start out using Factory Method (less complicated, more customizable, subclasses proliferate) and evolve toward Abstract Factory, Prototype, or Builder (more flexible, more complex) as the designer discovers where more flexibility is needed.

Prototype doesn't require subclassing, but it does require an Initialize operation. Factory Method requires subclassing, but doesn't require Initialize.

The advantage of a Factory Method is that it can return the same instance multiple times, or can return a subclass rather than an object of that exact type.

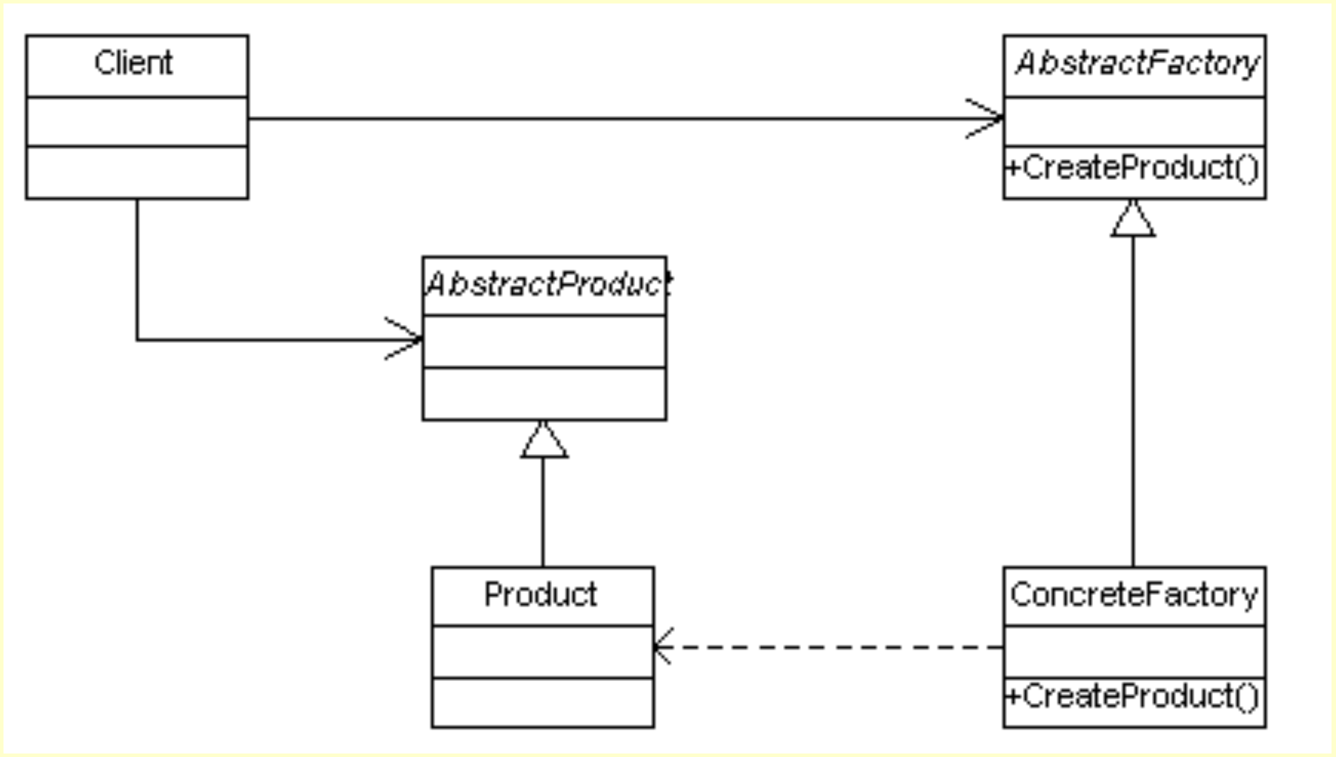
Factories: Additional Motivation

Factories are key to ***Software Product Lines***

See:

Software Factories Assembling Applications with Patterns, Models, Frameworks and Tools, Greenfield and Short, OOPSLA, 2003.

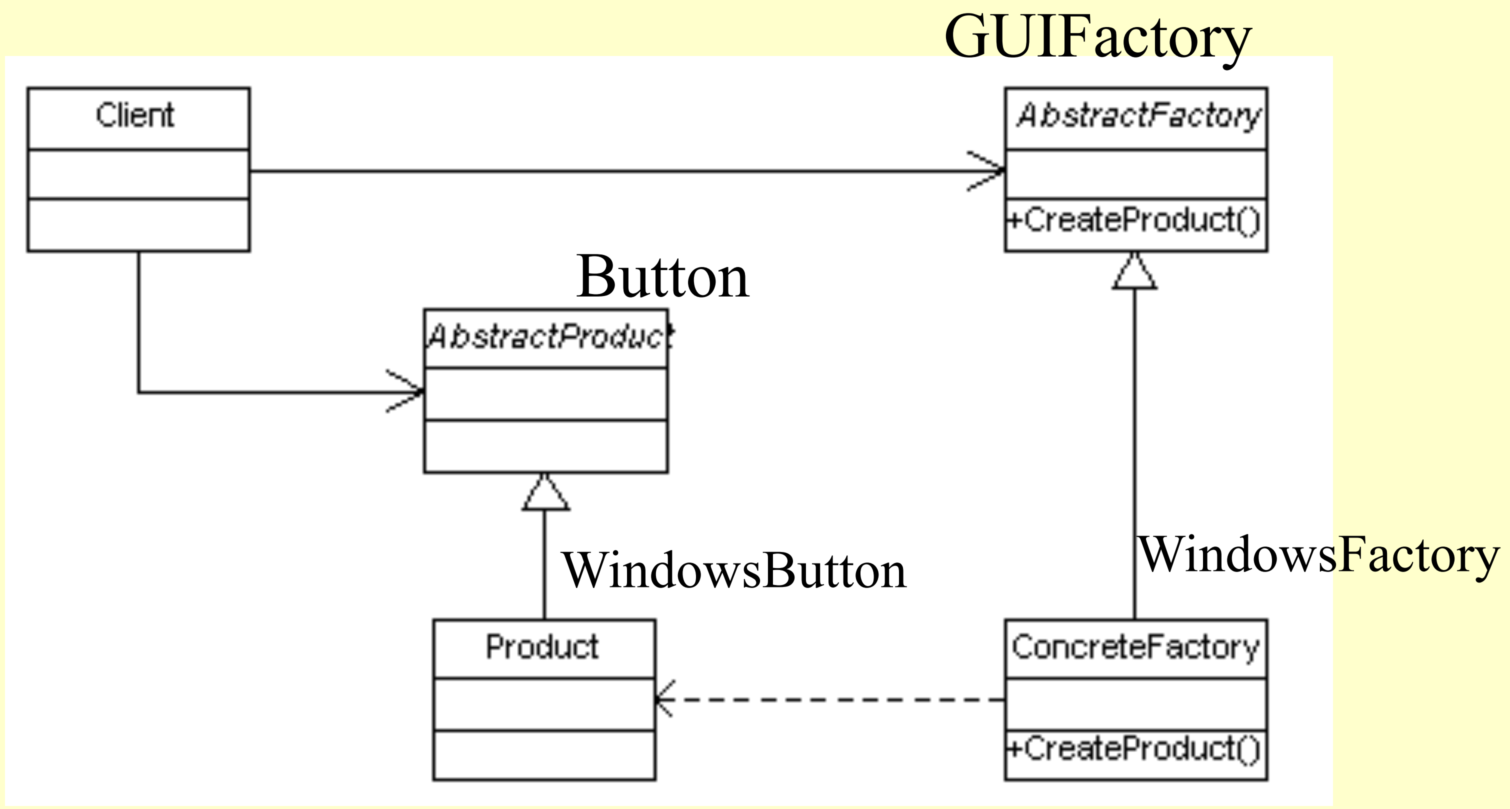
Patron: Factory (Fabrique): UML (generic)



Can be generalised to:

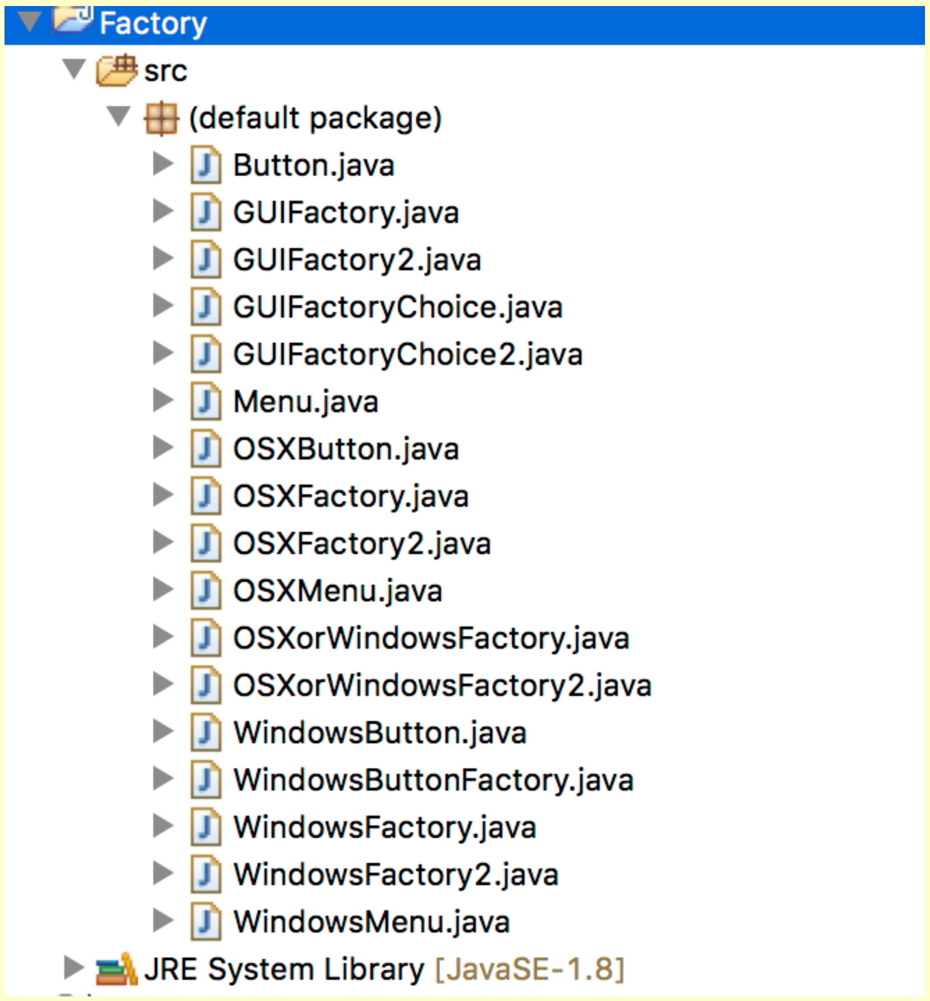
- multiple products (by subclassing)
- multiple clients (by association)

Factory UML: concrete example – WindowsButtonFactory



You can find the files in the Patterns folder in the **p_factory** package

Factory package



```
▼ Factory
  ▼ src
    ▼ (default package)
      ▶ Button.java
      ▶ GUIFactory.java
      ▶ GUIFactory2.java
      ▶ GUIFactoryChoice.java
      ▶ GUIFactoryChoice2.java
      ▶ Menu.java
      ▶ OSXButton.java
      ▶ OSXFactory.java
      ▶ OSXFactory2.java
      ▶ OSXMenu.java
      ▶ OSXorWindowsFactory.java
      ▶ OSXorWindowsFactory2.java
      ▶ WindowsButton.java
      ▶ WindowsButtonFactory.java
      ▶ WindowsFactory.java
      ▶ WindowsFactory2.java
      ▶ WindowsMenu.java
  ▶ JRE System Library [JavaSE-1.8]
```

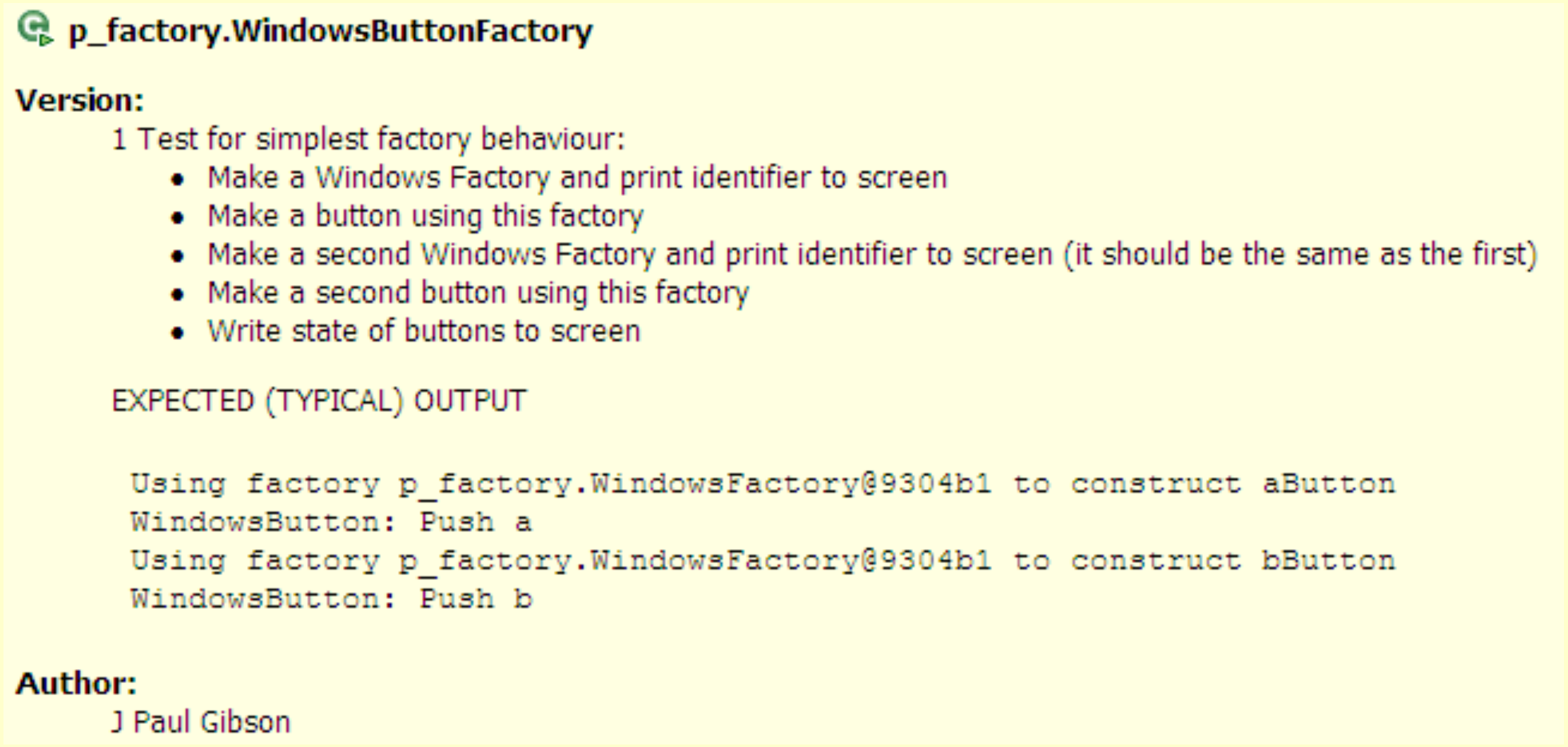
NOTE: I have not added any package structure – I suggest you restructure the Factory into separate dossiers:

- by OS or
- by components

Also, you may wish to have a package for the abstract classes, and for tests

Download from the web site - Code/Factory.zip

Factory - Windows GUI in Java



p_factory.WindowsButtonFactory

Version:

1 Test for simplest factory behaviour:

- Make a Windows Factory and print identifier to screen
- Make a button using this factory
- Make a second Windows Factory and print identifier to screen (it should be the same as the first)
- Make a second button using this factory
- Write state of buttons to screen

EXPECTED (TYPICAL) OUTPUT

```
 Using factory p_factory.WindowsFactory@9304b1 to construct aButton
 WindowsButton: Push a
 Using factory p_factory.WindowsFactory@9304b1 to construct bButton
 WindowsButton: Push b
```

Author:

J Paul Gibson

NOTE: Factories are often defined as Singletons.

Factory - Windows GUI in Java

```
public class WindowsButtonFactory {

    public static void main(String[] args){

        GUIFactory aFactory = GUIFactory.getFactory();
        System.out.println("Using factory "+ aFactory+" to construct aButton");
        Button aButton = aFactory.createButton();
        aButton.setCaption("Push a");
        aButton.paint();

        GUIFactory bFactory = GUIFactory.getFactory();
        System.out.println("Using factory "+ bFactory+" to construct bButton");
        Button bButton = bFactory.createButton();
        bButton.setCaption("Push b");
        bButton.paint();
    }

}
```

TO DO: Compile and execute to test for expected output

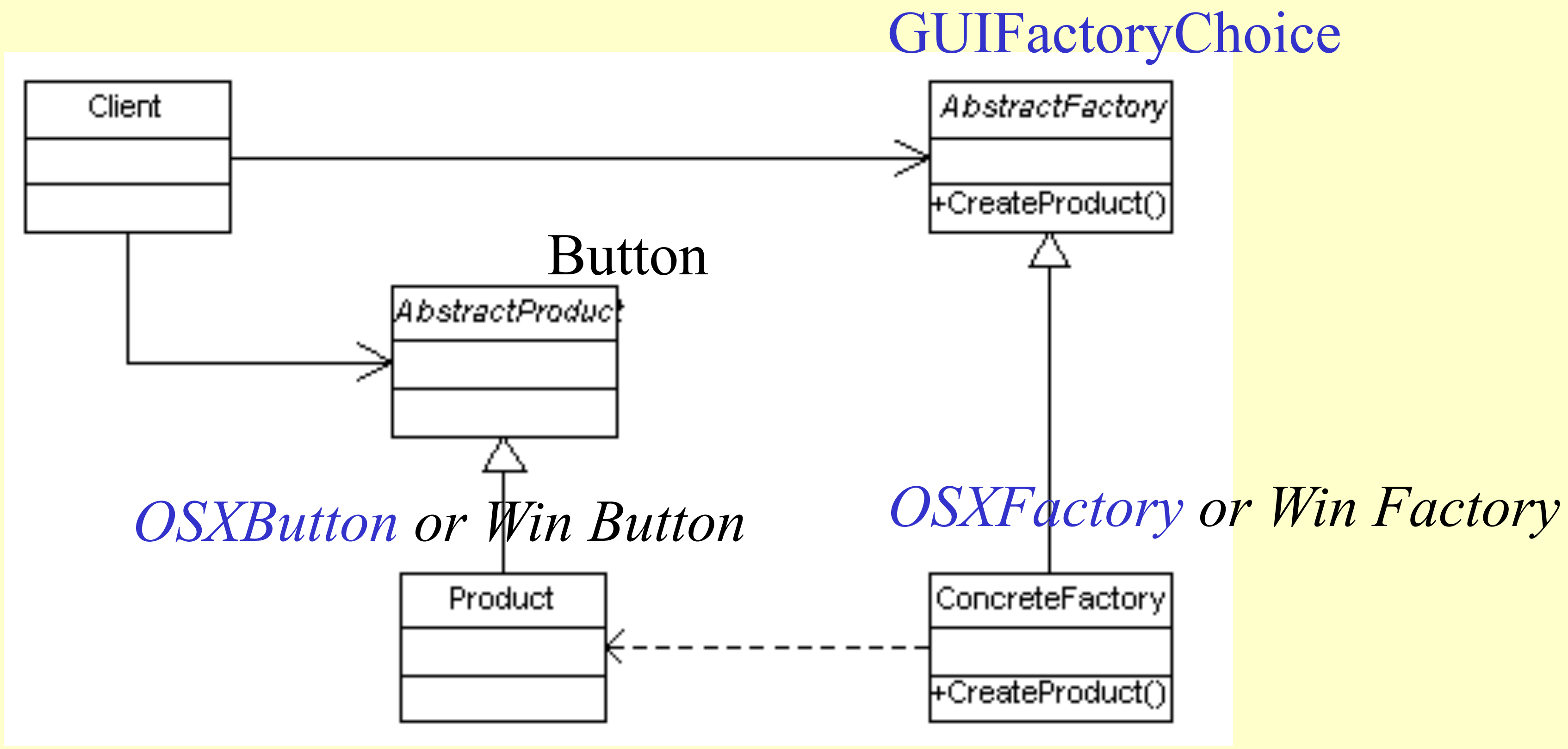
Factory - Windows GUI in Java

```java
abstract class Button
{
    private String caption;
    public abstract void paint();
    public String getCaption() {return caption;}

    public void setCaption(String caption){
        this.caption = caption;
    }
}

public class WindowsButton extends Button
{
    public void paint(){
        System.out.println("WindowsButton: "+ getCaption());
    }
}
```

Factory - Windows GUI in Java

```
abstract class GUIFactory{
    public static GUIFactory getFactory(){
          return WindowsFactory.getInstance();
    }
public abstract Button createButton();
}
class WindowsFactory extends GUIFactory{
private static WindowsFactory factory = new WindowsFactory();

    public static WindowsFactory getInstance () {return factory;};
    public Button createButton(){
        return(new WindowsButton());
    }
}
```

Factory « *UML* »: OSXorWindowsFactory



TO DO: Write code for OSXButton and OSXFactory

Factory OSX and Win GUI Buttons in Java

```
abstract class GUIFactoryChoice{
    public enum OS_Type {Win, OSX}

    protected static OS_Type readFromConfigFile(String param){
     if (Math.random() > 0.5) return OS_Type.Win;
        else return OS_Type.OSX;
    }

 public static GUIFactory getFactory(){
        OS_Type sys = readFromConfigFile("OS_TYPE");
        switch (sys) {
            case Win:
                return WindowsFactory.getInstance();
            case OSX:
                return  OSXFactory.getInstance();
        }
 throw new IllegalArgumentException("The OS type " + sys + " is not recognized.");
     }

    public abstract Button createButton();
}
```

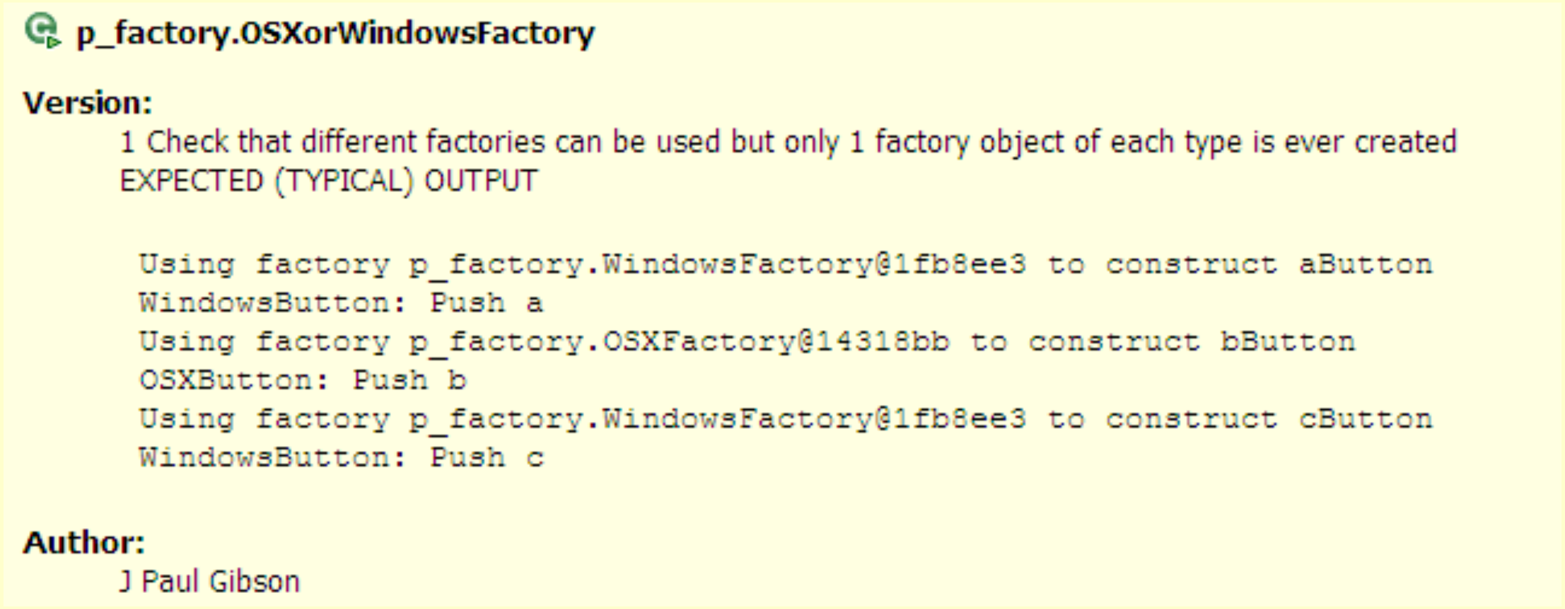
Use this more complex factory in your test code

Factory OSX and Win GUI Buttons in Java

```java
public class OSXorWindowsFactory {

 public static void main(String[] args){

        GUIFactory aFactory = GUIFactoryChoice.getFactory();
        System.out.println("Using factory "+ aFactory+" to construct aButton");
        Button aButton = aFactory.createButton();
        aButton.setCaption("Push a");
        aButton.paint();

        GUIFactory bFactory = GUIFactoryChoice.getFactory();
        System.out.println("Using factory "+ bFactory+" to construct bButton");
        Button bButton = bFactory.createButton();
        bButton.setCaption("Push b");
        bButton.paint();

        GUIFactory cFactory = GUIFactoryChoice.getFactory();
        System.out.println("Using factory "+ cFactory+" to construct cButton");
        Button cButton = cFactory.createButton();
        cButton.setCaption("Push c");
        cButton.paint();
    }

}
```

TO DO: Compile and execute this code

Factory OSX and Win GUI Buttons in Java



p_factory.OSXorWindowsFactory

Version:

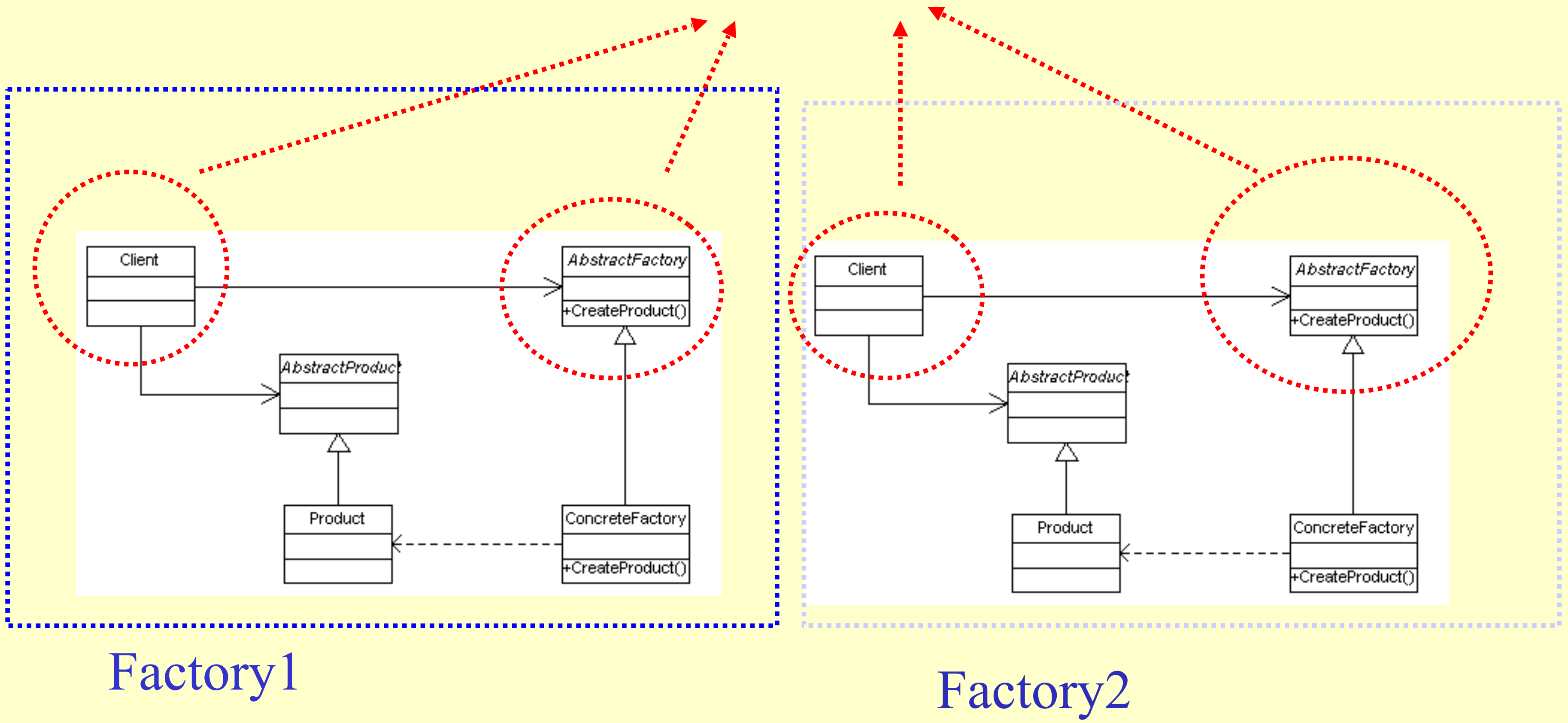
1 Check that different factories can be used but only 1 factory object of each type is ever created
EXPECTED (TYPICAL) OUTPUT

```
Using factory p_factory.WindowsFactory@1fb8ee3 to construct aButton
WindowsButton: Push a
Using factory p_factory.OSXFactory@14318bb to construct bButton
OSXButton: Push b
Using factory p_factory.WindowsFactory@1fb8ee3 to construct cButton
WindowsButton: Push c
```

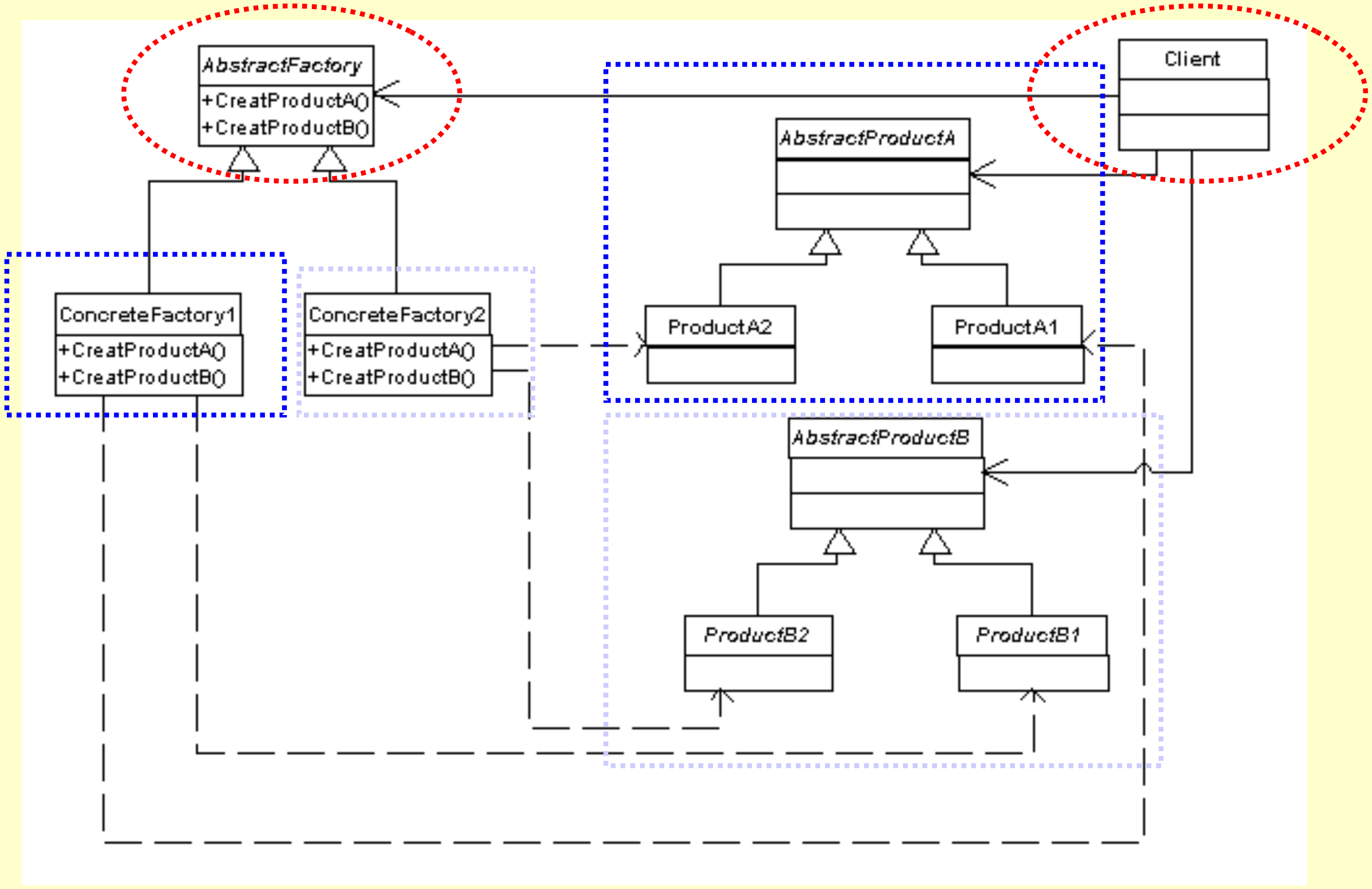
Author:

J Paul Gibson

Abstract Factory

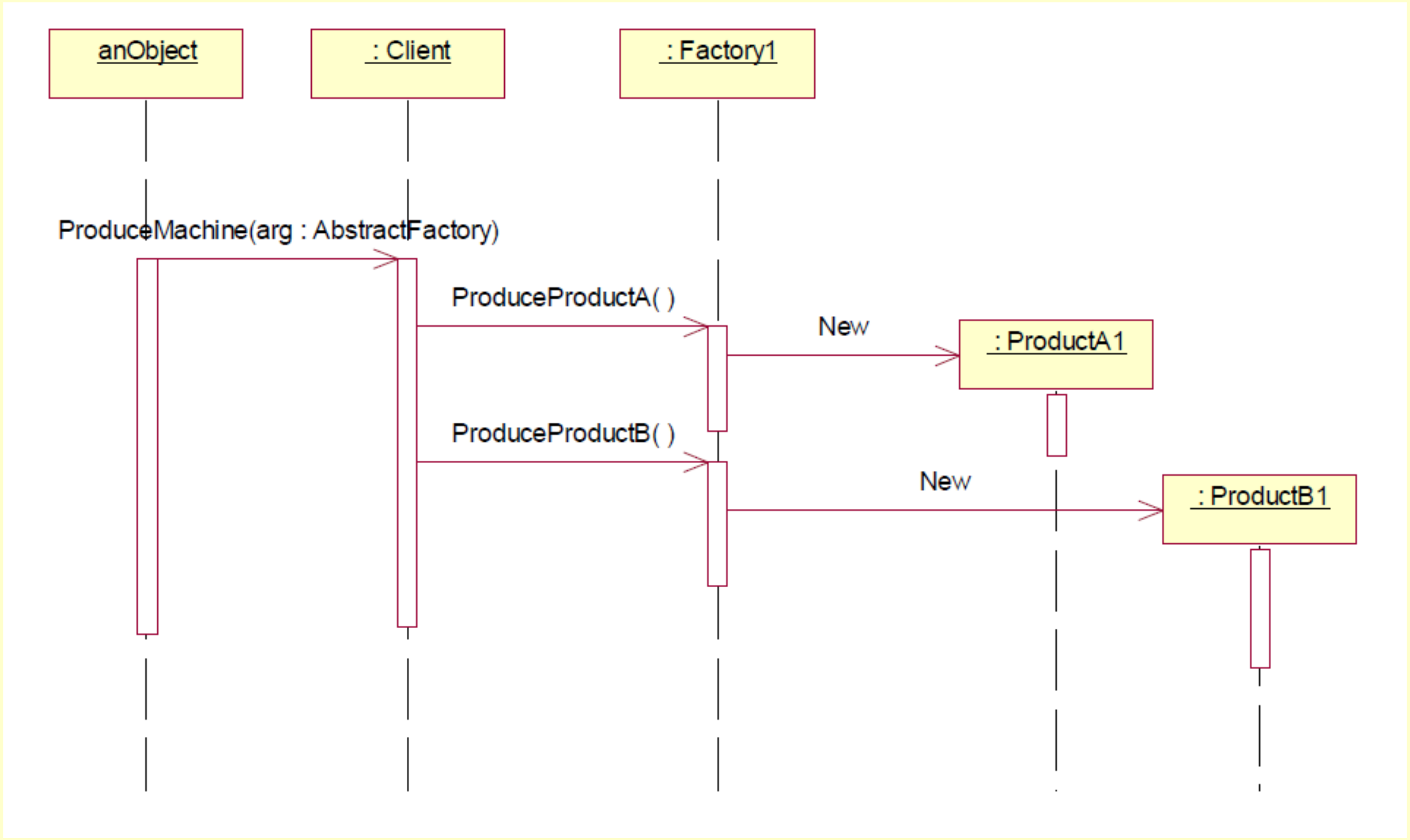


Abstract Factory: UML class diagram (2 products 2 factory types)



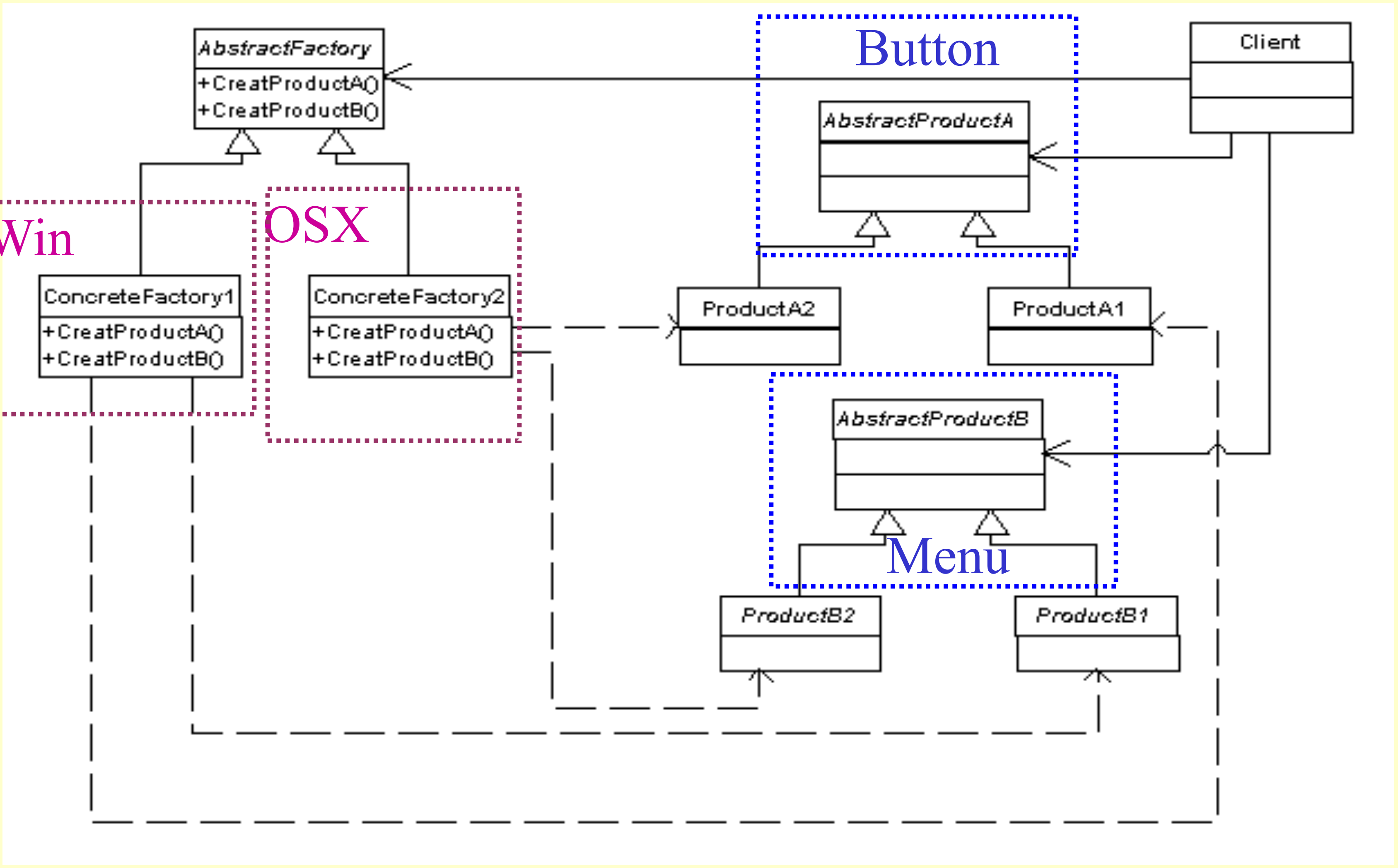
Can be generalised to multiple factories with multiple products

Abstract Factory: UML (sequence diagram)

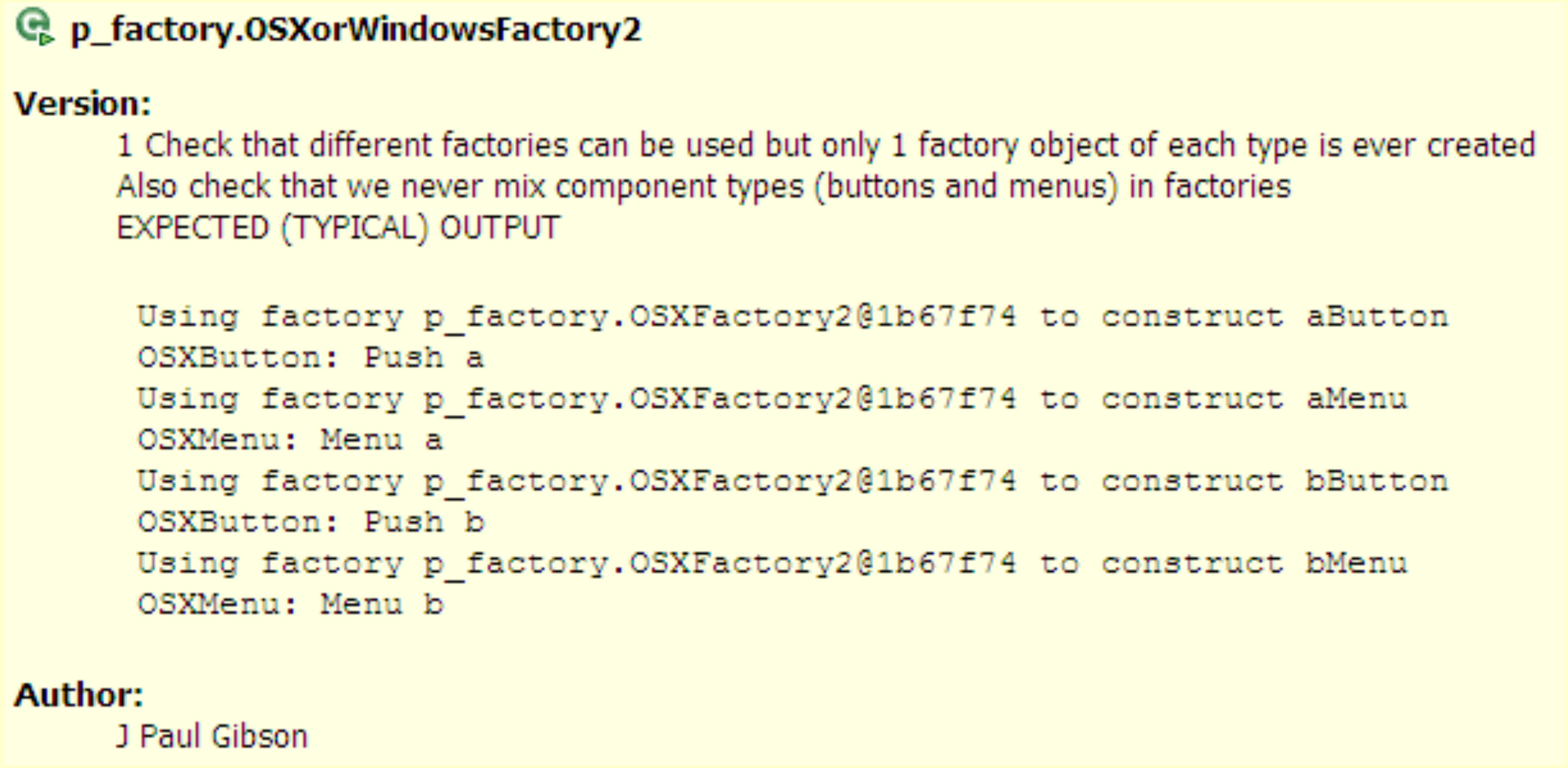


Can be generalised to multiple factories with multiple products

Abstract Factory: UML - Buttons and Menus for Win and OSX



Abstract Factory – GUIFactoryChoice2


p_factory.OSXorWindowsFactory2

Version:

1 Check that different factories can be used but only 1 factory object of each type is ever created
Also check that we never mix component types (buttons and menus) in factories
EXPECTED (TYPICAL) OUTPUT

```
Using factory p_factory.OSXFactory2@1b67f74 to construct aButton
OSXButton: Push a
Using factory p_factory.OSXFactory2@1b67f74 to construct aMenu
OSXMenu: Menu a
Using factory p_factory.OSXFactory2@1b67f74 to construct bButton
OSXButton: Push b
Using factory p_factory.OSXFactory2@1b67f74 to construct bMenu
OSXMenu: Menu b
```

Author:

J Paul Gibson

TP - TO DO: Compile and execute this code in order to test it against expected behaviour

Abstract Factory – OSXorWindowsFactory2

```
public class OSXorWindowsFactory2 {
public static void main(String[] args){
        GUIFactory2 aFactory = GUIFactoryChoice2.getFactory();
        System.out.println("Using factory "+ aFactory+" to construct aButton");
        Button aButton = aFactory.createButton();
        aButton.setCaption("Push a");
        aButton.paint();
        System.out.println("Using factory "+ aFactory+" to construct aMenu");
        Menu aMenu = aFactory.createMenu();
        aMenu.setCaption("Menu a");
        aMenu.display();
        GUIFactory2 bFactory = GUIFactoryChoice2.getFactory();
        System.out.println("Using factory "+ bFactory+" to construct bButton");
        Button bButton = bFactory.createButton();
        bButton.setCaption("Push b");
        bButton.paint();
        System.out.println("Using factory "+ bFactory+" to construct bMenu");
        Menu bMenu = bFactory.createMenu();
        bMenu.setCaption("Menu b");
        bMenu.display();
    }
 }
```

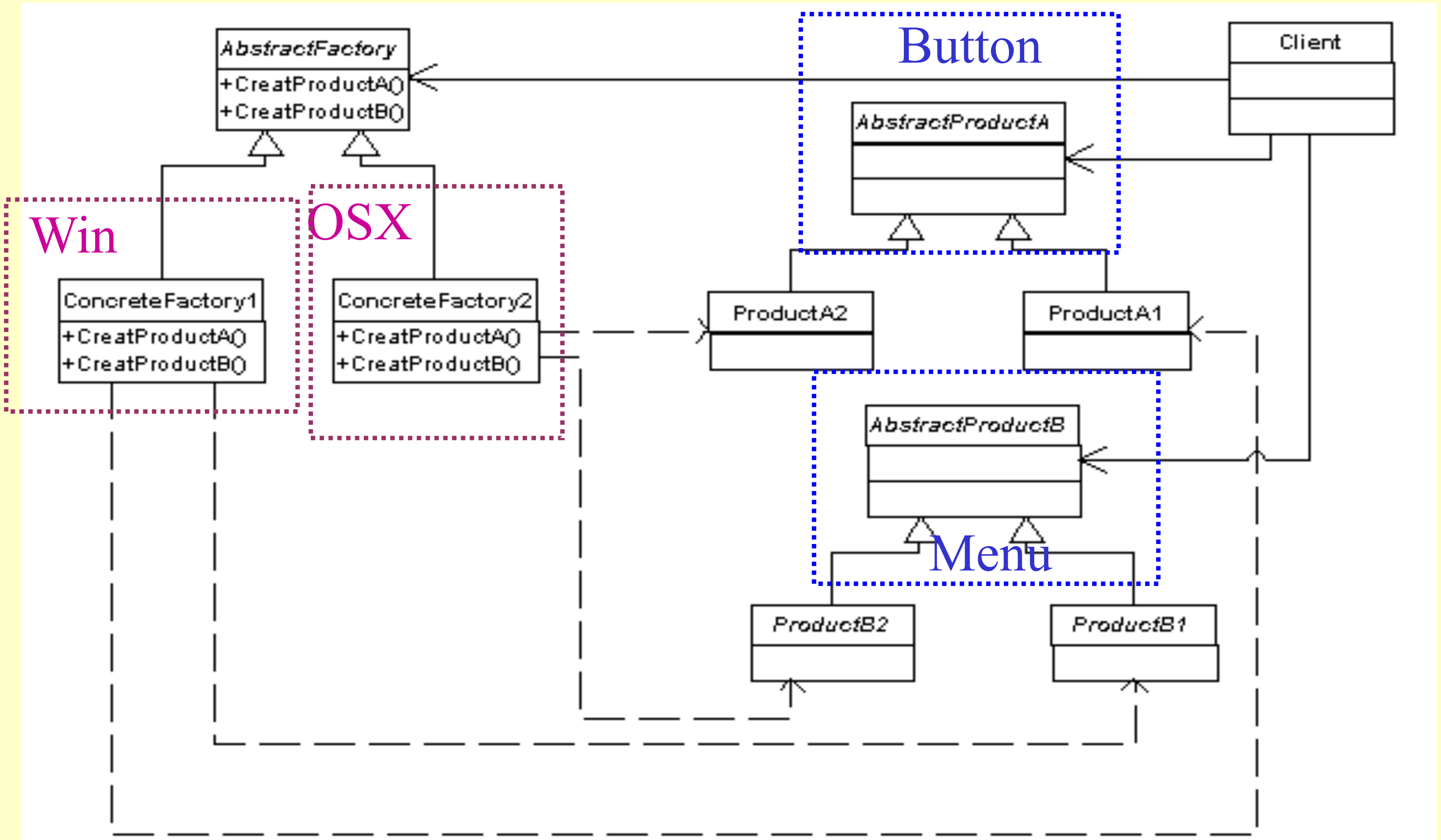
Abstract Factory – OSXorWindowsFactory2

Note that we had to extend the behaviour of classes in order to include buttons and menus (but we kept to the same design pattern):

```
public abstract class GUIFactory2 extends GUIFactory{
 public abstract Menu createMenu();
}


class WindowsFactory2 extends GUIFactory2 …


class OSXFactory2 extends GUIFactory2  …


class GUIFactoryChoice2 extends GUIFactoryChoice …
```

TO DO: Look at code and try to understand how it works

Problem – add an **OS (linux)** and a **Component (slider)**



Construct a linux *product* with button and slider components: test the behaviour of your product (code)

TO DO: Test the new factory and new factory component