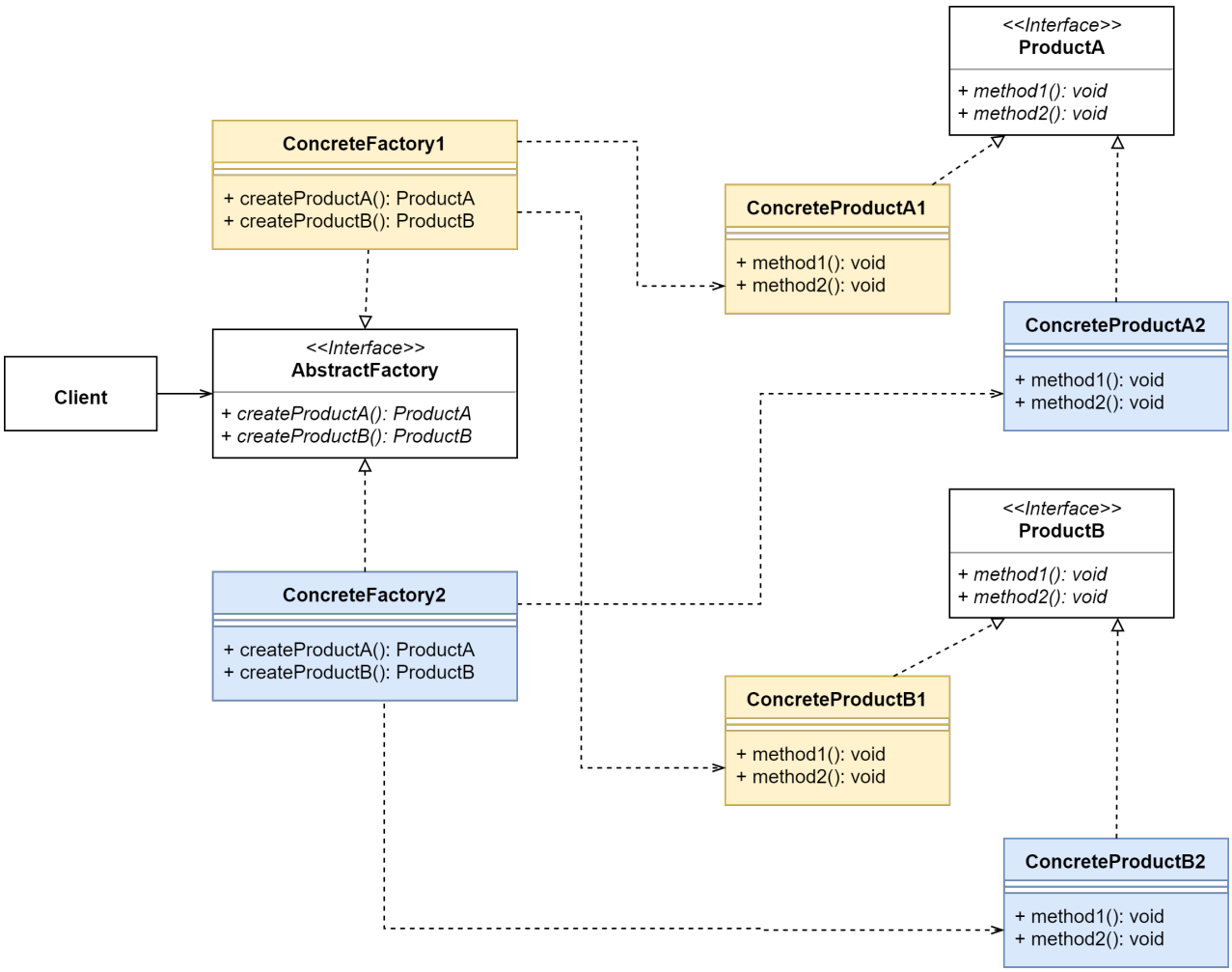
Abstract Factory

> Designed by Zhu Yueming

Abstract Factory

The abstract factory design pattern can provide a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes. Which means, the products created by only one concrete factory can not from different themes.

1. Class Diagram



Task 1

In `abstractFactory` package, please modify your code by using abstract factory design pattern in `dao` layer, and make sure that it can match the client.

Hints: You need to created two concrete classes including `MysqlDaoFactory` and `SqlServerDaoFactory` to implements the interface `DaoFactory`

Refactoring Abstract Factory by Singleton

The Singleton Pattern ensures a class has only one instance and provides a global point of access to that instance.

1. Class Diagram

2. Sample code

```
public class Singleton {
  private static Singleton instance = null;

  private Singleton() {}

  public static synchronized Singleton getInstance() {
    if (instance == null) {
      instance = new Singleton();
    }
    return instance;
  }
}
```

or

```
public class Singleton {
  private static Singleton instance = new Singleton();

  private Singleton() {}

  public static Singleton getInstance() {
    return instance;
  }

}
```

Using reflection to separate two layers

If we want to switch database from one to another, we need to instantiate another concrete factory accordingly, in the process the source code needs to be changed. A better way is that, we can define the name of concrete factory into configure file, and then using reflection to instantiate an instance, so that we can switch the database only by changing the configure file instead of modifying the source code. In this case, defining two concrete factories are duplicated, only one is enough.

Here is a sample code of instantiate a class from properties file.

```
public class demo {
    public String toString(){
        return "hello world";
    }

    public static void main(String[] args){
        Properties prop=new Properties();
        try{
            InputStream in=new BufferedInputStream(new
FileInputStream("src/resource.properties"));
            prop.load(in);

            try {
        Class clz=Class.forName(prop.getProperty("classname"));
       demo object= (demo)clz.getDeclaredConstructor().newInstance();
                System.out.println(object);
            }catch (ClassNotFoundException e) {
                e.printStackTrace();
            }catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            } catch (NoSuchMethodException e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Task 2

In **singleton** package, please merge two concrete factories into one concrete factory (`DaoFactoryImpl`), and you need make sure that the instantiate process for `StaffDao` and `ComputerDao` is by reflection, and the class name of concrete factories are from `resource.properties`.

Modify your concrete factory (`DaoFactoryImpl`) to be a **singleton**.

Your code need to match the client.