# Implementing the Factory & Proxy patterns to replace applicationContext.xml & @Before Advice respectively

Consider that a bank offers multiple accounts. You must be able to create SavingAccount & CurrentAccount customers. Whenever an operation is performed on a Saving Account, record a specific log message as a pre-step before the actual operation execution.

1. Create a JAVA Project

2. Create a component:

Following is the Account.java file:

```
package com.demo.account;

public interface Account {
     String getTotalBenefits();
}
```

3. Create concrete components classes:

Following is the SavingAccount.java file:

```
package com.demo.account;

public class SavingAccount implements Account {
     @Override
    public String getTotalBenefits() {
       return "This account has 4% interest rate with per
day $5000 withdrawal limit";
    }
}
```

4. Let’s create another concrete class for Account component:
Following is the CurrentAccount.java file:

```java
package com.demo.account;

public class CurrentAccount implements Account {
	@Override
	public String getTotalBenefits() {
		return "There is no withdrawal limit for current
account";
	}
}
```

5. Let's create a Factory class to obtain a specific bean object (SavingAccount / CurrentAccount) depending on user’s request.

***Please note that we are using a Proxy class for SavingAccount SavingAccountProxy to perform additional operation of logging on SavingAccount operation.***

Following is the AccountFactory.java file:

```java
package com.demo.factory;

import com.demo.account.Account;
import com.demo.account.CurrentAccount;
import com.demo.account.SavingAccount;
import com.demo.proxy.SavingAccountProxy;

public class AccountFactory {

	public Account getBean(String beanType) {
		if(beanType.equals("savings"))
			return new SavingAccountProxy();
		else if(beanType.equals("current"))
			return new CurrentAccount();

		return null;
	}

}
```

6. Let's create an Aspect class which is supposed to be logging a logger message.

Following is the **LoggingAspect.java** file:

```java
package com.demo.aspect;

public class LoggingAspect {

     public void loggingAdvice() {
          System.out.println("Logging from the Advice");
     }

}
```

7. Let's create a Proxy class for SavingAccount so that it can perform additional operations apart from traditional methods of business logic.

***Note that it is invoking loggingAdvice() before the invocation of super class business method.***

Following is the **SavingAccountProxy.java** file:

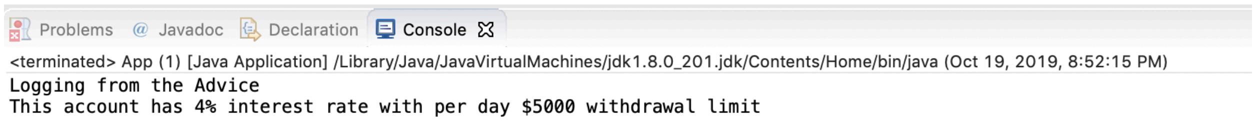
```java
package com.demo.proxy;

import com.demo.account.SavingAccount;
import com.demo.aspect.LoggingAspect;

public class SavingAccountProxy extends SavingAccount {

     public String getTotalBenefits() {
          new LoggingAspect().loggingAdvice();
          return super.getTotalBenefits();
     }

}
```

8. Let's now write some test code to see how the Factory & Proxy pattens are working together:

Following is the **App.java** file:

```
package com.demo.main;

import com.demo.account.SavingAccount;
import com.demo.factory.AccountFactory;

public class App {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		AccountFactory accountFactory = new AccountFactory();
		SavingAccount savingAccount = (SavingAccount) 
accountFactory.getBean("savings");
		System.out.println(savingAccount.getTotalBenefits());
	}

}
```

9. Let's run this demo class and see the following output at the console:



```
<terminated> App (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java (Oct 19, 2019, 8:52:15 PM)
Logging from the Advice
This account has 4% interest rate with per day $5000 withdrawal limit
```