

# Managed Beans I

# Property & Method Names

Method Names	Property Name	Example JSF Usage
getFirstName setFirstName	firstName	<code>#{customer.firstName}</code> <code>&lt;h:inputText value="#{customer.firstName}"/&gt;</code>
isExecutive setExecutive (boolean property)	executive	<code>#{customer.executive}</code> <code>&lt;h:selectBooleanCheckbox value="#{customer.executive}"/&gt;</code>
getExecutive setExecutive (boolean property)	executive	<code>#{customer.executive}</code> <code>&lt;h:selectBooleanCheckbox value="#{customer.executive}"/&gt;</code>
getZIP setZIP	ZIP	<code>#{address.ZIP}</code> <code>&lt;h:inputText value="#{address.ZIP}"/&gt;</code>

# Business logic

How to incorporate Business logic into Web App?

1. Making separate methods for the business logic
2. Passing and returning simple types
3. Implementing interfaces of the business logic
4. Using dependency injection

# Managed beans typically have three parts

## 1.Bean properties (i.e., pairs of getter and setter methods)

- One pair for each input element
- Setter methods called automatically by JSF when form submitted. Called before action controller method.

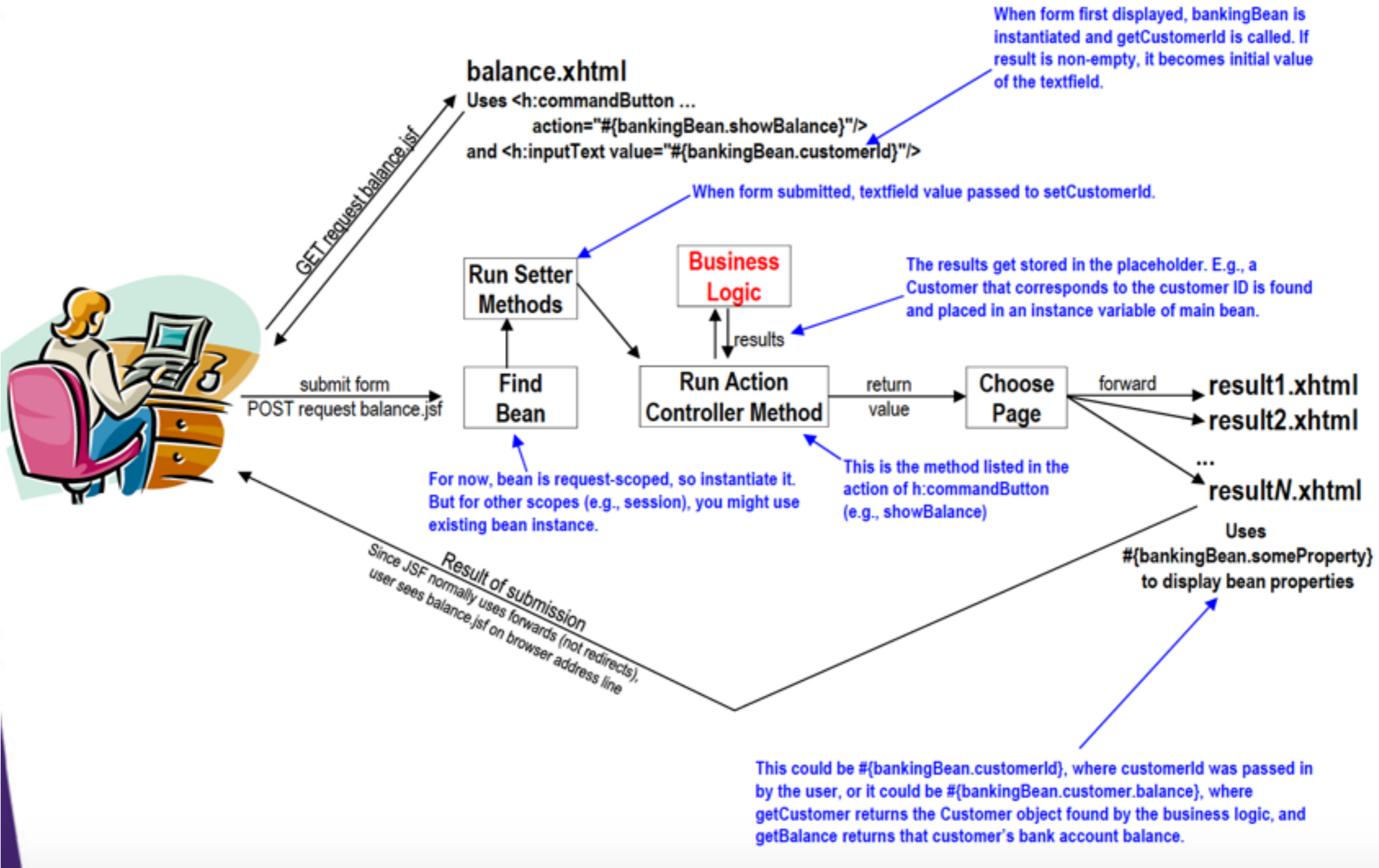
## 2.Action controller methods

- Often only one, but could be several if the same form has multiple buttons
- Action controller method (corresponding to the button that was pressed) called automatically by JSF

## 3.Placeholders for results data

- Not automatically called by JSF: to be filled in by action controller method based on results of business logic.
- Needs a getter method so value can be output in results page, but no requirement to have a setter method

# Business Logic

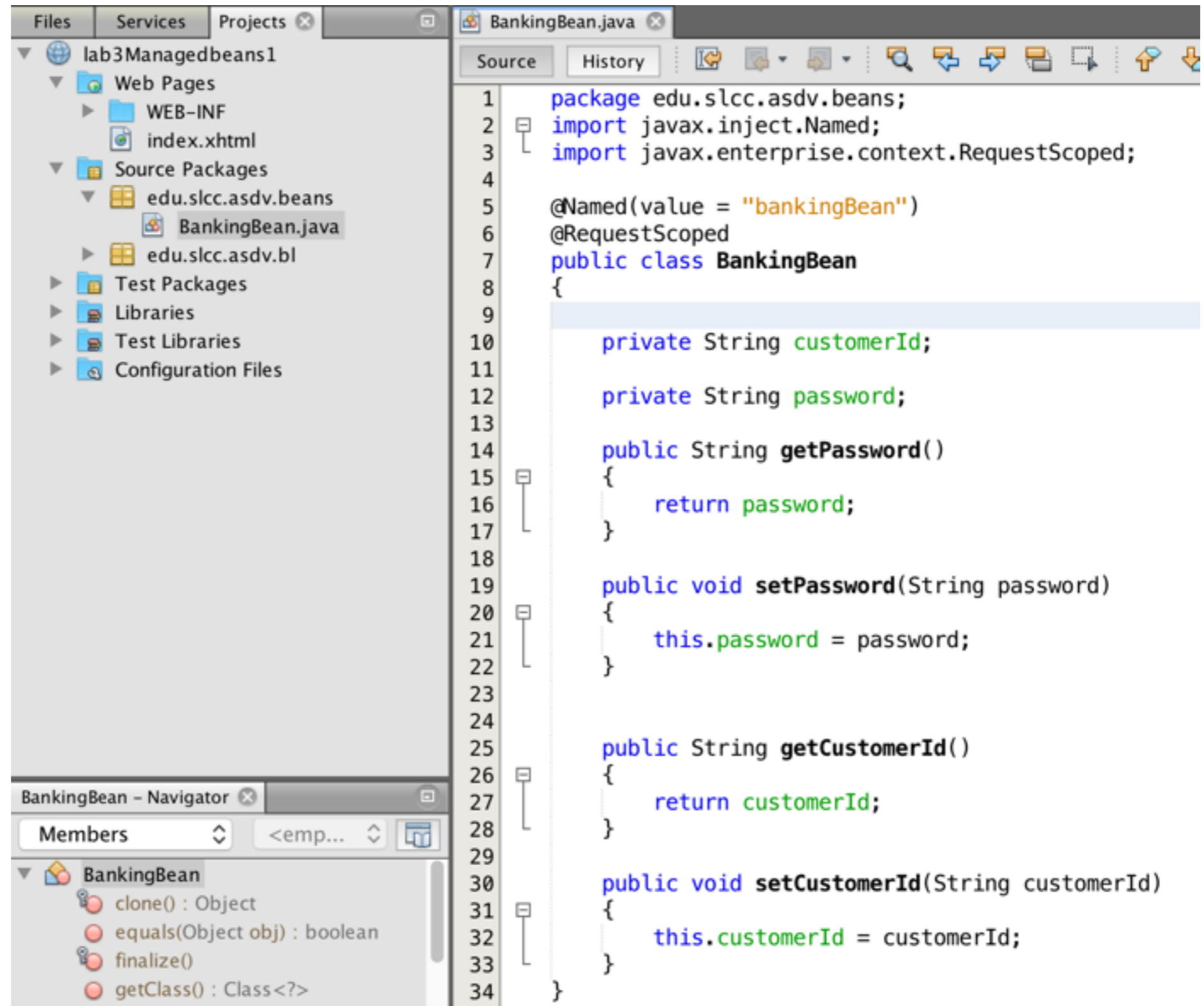


# Basic Guidelines for implementing Business Logic

- Use a separate method (**do always**)
  - Do not compute the derived data directly in the action controller method, but use a separate method.
- Simple types in, simple types out (**do always**)
  - Never return a ResultSet (database data ) or anything specific to how you found the data. Return an object representing the result itself.
- Code to interfaces (**do always**)
  - Make an interface such as CustomerLookupService and use that type. Prevents accidental dependence on concrete type.
- Use dependency injection (do sometimes)
  - Inject the concrete type so nothing in main class changes when you swap out concrete implementations of the interface.

# Practice

1. Create a new Web App lab3ManagedBeans1
2. Create a managed bean BankingBean, request scope under package edu.lcc.asdv.beans
3. Use Netbeans Insert code for the properties shown



The screenshot shows the NetBeans IDE with the following components:

- Project Explorer:** Shows a project named 'lab3ManagedBeans1' with a package 'edu.lcc.asdv.beans' containing the 'BankingBean.java' file.
- Source Editor:** Displays the code for 'BankingBean.java' with line numbers 1 through 34. The code includes package declarations, imports, annotations, and methods for 'customerId' and 'password'.
- BankingBean - Navigator:** Shows the class hierarchy and methods for 'BankingBean', including 'clone()', 'equals()', 'finalize()', and 'getClass()'.

```
1 package edu.lcc.asdv.beans;
2 import javax.inject.Named;
3 import javax.enterprise.context.RequestScoped;
4
5 @Named(value = "bankingBean")
6 @RequestScoped
7 public class BankingBean
8 {
9
10     private String customerId;
11
12     private String password;
13
14     public String getPassword()
15     {
16         return password;
17     }
18
19     public void setPassword(String password)
20     {
21         this.password = password;
22     }
23
24
25     public String getCustomerId()
26     {
27         return customerId;
28     }
29
30     public void setCustomerId(String customerId)
31     {
32         this.customerId = customerId;
33     }
34 }
```

# Business Logic

Create POJO Customer  
package edu.slcc.asdv.bl

The screenshot shows an IDE window with a project named 'lab3Managedbeans1'. The project structure includes 'Web Pages', 'Source Packages', and 'Test Packages'. Under 'Source Packages', there is a package 'edu.slcc.asdv.bl' containing a file 'Customer.java'. The code editor displays the following Java code:

```
1 package edu.slcc.asdv.bl;
2 public class Customer
3 {
4     private final String id;
5     private final String firstName;
6     private final String lastName;
7     private final double balance;
8
9     public Customer(String id,
10                    String firstName,
11                    String lastName,
12                    double balance)
13     {
14         this.id = id;
15         this.firstName = firstName;
16         this.lastName = lastName;
17         this.balance = balance;
18     }
19
20     public String getId(){return id;}
21
22     public String getFirstName(){return (firstName);}
23
24     public String getLastName(){return (lastName);}
25
26     public double getBalance(){return balance;}
27
28     public String getBalanceNoSign()
29     {
30         String balanceString
31             = String.format("%.2f", Math.abs(balance));
32         return (balanceString);
33     }
34 }
```

The 'Members' view at the bottom shows the class structure:

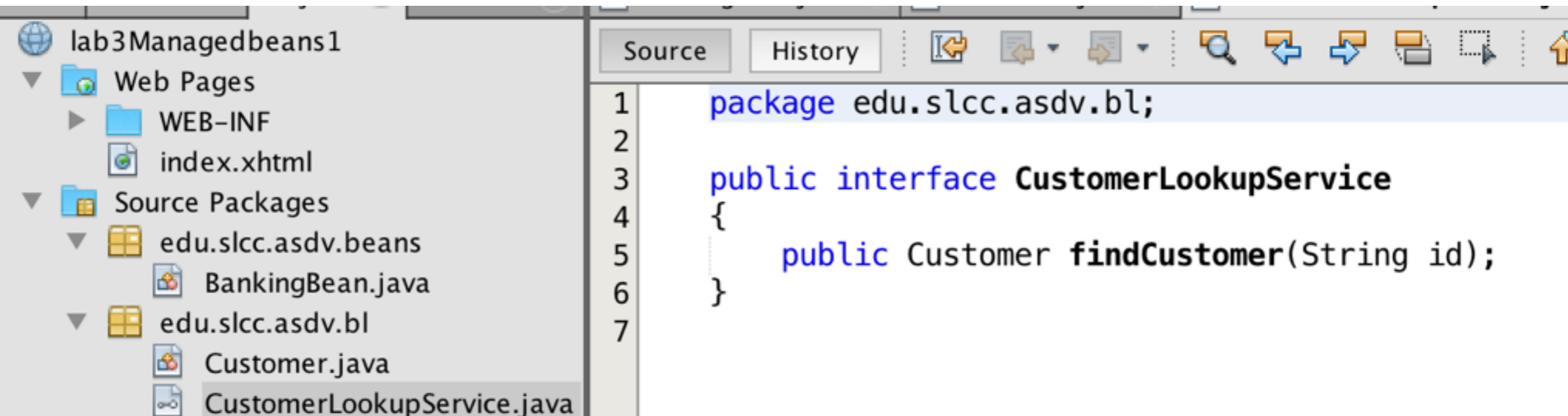
- finalize()
- getBalance() : double
- getBalanceNoSign() : String
- getClass() : Class<?>



# Business Logic

## Create Interface

### CustomerLookupService

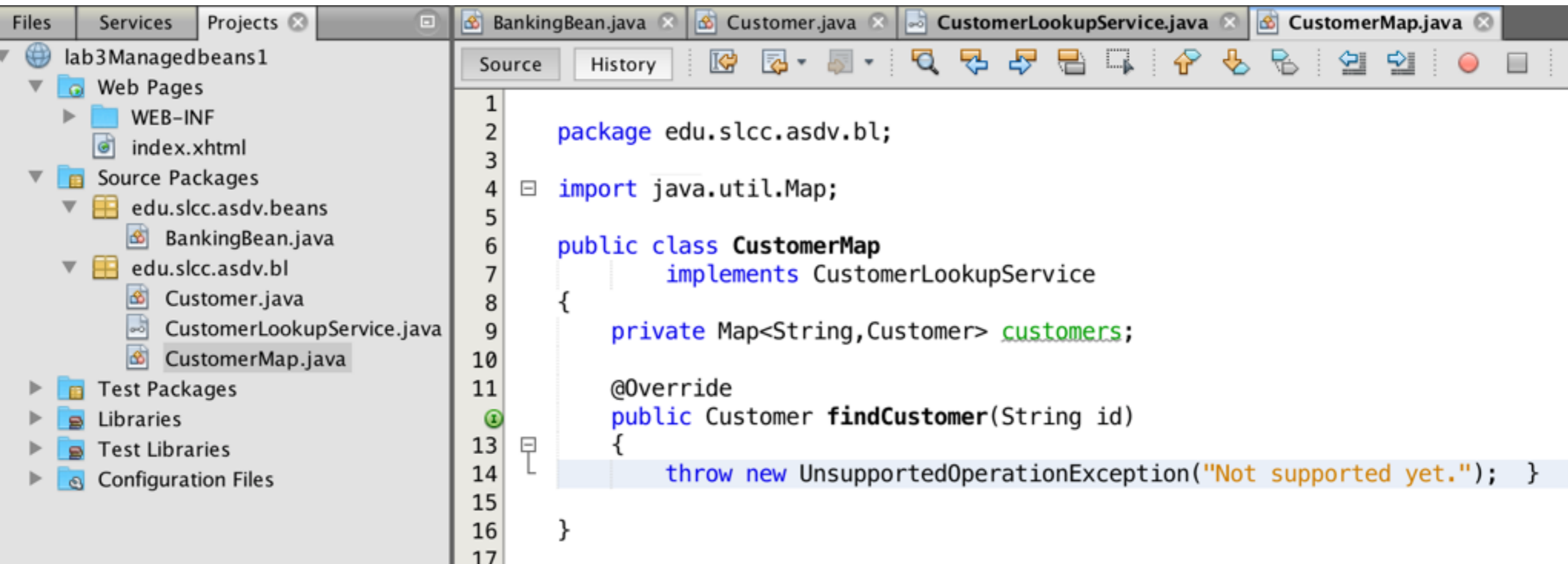


```
1 package edu.slcc.asdv.bl;
2
3 public interface CustomerLookupService
4 {
5     public Customer findCustomer(String id);
6 }
7
```

# Business Logic

## Create “interface”

### CustomerMap

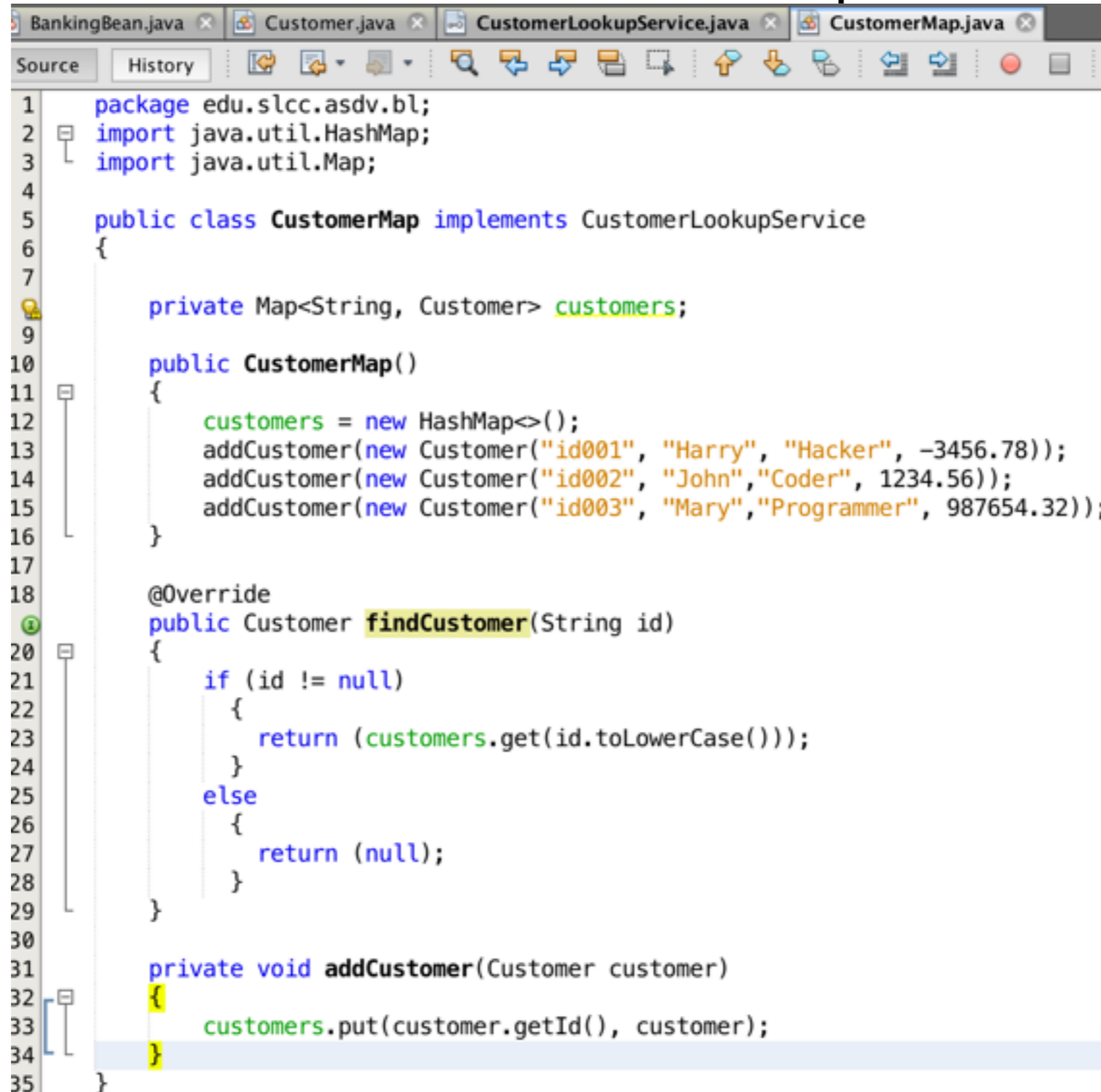


The screenshot shows an IDE window with the following tabs: BankingBean.java, Customer.java, CustomerLookupService.java, and CustomerMap.java. The left sidebar shows a project structure for 'lab3Managedbeans1' with folders for 'Web Pages', 'Source Packages', 'Test Packages', 'Libraries', 'Test Libraries', and 'Configuration Files'. Under 'Source Packages', there are sub-packages 'edu.slcc.asdv.beans' and 'edu.slcc.asdv.bl'. The 'edu.slcc.asdv.bl' package contains the files 'Customer.java', 'CustomerLookupService.java', and 'CustomerMap.java'. The main editor window displays the source code for 'CustomerMap.java'.

```
1 package edu.slcc.asdv.bl;
2
3
4 import java.util.Map;
5
6 public class CustomerMap
7     implements CustomerLookupService
8 {
9     private Map<String, Customer> customers;
10
11     @Override
12     public Customer findCustomer(String id)
13     {
14         throw new UnsupportedOperationException("Not supported yet."); }
15
16 }
17
```

# Business Logic

Complete the code for “interface”  
CustomerMap



```
BankingBean.java x Customer.java x CustomerLookupService.java x CustomerMap.java x
Source History
1 package edu.slcc.asdv.bl;
2 import java.util.HashMap;
3 import java.util.Map;
4
5 public class CustomerMap implements CustomerLookupService
6 {
7
8     private Map<String, Customer> customers;
9
10    public CustomerMap()
11    {
12        customers = new HashMap<>();
13        addCustomer(new Customer("id001", "Harry", "Hacker", -3456.78));
14        addCustomer(new Customer("id002", "John", "Coder", 1234.56));
15        addCustomer(new Customer("id003", "Mary", "Programmer", 987654.32));
16    }
17
18    @Override
19    public Customer findCustomer(String id)
20    {
21        if (id != null)
22        {
23            return (customers.get(id.toLowerCase()));
24        }
25        else
26        {
27            return (null);
28        }
29    }
30
31    private void addCustomer(Customer customer)
32    {
33        customers.put(customer.getId(), customer);
34    }
35 }
```