

Chapter 3 – Implementing Classes

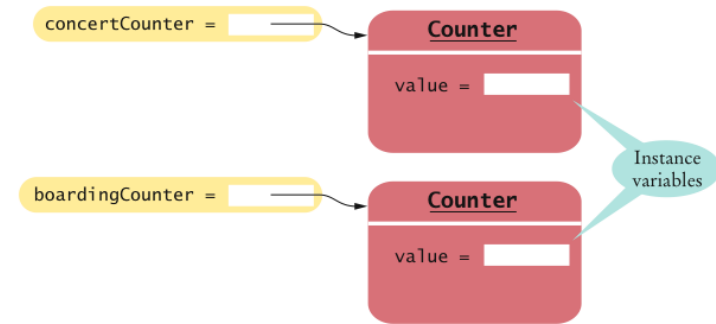
1. Instance Variables
2. Accessing Instance Variables
3. Encapsulation
4. *Class* Comment
5. Implementing Constructors
6. *Method* Declaration
7. Implicit Parameter
8. Implicit Parameters and *this*

Instance Variables

- **Instance variables** store the data of an **object**

The class declaration specifies the instance variables:

```
public class Counter
{
    private int value;
    ...
}
```



- An instance variable declaration consists of the following parts:
 - **access specifier** (`private`)
 - **type of variable** (such as `int`)
 - **name of variable** (such as `value`)
- Each object of a class has its **own set** of instance variables
- You should declare all instance variables as **private**

Accessing Instance Variables

The `count` method advances the counter value by 1:

```
public void count()  
{  
    value = value + 1;  
}
```

The `getValue` method **returns** the current value:

```
public int getValue()  
{  
    return value;  
}
```

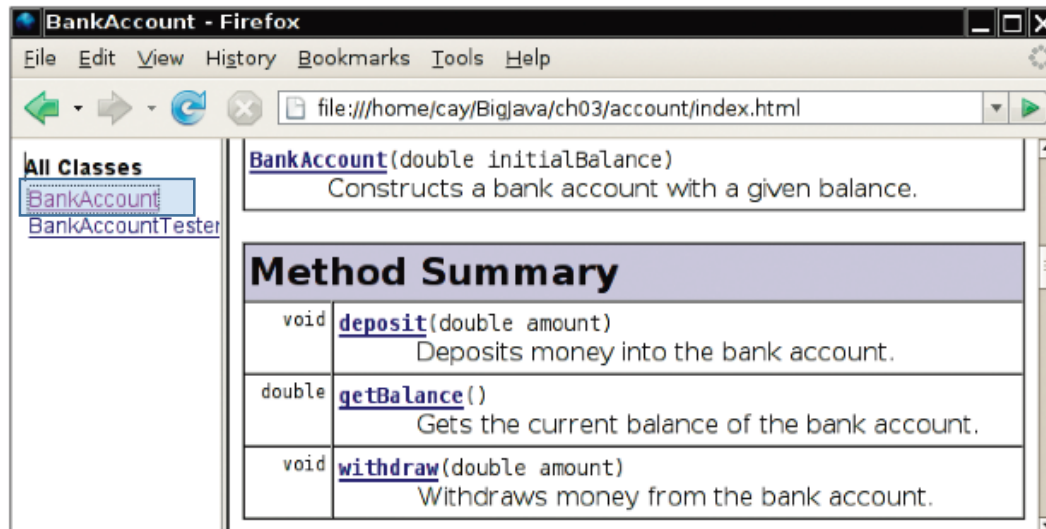
Private instance variables **can only** be accessed by methods of the same class

Encapsulation

- **Encapsulation** is the process of **hiding** object **data** and providing **methods** for data **access**
- To encapsulate data, declare instance variables as `private` and declare public methods that access the variables
- Encapsulation allows a programmer to use a class **without** having to **know** its **implementation**
- Information hiding makes it simpler for the implementor of a class to locate errors and change implementations

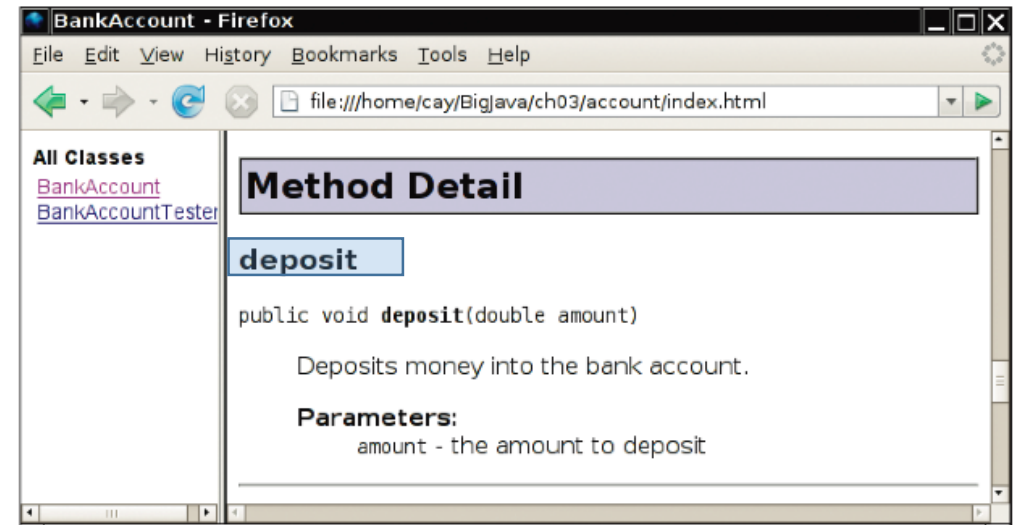
Class Comment

- Provide **documentation** comments for
 - *every class*
 - *every method*
 - *every parameter*
 - *every return value*



The screenshot shows a web browser window titled "BankAccount - Firefox" with the address bar displaying "file:///home/cay/BigJava/ch03/account/index.html". The page content includes a sidebar with "All Classes" listing "BankAccount" and "BankAccountTester". The main content area shows the class signature "BankAccount(double initialBalance)" with a comment "Constructs a bank account with a given balance." Below this is a "Method Summary" table:

Method Summary	
void	deposit (double amount) Deposits money into the bank account.
double	getBalance () Gets the current balance of the bank account.
void	withdraw (double amount) Withdraws money from the bank account.



The screenshot shows the same web browser window, but the "Method Detail" section for the "deposit" method is expanded. It displays the following code snippet:

```
public void deposit(double amount)
    Deposits money into the bank account.
```

Parameters:
amount - the amount to deposit

Implementing Constructors

Constructors contain instructions to **initialize** the instance variables of an object:

```
public BankAccount ()  
{  
    balance = 0;  
}
```

```
public BankAccount (double initialBalance)  
{  
    balance = initialBalance;  
}
```

Method Declaration

Syntax *accessSpecifier returnType methodName(parameterType parameterName, . . .)*
 {
 method body
 }

Example

These methods
are part of the
public interface.

```
public void deposit(double amount)  
{  
    balance = balance + amount;  
}
```

This method does
not return a value.

A mutator method modifies
an instance variable.

```
public double getBalance()  
{  
    return balance;  
}
```

This method has
no parameters.

An accessor method returns a value.

Local Variables

- **Local** and **parameter variables** belong to a method
 - *When a method or constructor runs, its local and parameter variables come to life*
 - *When the method or constructor exits, they are removed immediately*
- **Instance variables** belongs to an objects, not methods

- *When an object is constructed, its instance variables are created*
- *The instance variables stay alive until no method uses the object any longer*

- In Java, the **garbage collector** periodically reclaims objects when they are no longer used
- Instance variables are initialized to a default value, **but you must initialize local variables**

Implicit Parameter

The **implicit parameter** of a method is the object on which the **method is invoked**

```
public void deposit(double amount)
{
    balance = balance + amount;
}
```

In the call

```
momsSavings.deposit(500)
```

The **implicit** parameter is `momsSavings` and the **explicit** parameter is `500`

When you refer to an instance variable inside a method, it means the instance variable of the implicit parameter

Implicit Parameters and `this`

The `this` reference denotes the **implicit parameter**

```
balance = balance + amount;
```

actually means

```
this.balance = this.balance + amount;
```

When you refer to an instance variable in a method, the compiler automatically applies it to the `this` reference