Chapter 9:
Classes with Instance Variables
or Classes=Methods+Variables

Asserting Java
© Rick Mercer
Classes

- Classes are
  - a collection of methods and data
  - a blueprint used to construct many objects
  - a great way to partition a software system
  - A way to implement any type

  - A type defines a set of values, and the allowable operations on those values
Putting it All Together in a Class

- Even though quite different in specific methods and state, virtually all classes have these things in common:
  - variables store the state of the objects
  - constructors initialize the state of an object
  - some messages modify the state of objects
  - some messages provide access to the state of objects
General form of a Java class

```java
public class class-name {
    //--instance variables
    private class-name identifier ;
    private primitive-type identifier ;

    // Constructor
    public class-name ( parameters ) {
    }

    //--Methods
    public return-type method-name-1 ( parameters ) {
    }

    public return-type method-name-N ( parameters ) {
    }
}
```
public class BankAccount {
    //--instance variables
    private String ID;
    private double balance;

    // Constructor to initialize the state
    public BankAccount(String initID, double initBalance) {
        ID = initID;
        balance = initBalance;
    }

    // Credit this account by depositAmount
    public void deposit(double depositAmount) {
        balance = balance + depositAmount;
    }

    // Debit this account by withdrawalAmount
    public void withdraw(double withdrawalAmount) {
        balance = balance - withdrawalAmount;
    }
}
public String getID() {
    return ID;
}

public double getBalance() {
    return balance;
}

public String toString() {
    return ID + "$" + balance;
}

} // End class BankAccount
Objects are a lot about Operations and State

- The methods declared **public** are the messages that may be sent to any instance of the class *the objects*
- The instance variables declared **private** store the state of the objects
  - no direct access from outside the class
- Every instance of a class has its own separate state
  - If you have 5,234 BankAccount objects, you'll have 5,234 IDs and 5,234 balances
The Unified Modeling Language (UML) class diagram -- another view of a class methods and state

<table>
<thead>
<tr>
<th>BankAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>-String ID</td>
</tr>
<tr>
<td>-double balance</td>
</tr>
<tr>
<td>+BankAccount(String, double)</td>
</tr>
<tr>
<td>+withdraw(double)</td>
</tr>
<tr>
<td>+deposit(double)</td>
</tr>
<tr>
<td>+double getBalance()</td>
</tr>
<tr>
<td>+String getID()</td>
</tr>
</tbody>
</table>
Sample Messages

```java
BankAccount anAcct = new BankAccount("Moss", 500.00);
anAcct.withdraw(60);
anAcct.deposit(147.35);
// These assertions should pass
assertEquals("Moss", anAcct.getID());
assertEquals(500.0, anAcct.getBalance(), 0.001);
```
Instance Variables

- State of objects stored as instance variables
  - these are the variables that are declared inside the class, but outside of the method bodies
  - each instance of the class (object) has its own values stores in its own instance variables
    - with 954 objects you have 954 sets of instance variables
  - all methods in the class can access instance variables
    - and most methods will reference at least one of the instance variables
    - the data and methods are related
  - when declared private, no one else can access the instance variables
    - the state is encapsulated, nice and safe behind methods
Constructors

- Constructor
  - a special method that initializes the state of objects
  - gets invoked when you construct an object
  - has the same name as the class
  - does not have a return type
    - a constructor returns a reference to the instance of the class
- Assigning object references to reference variables

```java
String aString = new String("Initial state");
BankAccount anAcct = new BankAccount("Katey", 10.00);
JFrame window = new JFrame("My Application");
```
Constructors

- General form: calling a constructor to construct and initialize an object

```
class-name object-name = new class-name (initial-state);
```

- `class-name`: name of the class
- `object-name`: any valid Java identifier
- `initial-state`: zero or more arguments supplied during instance creation.
Constructors

- A constructor uses the arguments *initial-state* to help initialize the private instance variable

```java
public BankAccount(String initID, double initBalance) {
    ID = initID;
    balance = initBalance;
}
```

```java
BankAccount anAcct = new BankAccount("Kellen", 123.45);
```
Methods that Access State

- Other methods return the current value of an object's data member, or return some information related to the state of an object

  ```java
  anAccount.getBalance() // return current balance
  aString.substring(0, 3) // Get part of a string
  ```

- Some accessing methods simply return values of instance variables, others require some processing
return in non-void methods

- Java's return statement provide the means to return information, General form
  
  ```
  return expression ;
  ```

- When return executes, the value of `expression` replaces the message `control returns to the message`

- `expression` must match the return type of method heading
  - String return type means you must return a String
  - double return type? Return a double

- You can not return an an value from a void method
Returning state

```java
public String getID() {
    return ID;
}

public double getBalance() {
    return balance;
}

System.out.println("anAcct.getID(): " + anAcct.getBalance());
```

Output: Kellen 123.45
Methods that Modify State

- Some methods modify (change) the state of an object:

```java
anAccount.withdraw(120.00)
    // Account balance has been reduced by 120.00

    // Modify the state of any BankAccount object
    public void withdraw(double amount) {
        // balance is an instance variable
        balance = balance + amount;
    }
```
Naming Conventions

- Rules #1, 2, and 3:
  1: Always use meaningful names
  2: Always use meaningful names
  3: Always use meaningful names

- Rule #4

  Constructors: Same name as the class
public or private?

- When designing a class, do this at least for now
  - declare constructors and methods public
  - declare instance variables private
- Public messages can be sent from wherever the object is in scope
- Private state can not be messed up

```java
BankAccount myAcct = new BankAccount("Me", 10.00);
myAcct.balance = myAcct.balance + 99999999.99;
```
Protecting an Object's State

- Access modes make operations available to clients and also protect the state
- Instance variables and methods are known as follows

<table>
<thead>
<tr>
<th>Access Mode</th>
<th>Where is the symbol known?</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>From all methods of the class and in the scope (where it is known) of the object</td>
</tr>
<tr>
<td>private</td>
<td>Known only in the class</td>
</tr>
</tbody>
</table>
Some Guidelines

- Recommendations for writing your first classes:
  - declare instance variables `private` after class definition
  - declare constructors `public` no return type, no static!
  - declare most methods `public` no static!
    • however, private helper methods are often useful
  - look at examples in Book and on the Code Demos page as patterns
- use one file to store the class (no main method)
- use a unit test to test instances of the class
Object-Oriented Design Guidelines

- Object-oriented design guideline
  - a rule of thumb intended to help produce good object-oriented software
- Example: When deciding what access users of a class should have to methods and instance variables
- OOD Guideline

All data should be hidden within its class

- Translation: make instance variables private
Why private?

- If `balance` were public, what is it after this?
  ```javascript
  myAcct.balance = myAcct.balance - myAcct.balance;
  ```
- With `balance` private, compiletime error occurs
- You want programmers to use the methods
  - imagine the other things that occur during a real withdraw
    - write a transaction to a file
    - check for trying to withdraw more than balance
- Also, you can later change the class
- Also, programmers only need to know the methods
Cohesion within a class

• The methods of a class should be strongly related
  ○ don't put a blastOff method in BankAccount
• The instance variables should be strongly related
  ○ don't put double acceleration in BankAccount
• Cohesion means solidarity, hanging together
• Classes with high cohesion are a better design
  ○ the methods and data are related
• OOD Guideline

Keep related data and behavior in one place