What Every Java Developer Should Know About Roles, Responsibilities and Collaborative Contracts

Rebecca Wirfs-Brock
rebecca@wirfs-brock.com
Wirfs-Brock Associates, Inc.
Design Constructs

an application = a set of interacting objects

an object = an implementation of one or more roles

a role = a set of related responsibilities

a responsibility = an obligation to perform a task or know information

a collaboration = an interaction of objects and/or roles

a contract = an agreement outlining the terms of a collaboration
package ProblemDomain;
import java.math.BigDecimal;
/** @stereotype mi-detail */
public class CashSaleDetail {
    /** This indicates how many of this item are being purchased. */
    private int qty;
    /** This would allow you to use units like: each, pounds, case.
     * Of course, you would have to have the intelligence on the
     * product pricing side as well. We won't be using this.<g> */
    private int UOM = 0;
    // private BigDecimal negotiatedPrice;
    /** This is the item being scanned in/sold. This object holds all the pertinent
details. */
    @clientCardinality 0..*
    @supplierCardinality 1
    private ProductDesc product;

    /* ========================================
    * Constructors
    * ======================================== */
    public CashSaleDetail(ProductDesc prod) {
        product = prod;
        qty = 1;
    }
    /* ========================================
    * Business Methods
    * ======================================== */
    public BigDecimal calcTotal() {
        return product.calculatePriceForQty(qty);
    }
    public boolean verifyAvailability() {
        return false;
    }
    public void deductQty() {
    }
    /* ========================================
    * Accessor/Mutator Methods
    * ======================================== */
    public int getQty() {
        return qty;
    }
    public void setQty(int aQty) {
        qty = aQty;
    }
    public ProductDesc getProductDesc() {
        // Should probably return a clone to be safer.
        return product;
    }
} // ENDCLASS CashSaleDetail
The Dilemma

How do you express design constructs in Java code?
How can you describe your design so other programmers don’t misuse your classes?
Can you do this without a lot of work or tool support?
Why Roles?

Each object serves a purpose. Each plays at least one role in a given context:

A role is a category of objects that can be used interchangeably.

When a role is always played by the same type of object, the two are equivalent.

If more than one kind of object can fulfill the same responsibilities, a role represents a “slot” in the software that can be fulfilled by different players.
## Unlined Side of an RRC Card: Describing an Object

<table>
<thead>
<tr>
<th>Role Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose:</strong> <em>Brief Definition</em></td>
</tr>
<tr>
<td><strong>Stereotypes:</strong> <em>Information-holder, information-provider, structurer, coordinator, controller, service-provider, or interfacerc</em></td>
</tr>
<tr>
<td><strong>Utility level:</strong> <em>Application-specific, generic, or enterprise-wide</em></td>
</tr>
<tr>
<td><strong>Patterns:</strong> <em>Singleton, Whole Value, etc.</em></td>
</tr>
</tbody>
</table>
Describing Roles During Design

Build a definition that explains an object’s purpose and any distinguishing traits:

A RazzmaFrazzer accurately and speedily translates Razzmas into Frazzes

More generally:

An object is a type of thing that has certain characteristics
Stereotypes

“A conventional, formulaic, and oversimplified conception, opinion, or image.”
—American Heritage Dictionary, Third Edition

“Something conforming to a fixed or general pattern; especially a standardized mental picture held in common by members of a group and representing an oversimplified opinion.”
—Webster’s Seventh New Collegiate Dictionary
Stereotypical Descriptions

A *service provider* does specific work

A *controller* makes decisions and closely manages the work of others

A *coordinator* make simpler decisions and delegates work

An *information holder* contains certain facts

A *structurer* manages an organization of objects

An *interfacer* conveys requests between different levels
Preserving the Purpose in Class Comments

/**
 * This class implements a hashtable, which maps keys to values. Any
 * non-<code>null</code> object can be used as a key or as a value.
 * <p>
 * To successfully store and retrieve objects from a hashtable, the
 * objects used as keys must implement the <code>hashCode</code>
 * method and the <code>equals</code> method. …
 * 
 * A Hashtable manages a store of objects, each
 * referenced by a unique “key”. Hashtable maintains
 * the associations between each key and its stored
 * object. It provides services for adding, deleting,
 * querying about, and retrieving objects.

**Guideline**: Include both an overview and details in comments
RRC Cards: Describing Candidate Objects

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stereotypes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utility Level:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patterns:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Describing Responsibilities in JavaDoc

Responsibilities are high-level. Often, JavaDoc includes low level information.

Developers need both perspectives.

In class and method comments include a responsibility-based overview:

- Describe what a client needs to understand about the class in general—describe what the class does and any important behaviors and characteristics.
- Describe its major responsibilities and how the class can be used and/or extended.

...then go into the details.
Design Roles

*a role* = *a set of related responsibilities*

*a responsibility is implemented by one or more methods*

A role is a higher abstraction than a Java class or interface

Roles can be

primary – consisting of a set of responsibilities that make an object uniquely what it is

secondary – responsibilities an object assumes to fit in to a community of technical libraries, environments and business domains
Implementing Roles

*In Java, a role can be:*

*specified by an interface, and/or*

*implemented by one or more classes*
Constructing Classes

Define a class to implement any singular, primary role

<table>
<thead>
<tr>
<th>BankAccount</th>
<th>&lt;&lt;class&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Knows its customer</em></td>
<td></td>
</tr>
<tr>
<td><em>Knows its account ID</em></td>
<td></td>
</tr>
<tr>
<td><em>Maintain balance</em></td>
<td></td>
</tr>
<tr>
<td><em>Maintain transaction history</em></td>
<td></td>
</tr>
</tbody>
</table>

```java
BankAccount

Currency currentBalance;
Currency dailyWithdrawalLimit;
AccountID accountIdent;
...

public currency getBalance();
public void postDebit(Currency amount, Calendar date);
public void postCredit(Currency amount, Calendar date);
public void postInterest(Currency amount, Calendar date);
public historyRecord getHistory(String accountPeriod);
public historyRecord getRecentHistory(String accountPeriod);
...
```
Declaring a Role Interface

Declare an interface for a role that many different objects could support as being part of a larger “community.”

Objects that represent a FinancialAsset of the bank can be assigned a current and projected valuation. The FinancialAsset role is specified by a FinancialAsset interface. (A domain-specific role)
Framework designers also define roles that objects can assume to fit into a framework specific environment.
Adding Secondary Roles to Class Definitions

Add roles to a class that it should implement as part of fitting into the community

The BankAccount class implements the FinancialAsset interface and assumes the secondary role of FinancialAsset. Since we also intend to use it in an EJB environment, it also implements the EntityBean interface.

Rename classes to fit within their technical environment

If we are implementing a BankAccount in an EJB framework, rename it BankAccountEntityBean.
Class = Primary Role + Secondary Roles

<table>
<thead>
<tr>
<th>BankAccount</th>
<th>FinancialAsset</th>
<th>EntityBean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows its customer</td>
<td>Knows current valuation</td>
<td>Knows its context</td>
</tr>
<tr>
<td>Knows its account ID</td>
<td></td>
<td>Initialize</td>
</tr>
<tr>
<td>Maintain balance</td>
<td></td>
<td>Retrieve and Store</td>
</tr>
<tr>
<td>Maintain transaction history</td>
<td></td>
<td>Activate/passivate</td>
</tr>
</tbody>
</table>

BankAccountEntityBean
Collaborative Contracts

“An agreement between two or more parties, especially one that is written and enforceable by law.”

—The American Heritage Dictionary, Third Edition
Responsibility-Driven Design
Contracts

“The ways in which a given client can interact with a given server are described by a contract. A contract is the list of requests that a client can make of a server. Both must fulfill the contract: the client by making only those requests the contract specifies, and the server by responding appropriately to those requests. …For each such request, a set of signatures serves as the formal specification of the contract.”

—Wirfs-Brock, Wilkerson & Wiener, Designing Object-Oriented Software
Finding Contracts

A class which implements a single role that is viewed by its clients in identical ways offers a single contract.

Classes whose responsibilities are partitioned by distinctly different client usage should support multiple contracts.

Example: BankAccount Contracts
- Balance Adjustment
- Balance Inquiry
- Managing Challenge Data
- Transaction History
Preserving Contracts

In a good implementation, any class that inherits a contract should support all of it and not cancel out any behavior.

A subclass can extend a superclass by attaching new responsibilities and by defining new contracts.
Reconstructing Contracts from Existing Interface Definitions

A Java interface may map to one or more contracts

For example, the EntityBean interface defines two contracts:

1. “Bean Initialization”
   One, used by the container to initialize a well-formed bean

2. “Pool Management”
   Another, also used by the container, to manage its pooled bean resources

<table>
<thead>
<tr>
<th>EntityBean</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows its context</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initialize</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve and Store</td>
<td>2</td>
</tr>
<tr>
<td>Delete from container</td>
<td>2</td>
</tr>
<tr>
<td>Activate/passivate</td>
<td>2</td>
</tr>
</tbody>
</table>
Specifying the Fine Print: Bertrand Meyers’ Contracts

“Defining a precondition and a post condition for a routine is a way to define a contract that binds the routine and its callers…. A precondition-post condition pair for a routine will describe the contract that the routine (the supplier of a certain service) defines for its callers (the clients of a service).”

—Bertrand Meyer, *Object-Oriented Software Construction*
### Example: A Meyer’s Contract for a Bean Method, `ejbPassivate()`

<table>
<thead>
<tr>
<th>Obligations</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
<td><strong>Benefits</strong></td>
</tr>
<tr>
<td>(Satisfy precondition:)</td>
<td>(From post condition:)</td>
</tr>
<tr>
<td>Call <code>ejbPassivate()</code> before</td>
<td>Bean can now be disassociated</td>
</tr>
<tr>
<td>bean is disassociated from its EJB object</td>
<td></td>
</tr>
<tr>
<td><strong>Supplier</strong></td>
<td><strong>Benefits</strong></td>
</tr>
<tr>
<td>(Satisfy post condition:)</td>
<td>(From precondition:)</td>
</tr>
<tr>
<td>Bean must be in a state that won’t tie up</td>
<td>Allows bean to clean up all non-serializable</td>
</tr>
<tr>
<td>resources or that is inconsistent</td>
<td>state prior to container disassociating bean and</td>
</tr>
<tr>
<td></td>
<td>removing it from memory</td>
</tr>
</tbody>
</table>
Unifying These Two Definitions

A design can be viewed at different levels of abstraction

  Responsibility-Driven Contract name and description
    List of clients and suppliers
    List of responsibilities defined by the contract
      Method Signatures

  Bertrand Meyer’s add precision precisely at the level where we left off:
    Method Signature
      Client obligations
      Supplier benefits
      Preconditions, post conditions, invariants
What Is Missing From These Contractual Descriptions?

From a customer’s point-of-view:

\[
\text{an object} = \text{interface} + \text{terms and conditions of use}
\]

The more we publish about the behavior of an object, the more likely it will be used as its designer intended.
Even More Details Are Needed

When objects do business, they agree to certain terms:

- Only make requests for advertised services
- Provide appropriate information
- Use services under certain conditions
- Accept the consequences of using them

The shingle that each object hangs when it opens for business should describe all these terms.
Describing Terms and Guarantees

An interface defines the vocabulary for collaboration, not important behavioral semantics. Publish these additional terms and guarantees in method and class comments:

/* If you give me shoes before noon today, they will be ready to pick up tomorrow */

/* If you pay with a credit card, your credit rating may change. All credit card payments for an amount over XXX result in a credit check. Every credit check will lower the customer’s credit rating */

/* When shining shoes, we use the best materials. Our work is guaranteed. If you are not satisfied for any reason, we will refund your payment without question. */
Keys to Keeping Your Design Evident in Code

Today, the best roundtrip engineering tool doesn’t support roles, responsibilities, contracts, or terms and guarantees…. So…

These descriptions need to be part of class and interface comments
Roles should be declared in interfaces
Classes should preserve superclass contracts and be constructed from primary and secondary roles