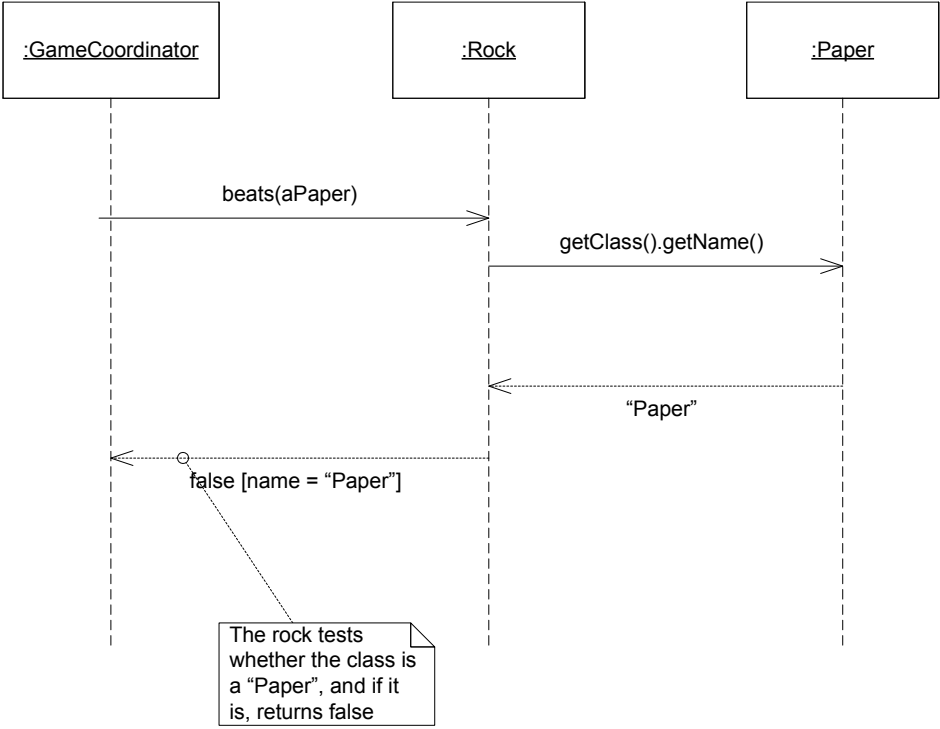
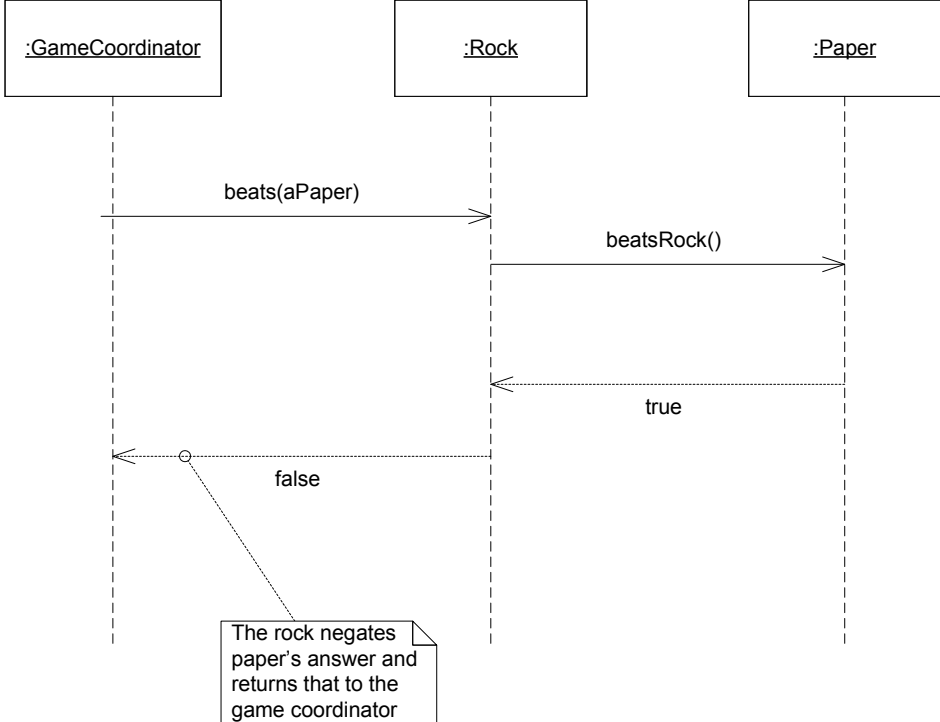


Errata for *Object Design: Roles, Responsibilities, and Collaborations*
 Last updated June 1, 2005

Page #	Correction
xix	Change sentence in second paragraph to “The informal tools and techniques in this book don’t require much more than a white board, a stack of index cards, a big sheet of paper, and chairs around a table.”
4	Change last sentence. Insert “primary” into phrase e.g. “but only one primary role because...”
5	Delete sentence: If its information is being used solely to support its service it assumes two stereotypes but only one role.”
21	getClass all places in the code should be getClass()
21	<p>Figure 1-8 should have the beats(aPaper) message coming from the GameCoordinator to the Rock object</p> <p>Corrected figure 1-8 This UML sequence diagram shows the process of deciding who wins, based on checking object type.</p>  <pre> sequenceDiagram participant GC as :GameCoordinator participant R as :Rock participant P as :Paper GC->>R: beats(aPaper) R->>P: getClass().getName() P-->>R: "Paper" R-->>GC: false [name = "Paper"] Note over R: The rock tests whether the class is a "Paper", and if it is, returns false </pre>

22	<p>Figure 1-9 should have the beats(aPaper) message coming from the GameCoordinator to the Rock object, and a false being returned by the Rock to the GameCoordinator object</p> <p>Corrected figure 1-9 This UML sequence diagram shows the process of deciding who wins based on polymorphism (illustrating the double dispatching technique)</p>  <pre> sequenceDiagram participant GC as :GameCoordinator participant R as :Rock participant P as :Paper GC->>R: beats(aPaper) R->>P: beatsRock() P-->>R: true R-->>GC: false Note over R: The rock negates paper's answer and returns that to the game coordinator </pre>
22	<p>Coding errors. See corrections below</p> <pre> public interface GameObject { public boolean beats(GameObject o); public boolean beatsRock (); public boolean beatsPaper(); public boolean beatsScissors(); } </pre>
23	<p>in Rock code...</p> <pre> public class Rock implements GameObject { public boolean beats(GameObject o) { // The receiver is a Rock. Ask the argument about a rock. return !o.beatsRock(); } } </pre>

	<pre> public boolean beatsRock() { // Return true since ties answer false return true; } public boolean beatsPaper() { //A Rock doesn't beat a Paper return false; } public boolean beatsScissors() { // A Rock beats a Scissors! return true; } } </pre>
23	<p>in Paper code...</p> <pre> public class Paper implements GameObject { public boolean beats(GameObject o) { // The receiver is a Paper. Ask the argument about paper. return !o.beatsPaper(); } public boolean beatsRock() { // A Paper beats a Rock return true; } public boolean beatsPaper() { // Return true since ties answer false return true; } public boolean beatsScissors() { // A Paper doesn't beat a Scissors! return false; } } </pre>
62	Delete callout that starts, “If you insist on using a computer...”. It is a duplicate of the callout on pages 128
79	First paragraph, line 6, should into be “in to”?
83	Top paragraph “The application will maintain additional user-supplied information and construct account history from online and other banking transactions” change to “and construct an account history from online and other banking transactions”
85	The repeated used of “whom” in the top paragraph seems stuffy. It may be right, but to Alan’s ears sounds wrong.
91	Callout “candidate’s” should be “candidates”
92	Change last paragraph to read: The synonyms for Properties, a class defined in the Java libraries, include these words: characteristics, attributes, qualities, features, and traits.

	Although “attribute” or “feature” might work “characteristic” seems stuffy and “quality” seems strained.
101	The type setting on the figure 3-3 CRC card’s comments are done poorly. Especially “Check on third-party products”.
114-115	<p>On page 115 in the section on “additional responsibilities that are more specific” we duplicate the last three paragraphs from the previous page’s list of responsibilities we found earlier. They should be omitted.</p> <p>Insert these two additional responsibilities into the list on p. 115 (in order of steps):</p> <p>Verify that student is known to the system. (From step 1. Probably can be assigned to some object that coordinates registration and interacts with a database of registered students.)</p> <p>Maintaining a proposed schedule and possibly reserving slots in courses until confirmed by a student. (From step 8. Schedules should know what their status is. Managing “course reservations” seems like a responsibility for a new object concerned with managing student-course registration status.)</p>
123	Last line “stimulus” needs to be plural “stimuli”
126	In callout replace “they” with “you”
127	The description on the EmailAddress card is very wrong. Delete the sentence “It also knows information about both the sender and the receiver it uses during guessing.”
130	Lists Rebecca, Wirfs-Brock as author of DOOS instead of Rebecca Wirfs-Brock
132	Next to last paragraph, middle “essentially introduces a new sub design problem” should be changes to read “essentially introduces a new subproblem.”
142	In the blue example “MoneyMarket Accounts” should be one word “MoneyMarketAccounts”
143	Caption on figure 4-4 should read “A BankAccountBean is the sum of its one primary and multiple secondary roles.”
144	Add “may” to first sentence: Objects may know and do similar things, but because they do them differently, they may require different interfaces and implementations.
163	Second bullet misspelled “responsibilities”
176	<p>Run on sentence in first paragraph of Simulation Collaboriatons. It reads “It helps you to fin new objects (you will likely have to invent new objects for controlling the flow of work or responding to events), to discard ill-conceived objects, and to elaborate any vague responsibilities, and it results in responsibilities shifting from one object to another.”</p> <p>Break it in two: “It helps... and to elaborate any vague responsibilities. As a result, responsibilities often shift from one object to another.”</p>
178	Just before “Set the Boundaries”. delete the for in “for which details are best left out.”

191	Second callout should read: “Leaving things in an inconsistent state shifts the burden to clients to figure out how to recover—something they may not always be equipped to do.”
192	“which relations between collaborators should be static and which should be fixed” should be changed to “which relations between collaborators should be dynamic and which should be fixed”
194	Our URL should read like others on the page, including “http://www.wirfs-brock.com”
195	Misspelling of Douglas Hofstadter at the very bottom of the page
205	In Figure 6-5 both places where “coordinators” is mentioned should be replaced with “controllers”.
221	In callout, remove parentheses and rephrase: When you discover a new role, create a CRC card for it. Note on each candidate’s card that plays this role the fact that they do.
264	Book title should be <i>The Elements of Style</i> . Missing “of”.
301	Delete first two sentences from callout. Make it start with the sentence that starts: “Concentrate on who should be responsible...”
306	Figure 8-7 caption should read “UserSession takes one of two branches.”
308	Remove callout. It is identical to the callout on p. 156.
324	Callout. Remove the phrase “ are informal tools for capturing” and replace with “capture”.

Questions and clarifications:

Rock, Paper, Scissors Example on pages 20-24. A reader, after pointing out an error in the errata for the Rock, Paper, Scissors example (the correction on page 21 should refer to figure 1-8) asks:

I don’t understand the purpose for three added methods, beatsRock(), beatsPaper(), and beatsScissor(). Since a rock only calls beatsRock() and a paper only calls beatsPaper(), then why do we need three methods in this interface? Why not just have a beatsMe() method that is implemented for the particular class being defined?

The example in the book is not a practical demonstration of how to put double dispatching to work, but a rather contrived example. With both coding errors combined with diagramming errors, it makes it even more difficult to understand. If I could, I would yank it from the book and replace it with a more realistic example that isn’t so contrived. For one thing, this doesn’t report “ties” where a rock is asked if it beats a rock. (There really should be a “yes” “no” and “tie” answer possible.) But in spite of that, there are still lessons to be learned from this example.

The idea behind double dispatching is to make a decision by sending specific messages to an object passed in as an argument in lieu of writing

any if-then-else-if or case statements in the code. In this case for rocks, papers, and scissors, we want to make a decision based on two objects. We start the ball rolling by asking the first object "does it beat the second?" (which is passed in as the argument to beats()). To avoid any conditional checks, we want to turn around and negate the answer to the question beatsX?, where X is the specific type of object we sent the original message to.

If we implement 3 classes for the rocks, papers, scissors game, each that implements beats(), beatsRock(), beatsPaper, and beatsScissor() we can accomplish this.

To illustrate, let me work through the case where we want to try to answer the question "does a rock beat a scissors?"

A rock should answer "no" to the question "do you beat a rock?", "yes" to the question "do you beat a scissors?", and "no" to the question, "do you beat paper?"

To accomplish this, the beats method for rock turns around and asks the specific question "do you beat a rock" to whatever the argument passed in to its beat method, and then negates that answer and returns that to the game coordinator (if a scissors doesn't beat a rock, and returns false, by negating that answer and returning it to the game coordinator, the rock gets this question answered without having to do any checking on type. Sure, we could have done a case statement and had one method for each object, beats, but that wouldn't have illustrated the concept of double dispatching where there are no case statements or conditionals and still questions are being answered.)

The real payoff of double dispatching isn't in some silly game example like this, but when you want to make changes to code (add more cases that can be handled, without having to modify existing code that works). The example on pages 175-177 in the book is a more realistic example of the concept, but even it is overly simplified.