

# *JavaMBS Role Play Information*

## **Introduction**

When we were first confronted with the C++ version of the Marine Biology Case Study (MBCS), we debated how we could present this material to students and teachers. After some thought, we developed a scripted role-playing exercise. Since then, many other people have used this exercise, added ideas, and suggested improvements. We have now developed a parallel exercise for use with the Java Marine Biology Simulation (JavaMBS). We hope that you may find it useful.

We believe that one of the best ways to emphasize the interobject communication and inheritance of the APCS Java Marine Biology Simulation is through the use of scripted role playing. The “scripts” for the roles are included in a companion document. This document discusses some tips about how to use those scripts most effectively in the classroom. The most recent copies of both of these documents can be found at <http://web.sbu.edu/cs/dlevine/RolePlay/roleplay.html>.

Some of the advice contained in this document is self-contradictory. This is because different teachers who have used this exercise have different, and sometimes contradictory, ideas about how to make it work best. We have included the vast majority of the suggestions given to us. Read all of the advice and choose those points that seem most relevant to you.

Instructors are welcome, encouraged even, to use these scripts to help students and are requested to send feedback (positive, negative, suggestions for changes, etc.) to [dlevine@cs.sbu.edu](mailto:dlevine@cs.sbu.edu). There are no restrictions on the distribution of these documents except that authorship credit must be given where it is due.

## **How the Role Play Is a Compromise**

JavaMBS comprises many classes. Some are part of the Java runtime library and others are developed just for this case study. Some of the latter will be tested on the AP exam while others are “Black Box”; the functioning of the black box classes is of minimal concern to us. In particular, this role play concentrates on the Fish, BoundedEnv, Simulation, EnvDisplay, and “main” classes. If desired, roles can be added for the DarterFish and SlowFish classes, as well as for random number generation.

The primary goal of the exercise is to expose students to the idea of intercommunicating objects in general, and to the classes of JavaMBS in particular. The scripts were designed to be usable in a single class period (see notes below about timing.) We have attempted to ensure that interactions that occur during the role play accurately reflect the intercommunication of objects within JavaMBS, but in the interests of time, we have not included every interaction. (For example, most interactions involving error checking in the program are not part of the role play.) The role play strives to “tell the truth, but not the whole truth”.

## Timing

“Past performance is no guarantee of future results.” Everyone who has done this has had a different group of people doing this under different circumstances. Depending upon your students, upon the mechanisms you use for sharing data as they act out roles, and (most importantly) upon the amount of external commentary during the activity itself, the exercise may take as little as forty-five minutes or as much as two hours. At one workshop with AP teachers, we were able to run two groups of teachers (who had not previously seen the exercise) through the exercise in about fifty minutes.

(Side note: we are particularly interested in learning how long it takes others to go through the exercise. Send comments to [dlevine@cs.sbu.edu](mailto:dlevine@cs.sbu.edu).)

## Instructions for the Cast

One of the areas where participants have the most difficulty is in deciding what to say. It is very important (see debriefing) that individual actors are addressed by their (personal) names and not their class names. We emphasize this during previous scripted role playing exercises, but anything that can be done in advance to “train” the actors will make the entire exercise go more smoothly. If the instructor has pre-cast the roles, the scripts can even be customized to help prevent this problem. (We have never done this ourselves, but have recently heard of others doing so with some success.)

In a similar vein, it is important that the actors speak all of the lines assigned to them AND that they not speak “stage instructions.” Reminding them to read their entire scripts before performing is important. On some occasions, a clarifying sentence follows an action. Participants who have not read the entire script tend to make the very mistake that the clarifying sentence is designed to prevent.

It may help to remind the cast that

- words in boldface represent requests that someone perform a task
- underlined phrases enclosed in angle brackets represent ideas that should be applied, e.g. <your name>
- indentation matters; thus all sub-instructions are to be done provided the “if” condition is met – and none of them should be done if the condition is not met

If the SlowFish/DarterFish version of the role play is being done, actors need to know how to choose among (potentially conflicting) sets of instructions in the scripts. The rule is simple, and intuitive to most folks, but should be explicitly stated: “Unless otherwise instructed, always use the FIRST set of instructions – starting on the top page – that is appropriately named; if otherwise instructed, then do what those instructions say.” (During the exercise, this should be carefully monitored.)

## Mechanics – Advance Prep

As part of the role play, the EnvDisplay will need to draw fish in various colors. If your set of color implements does not include red, green, and blue, then the script for

main should be altered so that Fish objects are given colors that can indeed be displayed. (Alternatively, one can draw a fish outline and write the word “red” inside the body to indicate a red Fish.)

Before running through the role play, one needs to assign the roles to appropriate students – see below regarding casting.

If the SlowFish/DarterFish version is being used, then the scripts for those roles are created by stapling (or otherwise attaching in a semi-permanent manner) the respective SlowFish/DarterFish scripts on top of a Fish script. (See note below on inheritance.)

Assigning the roles the day before the role-play activity and having the students read over their assigned class code helps move things along. Some teachers quiz the students on the scripts in advance; they can then assign the roles randomly (among those students who pass!)

The “Cast of Characters” should be publicly displayed (on the blackboard?) in an easy-to-read format, e.g.

- BoundedEnv – Happy
- EnvDisplay – Doc
- Simulation – Sleepy
- main program – Grumpy
- first Fish – Sneazy
- second Fish – Bashful
- third Fish – Dopey

If the students don’t know each other well or if visibility is a problem, it can also be helpful to preprint the “Cast of Characters” and distribute these to all students. (Some people have found this list to be helpful even after the completion of the role playing exercise.) Nametags with LARGE labels, e.g. “Sleepy/Simulation” can help. Some people have used large sections of poster board for this purpose.

It will also be necessary to reserve a special part of the blackboard (or projection device) for the EnvDisplay.

Many instructors assign roles and have the participants study their scripts in advance (e.g. the night before the exercise). In some situation, e.g., workshops, this may not be feasible. Regardless each student should have read his/her entire script before the play begins and should realize that s/he will perform only one task at a time. Be sure to clarify the “method of return” for the **allObjects** and **getNeighbors** methods of the BoundedEnv class if the script has not already been modified accordingly. (See “A Note on Returning Lists” below for suggestions.) If the roles have been assigned in advance – as mentioned above – this step is probably not necessary, but in a more ad hoc format – such as a workshop presentation – it is worth taking some time just before beginning to clear these matters up.

The person playing the role of the main method is instructed to begin.

In a time-tight situation, the facilitator should intervene as little as possible, generally doing so only to correct errors or to minimize duplicated work.

(Extra hint: It can make the facilitator's life a lot easier if s/he has a copy of each script!)

Note (6/26/03): In addition, several teachers have suggested an "answer key" that shows who says what and when. Such a document is in preparation.

## **A Note on Method Invocation**

As noted above, one of the more difficult tasks for some students is figuring out how to make a request of another object. This must be clear to all participants before the role play begins. One way to accomplish this is to have run through a simpler role playing exercise before beginning this one. (See <http://web.sbu.edu/cs/dlevine/RolePlay/roleplay.html> for an example of a simple exercise that can be done as early as the first day of class that accomplishes this – and a good bit more.) If your students have no such experience, it is worthwhile to take some time to "teach" them your message passing protocol. Keep it simple. For example, consider the cast above; if the main program wants the EnvDisplay to **show** the data, then an appropriate statement for Grumpy to make would be, "Doc, please SHOW." When a method is complete, the actor will either state a result (for a non-void method) or make a statement that obviously terminates the work (for a void method.) Note that the caller's name is not used as part of this statement; the caller is usually anonymous – though known to everyone in the room – and presumed to be paying attention.

## **A Note on Private Data**

In an ideal world, only the actor playing the role of a given object would have access to that object's private data. While this would be the most faithful representation of the code, it may not be the best pedagogical use of the role play. Non-participants (or even non-active participants, such as the actor playing main) often learn more if they can watch and monitor such bookkeeping. Some of the various options for dealing with private data include:

- Keeping the data totally private (the "faithful" solution). Each actor writes the data on a sheet of paper in front of her/himself – or even keeps it in her/his head.
- If one is using large (8½ by 11) name tags, the actor can write the data on the back of the sheet. It can be private, but the audience (or the facilitator) can ask to see it at any point in time. In this way, the privacy is emphasized, but the engagement can be maintained.
- The actor, a confederate/understudy, or the facilitator can draw the data on the blackboard, flip charts, or in some other public forum. This does not emphasize

- the privacy as well, but physical separation and facilitator “guidance” can reinforce this.
- Depending upon the learning outcomes desired, it may be beneficial to have the BoundedEnv show its data in a public manner. If this is done, care must be taken to ensure that the EnvDisplay does NOT ever directly view/use the BoundedEnv’s private data. In other situations, it may be better to have audience members maintain their own copies. (Keep in mind that one aspect of the MBS program is that the display does not always represent the environment; rather it is “synced up” at various times, most often at the end of a step() operation.)

## A Note on Returning Lists

Two methods in the Fish class return lists. Returning lists of arbitrary length can be problematic, but in the context of the role play, the lists will be of size 4 or smaller. These lists can be returned orally, or via a piece of paper. It is often helpful if the facilitator at least writes down the list (temporarily?) on the blackboard to ensure that everyone (including the Fish) hears and understands what was passed. Regardless of what is chosen, it is necessary for the facilitator to tell the BoundedEnv actor what is desired.

## Mechanics – As the Action Unfolds

There are several points to consider/emphasize regarding the “acting on stage”:

- Objects must be addressed by their names, not by their roles, e.g. “Happy, can I please have allObjects?”, not “BoundedEnv, can I please have allObjects?” [Note that Happy is the actor’s name, i.e., the name of the object. BoundedEnv is the role, i.e., the name of the class.] Although the protocol for this should have been worked out in advance, this is one of the most common errors made by participants and the facilitator needs to show constant vigilance.

[Editor’s notes: (7/2001) this error was unfortunately common among student answers on the 2001 APCS exam – and not just on the case study question! It is my opinion that this error has become more common over the last two years, perhaps due to the greater number of questions involving interacting classes. (6/2003) Having read the 2002 and the 2003 exams, I sadly note that the problem is still prevalent, especially among students taking the “A” exam.]

- There are three Fish and their names are what is stored in the BoundedEnv, e.g. Sneezzy, Bashful, and Doc – or whoever is playing those roles, not their locations.

- Some people like to indicate flow of control through the use of an artifact, e.g., a nerf ball held by the object currently executing, or a jester's cap, etc. If this is done, then one actor should pass the artifact to another as "control" is passed via a message. Note that furniture configurations can play a role in the success of this idea! [On the negative side, some people don't care for this as it isn't always clear where the artifact should go at the completion of a task.]
- Some people like to maintain a call stack of some sort. One way to do this is to have the actor playing the main method start holding the end of a strand of a ball of yarn. Whenever a message is passed, the source object grabs the near end of the ball, and then passes the ball to the object being called, unrolling the ball along the way. When a task completes, the ball is returned (and rewound) as it is passed back to the caller. Note that if an object appears several times on the call stack, then its actor will need to hold several points along the string of yarn.
- Since the role is so "active", some people like to have the BoundedEnv in front of the group.
- (Unauthorized) shortcutting is NOT allowed. After a bit, some actors/objects will try to bypass intermediaries "for efficiency." This should not be allowed to happen. There is a great "teaching opportunity" here regarding efficiency of the code.
- Student actors will make errors. It is important to correct them politely, but firmly when this does happen.

## Mechanics – Ending

Currently, the role play suggests that it will simulate twenty-five steps. In practice, most people halt the simulation after one iteration of that loop. Sometimes, people go through a second pass but with significant short-cutting. (Hey, not everyone practices what they preach!) The ending is relatively unimportant, though *it is essential* to "debrief" the experience soon after, no later than the next class.

## Debriefing

During the debriefing, it should be emphasized that the role play is a simplification. Some of the simplifications of the role play can lead to student misunderstandings later. These can be avoided if you are alert to them. Things to discuss during the debriefing:

- Importance of name of object vs. name of class – particularly for Fish (i.e. we have only one Fish class, but three – or more – Fish objects.)
- Behaviors of some methods have been considerably simplified in certain cases, notably to remove various kinds of bounds checking.

- In the role play, Fish are constructed using the four-parameter constructor. In the simple version of the case study, the two-parameter constructor is used. This change was made to prevent the fish from needing to determine random directions and colors. The random number generator should only return integers; thus, it should not be asked for directions or colors.
- The Direction class (black box for testing purposes) is subsumed by the Fish class and “common sense,” i.e., Fish are expected to be able to determine the direction behind themselves.
- In the role play, Fish do know their own location and direction. In the real MBCS, they have a Location object and a Direction object to care of this. This is one of the key simplifications of the role play.
- In a similar vein, some of the private methods of the Java MBS program (e.g., the move() and emptyNeighbors() methods in the Fish class) have been “folded in” to other tasks within the context of the role play.
- Only one “copy” of a given fish exists. Thus changes to a Fish are immediately reflected in the Environment. In a similar vein, it should be clear that although each Fish “has” an Environment, in fact, each Fish really has a reference to the same Environment. By contrast, the Display contains “pictures”, not Fish objects. Thus, changes to a Fish or the Environment are not immediately reflected in the Display.
- [*Only for those familiar with the C++ version of the case study*] The C++ version of the case study involved having many copies of a given instance of Fish; as a result, the Environment needed to be Update()ed periodically. As the preceding bullet noted, this is not the case with Java. The C++ role play “hid” the fact that many copies of the same fish existed. This set of scripts is more accurate in that respect.
- It is ambiguous whether or not the EnvDisplay erases any or all of the display prior to a showEnv. If a particular behavior is desired, it can easily be added to the script. Otherwise, a discussion about multiple implementations of display (blackboard vs. screen vs. ???) is probably a good idea. Regardless of what one desires some sort of clear screen is in order. This might be accomplished by erasing the board or by simply drawing the new grid somewhere else. If the EnvDisplay is using transparencies and an overhead projector then the transparency may be wiped clean – or another one might be used. (Note that if transparencies are used, the EnvDisplay can “cheat” and draw some grids in advance.)

Note: If each student has a copy of the “Cast of Characters”, the discussion often goes more smoothly during the debriefing.

## Casting

Who is chosen to play which role can have a major effect on the time needed to complete the exercise. Some tips are below:

- main – This role is quite easy and can be played by any student who will stay on task.
- Simulation – Another quite easy role.
- RandomNumberGenerator – Again, an easy role, though not one for someone who thinks too much when asked for a random number. With a small number of participants, this is a good role for the facilitator to assume especially since the facilitator will then be able to control Fish movements.
- EnvDisplay – This role is somewhat tricky and should be played by a student who is relatively mentally agile. It is not often “on duty”, though. When casting this role, keep in mind that it is the only role that cannot be played from a student’s chair.
- BoundedEnv – The “starring” role; it is best played by a student who can think on his/her feet (either with or without preparation as circumstances dictate) and will “stick with it” for an hour.
- Fish – The first Fish to be processed will need to be played by a strong student, but the other two Fish can be played by less bright students who can learn from what the first student does.

It may be best to cast by trying to find three bright students to play the roles of EnvDisplay, BoundedEnv, and first Fish. Don’t worry too much about the rest of the casting. If the class size is small, the facilitator can assume the roles of the random number generator and the main program (in that order). If the class is too large, then pairs of students can be assigned to a given role – an actor and an understudy!

## Using Inheritance

The JavaMBS Case Study introduces inheritance in Chapter 3. Inheritance is seen through the existence of two new classes, DarterFish and SlowFish. The scripts document contains scripts for these two classes as well as a second script for main that uses these two Fish. The main script can be used unchanged. However, to properly implement inheritance, the DarterFish script must be stapled *on top of* a Fish script to build the complete DarterFish script. (Likewise, the SlowFish script should be stapled *on top of* a Fish script.) The actors playing these roles should be instructed that when a request is made of them, they always start on the first page and look for instructions, continuing forward through the script until they find the task in question. Of course, if specific instructions dictate otherwise, then those instructions should be followed.

[Note: the instructions sometimes dictate that an actor should look “on a subsequent page”. These “calls” represent execution of a line of code that begins with “super.” By contrast, implicit turning of the pages represents cases where the subclass simply did not reimplement the method found in the super class.]

When running the role play with SlowFish, the facilitator needs to be aware of an issue involving probability. It is probably best to show SlowFish moving sometimes and not moving at others. Unfortunately (for purposes of the role play) the SlowFish have a relatively low chance of moving. The facilitator may wish to “coach” the random number generator to ensure that both behaviors are exhibited.

## **Acknowledgements**

Over time, the percentage of the two documents that we (Steve and David) have written has declined significantly. We are very grateful to the large number of people who have encouraged us with this work, and we are even more grateful to the people who have sent us suggestions about improving the role play or this document. In particular, we would like to thank Fran Trees, Sharon Lee, Kathy Larson, and Alyce Brady for their detailed reviews. We owe credit to several others as well and promise to include all the names as we complete this round of rewriting.

## **A Final Note**

While we developed this exercise for ourselves (and our students), we hope that you find it useful, too. All we ask is that if you use it, you give us feedback on it.

The role play itself has been tested only a few times. As of 6/25/03, these notes have been field tested about a dozen times. If you find errors of omission (or of commission) within either of them, please send corrections, comments, etc. to [dlevine@cs.sbu.edu](mailto:dlevine@cs.sbu.edu).

We have had great success with this exercise as have several high school teachers that we know of. We hope that you do, too.

--david levine and steve andrianoff, st. bonaventure university, 8/2/02; revised 7/11/03

***P.S. Links to all of the role play documents can be found  
at***

***<http://web.sbu.edu/cs/dlevine/RolePlay/roleplay.html>***