Laboratory 4: Baby, Crab, and Cone Puzzle

Objective. To use booleans, conditional statements, and loops to help solve a classic puzzle.

Discussion. In this lab, we develop a program to validate potential solutions to a classic puzzle[1] recast as Mama's puzzle of the baby, crab, and cone. As the story goes, Mama headed off to the ice cream shop on a hot summer day with her delightful baby and pet crab. On the way she crossed a river with the aid of a ferry boat that, rowed by herself, could carry one other. A little thought demonstrates that this was no minor obstacle—it required three trips in the ferry to get everyone across (the crab it seems, was no great swimmer). By the time she got to the shop, she was clearly ready to make a purchase.

Having purchased the largest cone possible, Mama set back toward home with all in tow: baby, crab, and cone. At the river, however, she faced a dilemma: in what order should she row everyone across in the ferry? The difficulty was that since only one of the baby, crab, or cone could be ferried across (it must have been a *huge* cone), two must be left behind. Unfortunately, if the baby was left unattended with the crab, violence of some form would surely ensue. If the baby was left with the cone, it would overindulge. Thus, each trip of the ferry had to be made with care, making sure that the baby was never left with either of the other two on same shore.

To facilitate Mama's journey, you are to write a planning program of sorts, that allows the harmless simulation of the various possibilities. We have in mind a program with output similar to the following:

```
Welcome to the Puzzle of the Baby, the Crab, and the Cone.

One hot summer day, Mama crossed the river to the left
bank take a break from programming and buy an ice cream.
She brought her baby and her pet crab.  On the return
she came to the river and was faced with a dilemma: how
could they all cross without disaster?  The ferry holds
Mama and just one other item.  The problem is that
  * if the baby is left with the cone, it spoils its dinner.
  * if the baby is left with the crab, the crab bites the baby.
Please help Mama make her journey across the river!

Mama is on the left.
The baby is on the left bank.
The crab is on the left bank.
The cone is on the left bank.
Who would you like Mama to cross with?
(type 0 for no one, 1 for baby, 2 for crab, or 3 for cone)
2
```

[1] The farmer's puzzle of the chicken, fox, and the corn.

```
Mama crosses the river with her pet crab.
Oh no! The baby ate the ice cream.
We'll have to stop and help Mama and her family.
Undoubtedly, she'll need your help again.
```

As you can tell, the program keeps track of the location of each of the parties, perhaps by a boolean. After describing the state of the situation, the facilitator is asked to determine who rides in the ferry with Mama. As Mama leaves shore, the various lethal conditions are checked for and the simulation is potentially terminated. If, however, everyone eventually makes it to the destination (here, the "right bank"), the puzzle is solved. A good program would point out that the solution might have been faster if more than seven moves were required.

Procedure. This is a complex program, so the following steps should be considered in the *design* of your program. Once designed, the program is more easily coded:

1. Identify all the variables that will be needed. Variables help to keep track of the state of the program. If something must be remembered, it must be accounted for as a variable. For example, it might be useful to keep track of the number of times the ferry has crossed the river.

2. For each variable, identify its type, and if necessary, its initial value. This should complete your declarations.

3. Break the program down into several sizeable pieces. It's not important to know immediately how each piece is implemented, but it is necessary to identify logically distinct portions of the program. For example, the instructions must be written at the beginning, the results at the end, and in the middle it is necessary to print the location of all the participants. There may be other parts, as well.

4. Identify those parts of the program that are part of a conditional or looping construct, remembering the features that distinguish each of the loop types.

5. Our informal design is completed if a reasonable person can follow the logical states of the program without doing anything that seems contrary to the imagined execution of the program.

6. Armed with a design, it is now possible to implement each of the logical components in Java. It is often the case that the design had to be augmented to take into account the finer details. As you implement each of the logical components of the program, you may find it useful to test to see if the program works as expected. For example, you run the program to see if the instructions and initial state is printed correctly. The intermediate steps also help to ward off any unforeseen problems as early as possible.

7. Once completed, test your program thoroughly. Does it work in short simulations (as seen in the one above)? Does it work when you follow the optimal solution? Does it work, even if Mama makes far too many trips? What happens if the number 4 is provided as input to the question?

8. When completed, sit back, take pride in your work, and save your program.

Thought questions. Consider the following questions as you complete the lab:

1. In your finished program, are there any unnecessary variables?

2. Are there any statements that don't get executed?

3. Return to this code a couple of days later. Are there improvements you can see after this fresh view of your code?

4. If crabs liked ice cream (they don't), could the puzzle be solved? (Assume Mama eats neither the crab nor the cone in transit.) If so, how? If not, why?

5. Suppose there were four players (include, say, a cat). What is the simplest set of rules that makes Mama cross the river the most times before success? (It must be solvable.)

6. How might you implement this program graphically?