Exercise 1: Template Method and Strategy

Download today's Ruby application and play around with it. It models a game where an oracle thinks of a number in a certain range and each participant makes a number of guesses to find out what the number is. After each guess, the oracle tells the participant if the are correct or not, and if not, whether their guess was too big to too small.

The `Participant` class has 4 play methods representing 4 different strategies: random guessing, linear search, smarter random guessing and binary search.

What's the problem?

Study the code of the `Participant` class. This is the one you are concerned with. Make notes on what problems you see and write this up fully in your journal at a later stage.

Some of the problems are immediately visible in the code as it stands. Others become apparent only when we're asked to update the code. Consider a new requirement that a player be able to switch their playing strategy easily at runtime, without clients knowing.

What's the solution?

You can guess that there are two patterns to be applied here! The motivation for each pattern is completely different, so make sure you clearly grasp why each one is worth applying (if you don't agree, by all means make your case in the journal). These two patterns don't normally go together, usually one is used without the other, but the sample program is one where both can sensibly be applied.

What order should you apply them in? Think about this before you start.

Notes

Feel free to massage the code the make the patterns fit better. Make your solution as DRY as possible of course. The test cases should help keep you on track throughout.

Patterns are tailored to fit the context they fit into. Note the design decisions you make in applying the patterns and write these up in your journal.

Identify the abstract, final and hook methods in your template method. How do you prefer to implement the abstract methods? How do you highlight the hook methods to programmers writing new subclasses?

Leave `@max_num_attempts` and `@num_attempts` in the `Participant` class. Strategies often have to read and write instance variables in their 'context' class, as it's their natural home in a sense.

When you've applied the patterns, try adding a new strategy (guess from `upper` down to `lower` if you can't think of anything smarter)

Tidy up the code in main.rb. Do the patterns you've applied help you in doing this?