Kent Beck's rules of Simple Design

The code must first be correct (as defined by tests);

- then it should be a clear statement of the design (what J.B.Rainsberger calls "no bad names");
- then it should contain no duplication (of text, of ideas, or of responsibility);
- and finally it must be the smallest code that meets all of the above.

It's time to stop refactoring when the code makes a reasonable stab at meeting those goals, and when any further changes would add no further benefit."

Keep it simple	Make changes simple, without over architecting for what maybe in the future.
Keep it DRY	Don't repeat yourself. Extract into methods if you find yourself repeating.
Make it expressive	If you find the need to comment, change method and variable names to make code express intent.
Reduce overall size of methods and classes.	Fewer lines of code makes it easier to understand. Introduce classes and methods with single responsibility.
Separation of concern	Divide the responsibility into modules that do one thing well. They should be easy to read, easy to test and easy to refactor.
Appropriate levels of abstraction	Think of software as having layers. Each layer exposes a clean interface that hides the complexity in the next layer.
Follow YAGNI	Avoid presumptions in your software. You Aren't Gonna Need It. Build the simplest thing that delivers the most value.
Girl-Boy-Scout Rule	Leave the code in a better state than you found it. Clean up messes, no matter who made it.

Refactoring Steps:

1. Characterization Tests - The Golden Rule
2. Red Green Refactor
3. Make a checklist
4. One thing at a time
5. Baby steps - small non breaking changes
6. Add additional test cases if needed
7. Check in green frequently
8. Review with pair

Credits:
Refactoring: Improving the design of existing code by Martin Fowler
Refactoring Applications using SOLID principles by Steve Smith
The Coding Dojo Handbook by Emily Bache : https://github.com/emilybache/Tennis-Refactoring-Kata