Decouple

- Take advantage of inheritance and polymorphism
    - A variable declared in the form of a parent class, assigned to any child class that inherits from this parent class does not affect the execution of the program.

SOLID:

- S: The Single Responsibility Principle (SRP)
    - A class should have only one reason to change
        - This means that one class should only have one responsibility.
    - Eg:
        - OnBindViewHolder has only one responsibility which is to display data and not make data formatting operations.
        - Making data formatting operations should move into the Util class.
- O: The Open-Closed Principle (OCP):
    - Software entities such as classes, functions, modules should be open for extension but not modification.
        - This means that if we are required to add a new feature to the project, it is good practice to not modify the existing code but rather write new code that will be used by the existing code.
    - Eg:
        - Every time a method is called, if-else conditions will be executed.
        - Created an interface to handle action. Created different classes for different behaviours.
- L: The Liskov Substitution Principle (LSP):
    - Child classes should never break the parent class' type definitions.
        - This means that a sub class should override the methods from a parent class that does not break the functionality of the parent class.
    - Eg:
        - We handle the individual logic inside the overridden methods in the different framgents. In the main implementation we should never handle logic.
- I: The Interface Segregation Principle (ISP):
    - The interface-segregation principle (ISP) states that no client should be forced to depend on methods it does not use.
        - This means that if an interface becomes too fat, then it should be split into smaller interfaces so that the client implementing the interface does not implement methods that are of no use to it.
- D: The Dependency Inversion Principle (DIP):
    - High-level modules should not depend on low-level modules. Both should depend on abstractions.
        - If you use a class insider another class, this class will be dependent of the class injected.
    - Abstractions should not depend upon details.

    - Details should depend upon abstractions.

Others:

- Use interfaces for describing relationship of classes
- Class should know as less as possible of the other classes it calls