Bismillahir Rahmanir Rahim

In the name of Allah, the most gracious the most merciful

Welcome to the '**ADDA**' on

Strategy Pattern

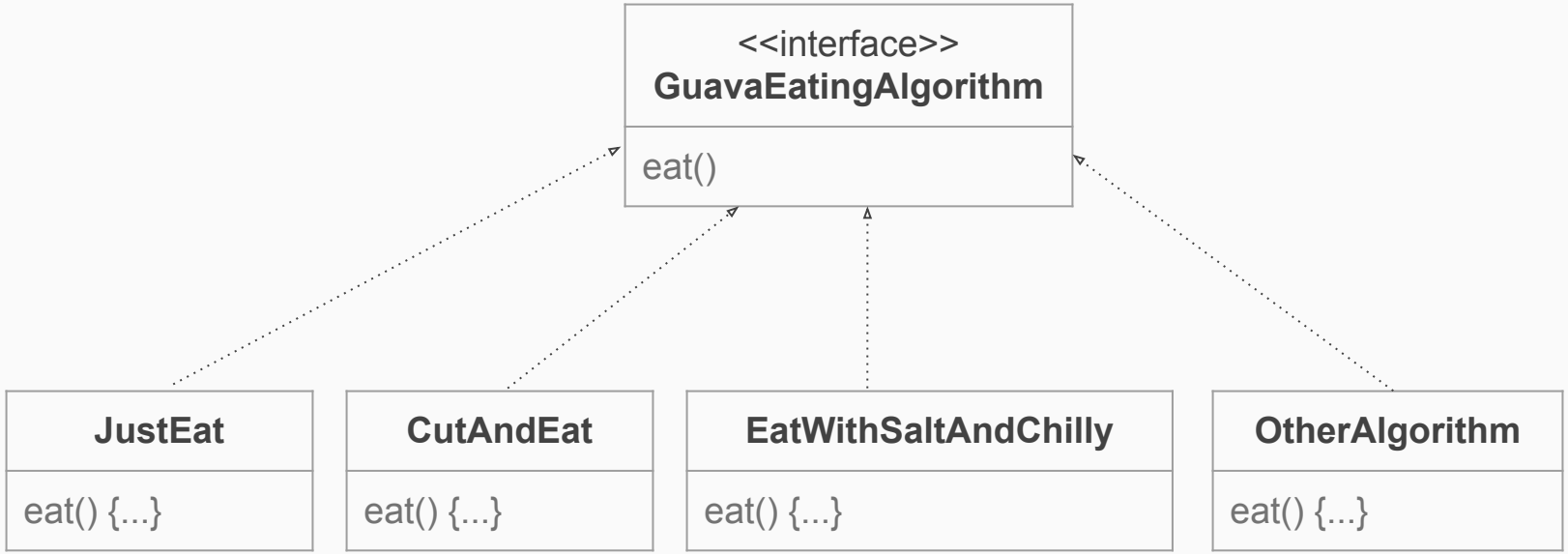
aka Policy Pattern

What do we do?

- ❏ Taking Breakfast
- ❏ Coding
- ❏ Doing Lunch
- ❏ Eating Fruits
- ❏ Drinking Coke
- ❏ Reading Newspapers
- ❏ ...and many more

Let’s Design The Class Diagram

GuavaEatingAlgorithm

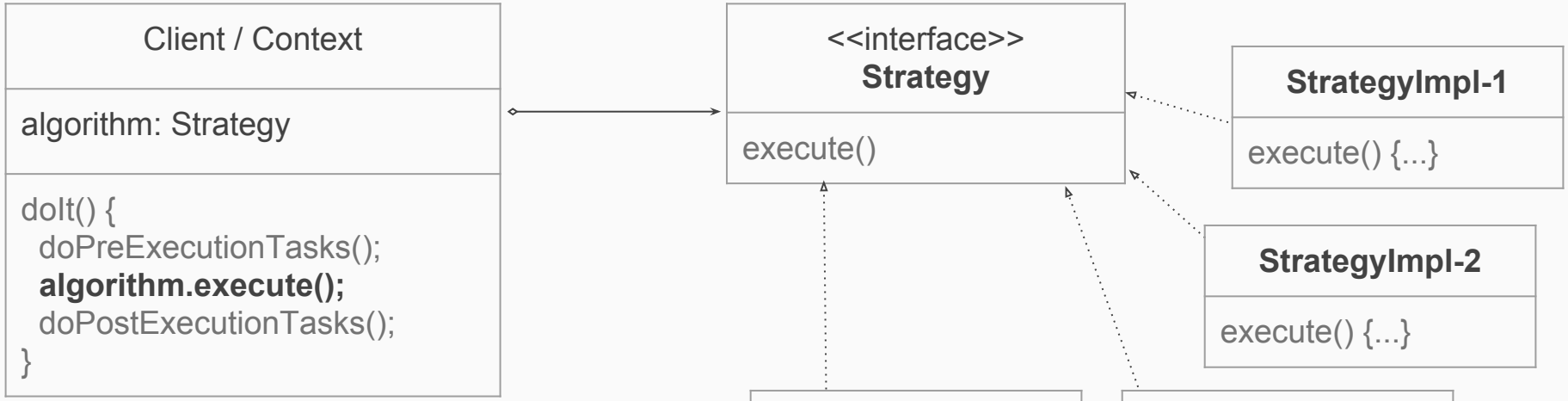


Employee Class: Revisited

Employee
guavaAlgorithm: GuavaEatingAlgorithm

takeBreakfast() {...}
doCode() {...}
eatGuava() {
  washGuava();
  guaveEatingAlgorithm.eat();
}
haveLunch() {...}
drinkCoke() {...}
...

Let’s understand the Strategy Design Pattern



The Strategy Pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable.

Strategy lets the algorithm vary independently from clients that use it.

Design principles honored

- ❏ Identify the aspects of your application that vary and separate them from what stays the same.
- ❏ Program to an interface, not an implementation.
- ❏ Favor composition over inheritance.

Use this pattern when -

- Many related class differ only in their behavior.
- We need different variants of an algorithm. For example, we might define algorithms reflecting different space/time trade-offs.
- An algorithm uses data that clients shouldn’t know about.
- A class defines many behaviors, and these appear as multiple conditional statements in its operations.

Consequences

- Families of related algorithms.
- An alternative of subclassing the client.
- Eliminate conditional statements.
- A choice of implementations. Client have option to trade-off in between.
- Clients must be aware of different Strategies.
- Communication overhead between Strategy and Client.
- Increased number of objects.

Real Life Use of Strategy Pattern

Let’s share our ideas