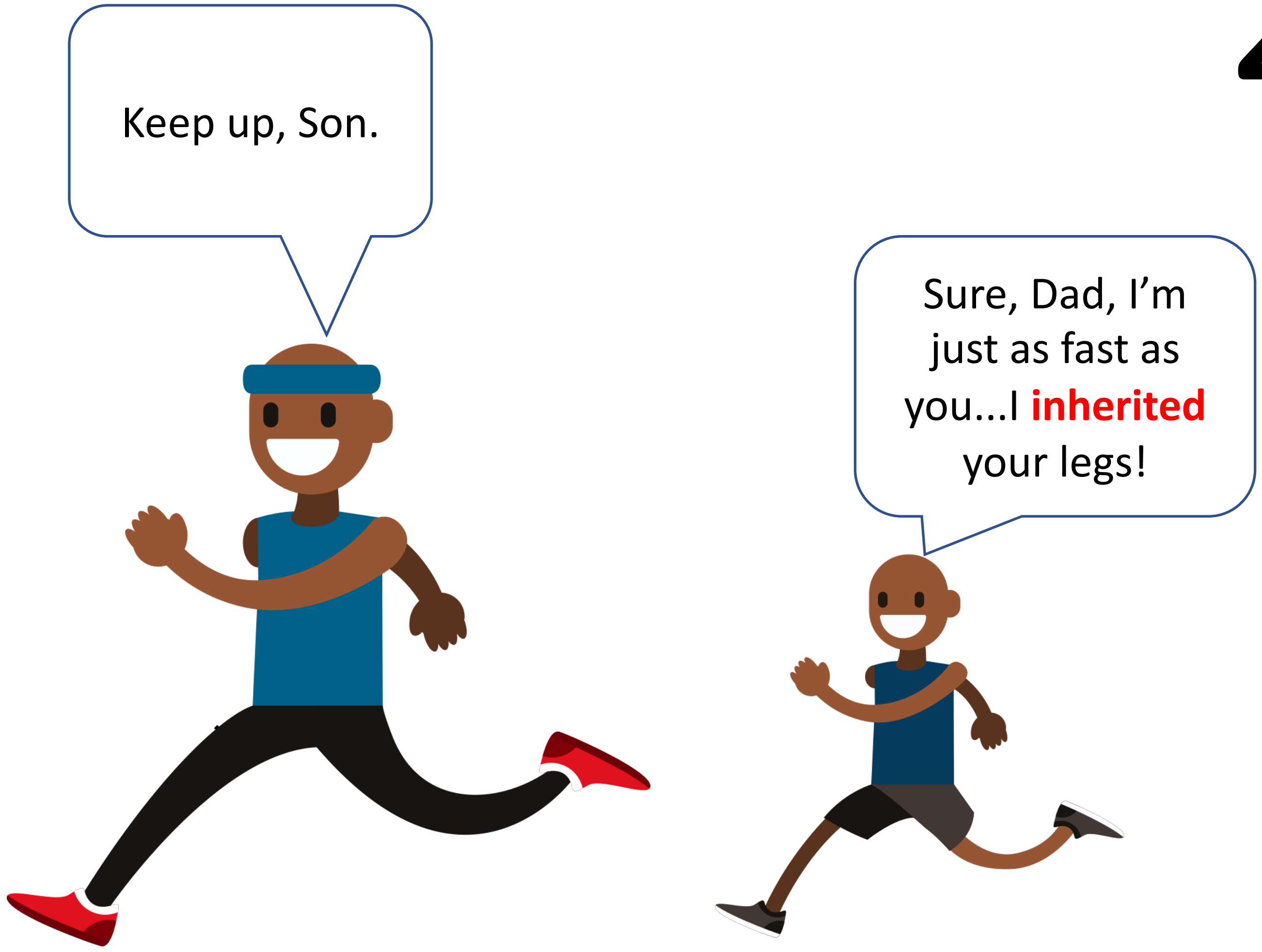
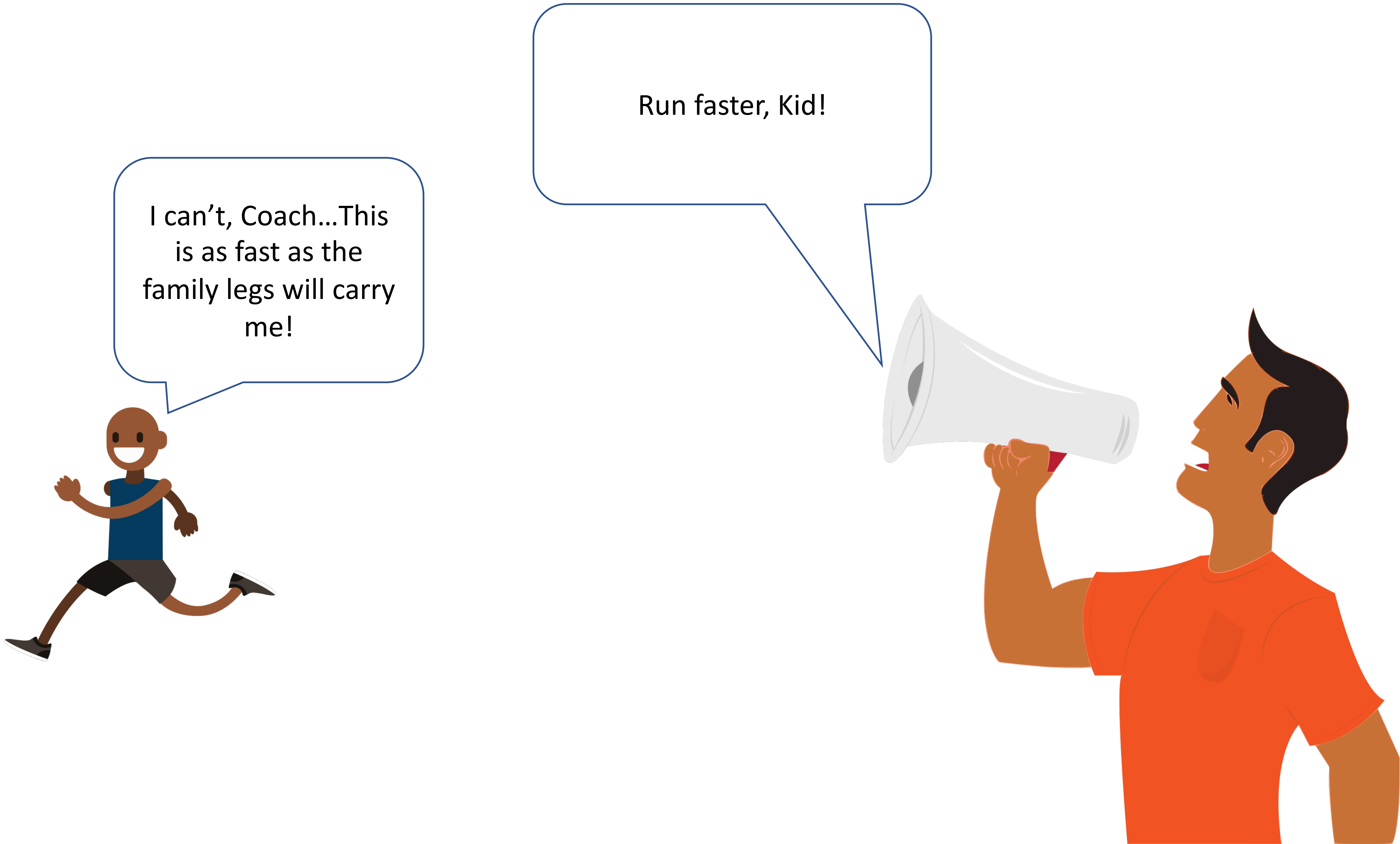
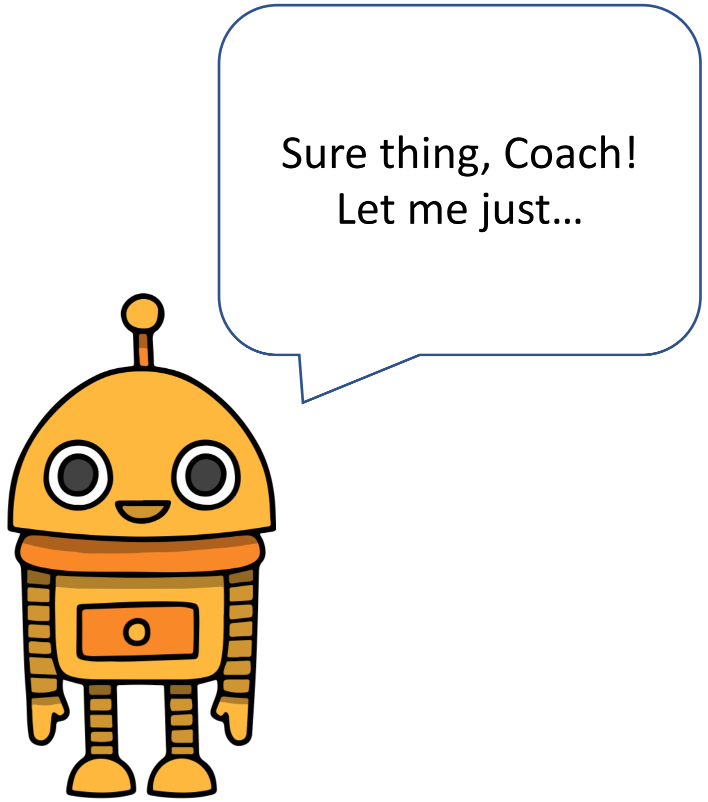
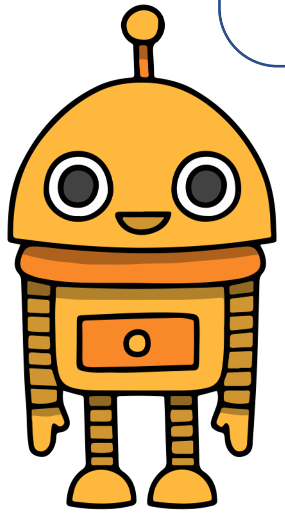
2018


Keep up, Son.
Sure, Dad, I'm just as fast as you...I **inherited** your legs!

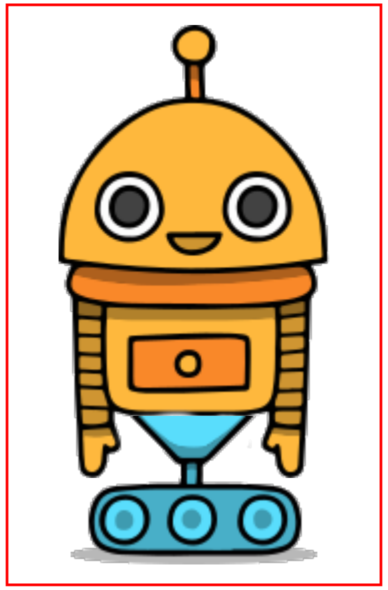

Run faster, Kid!
I can’t, Coach…This is as fast as the family legs will carry me!

2081

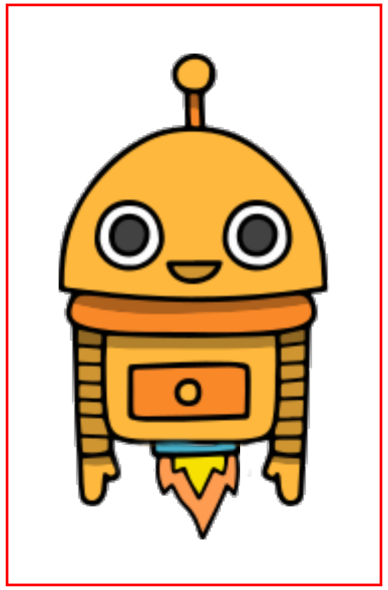


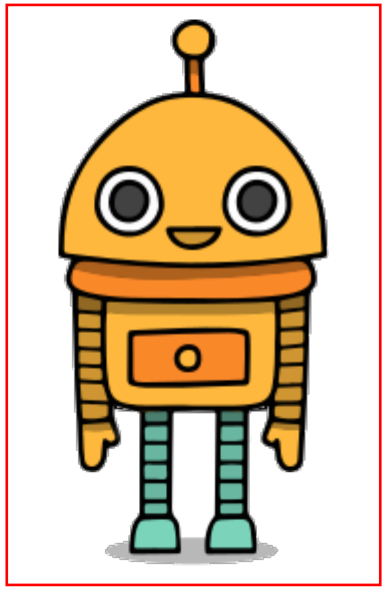


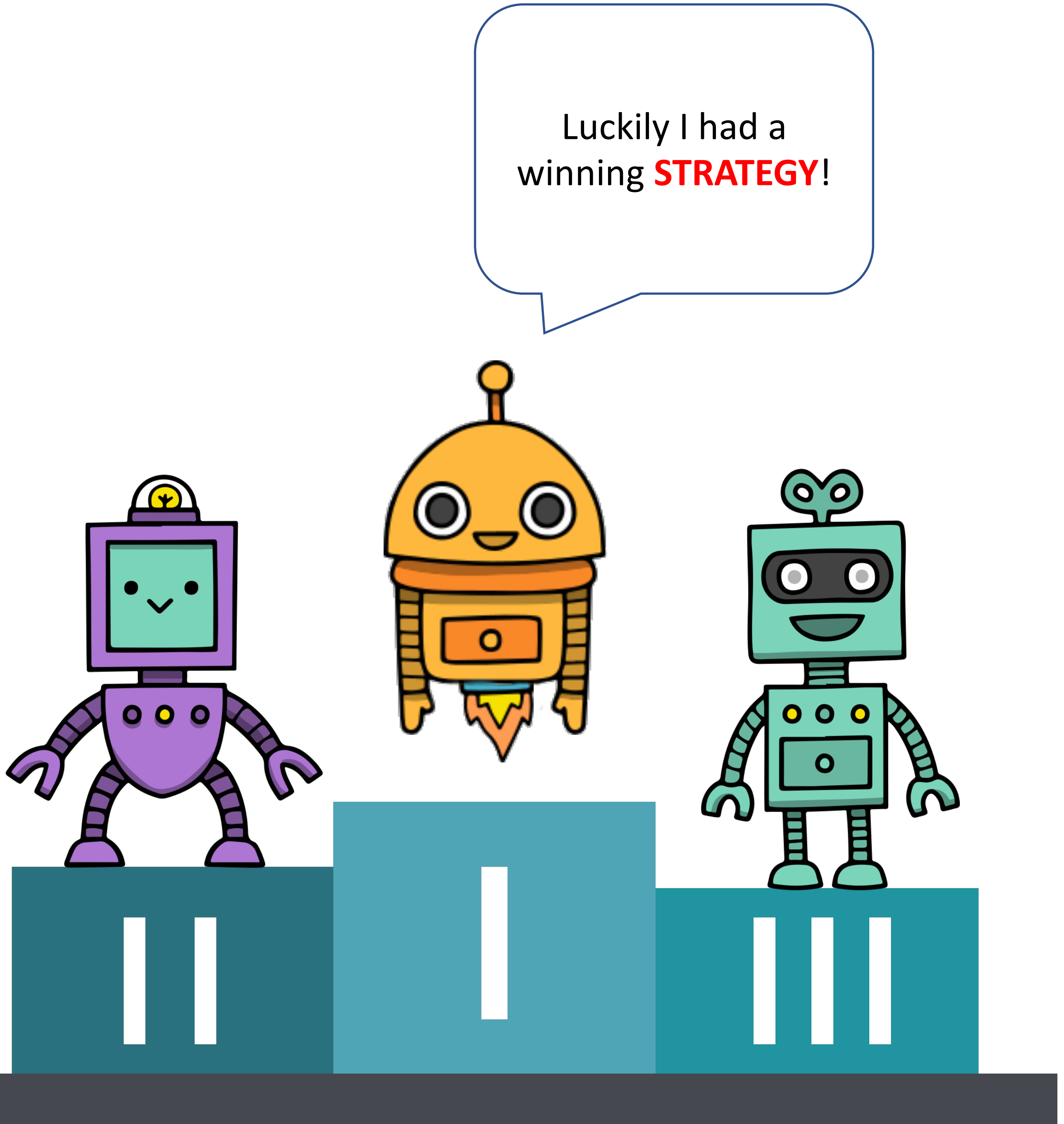
Swap in another pair of legs!



Choose your legs…






Luckily I had a winning **STRATEGY**!
II
I
III

Strategy Pattern

Theory

Definition

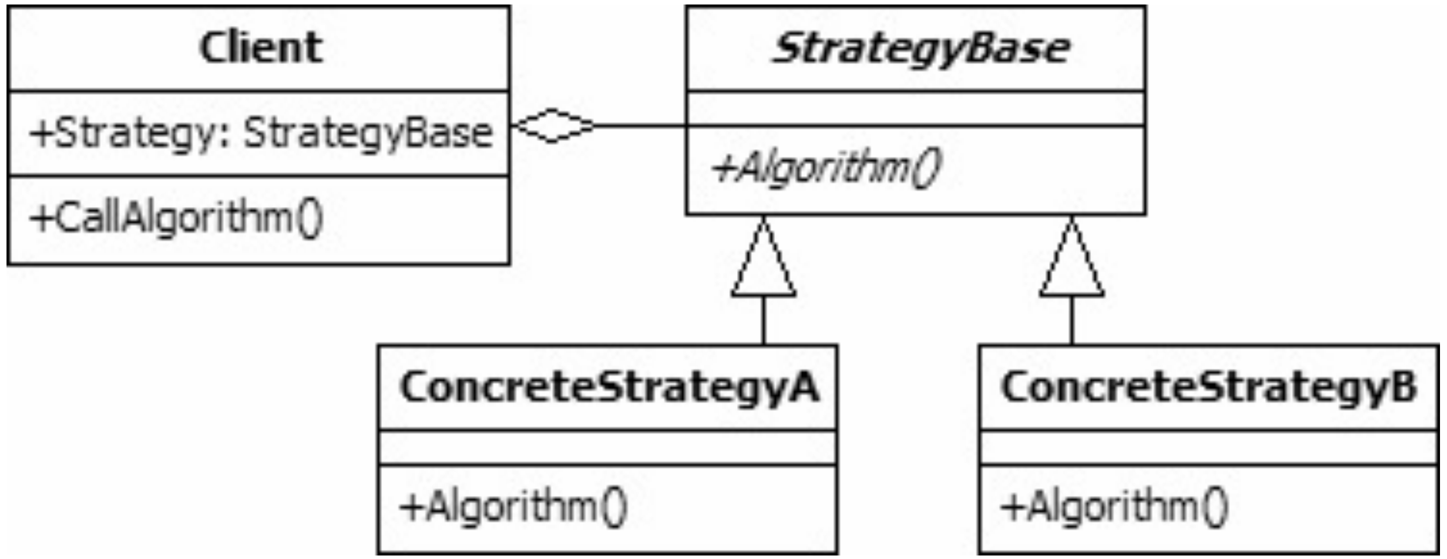
The strategy pattern is a design pattern that allows a set of similar algorithms to be defined and encapsulated in their own classes. The algorithm to be used for a particular purpose may then be selected at run-time according to your requirements.

Overview

- This is a ***behavioural*** pattern as it defines a manner for controlling communication between classes or entities.
- The strategy pattern is used to create an interchangeable family of algorithms from which the required process is chosen at ***run-time***.
- This allows the behaviour of a program to change dynamically according to configuration details or user preferences.
- It also increases flexibility by allowing new algorithms to be easily incorporated in the future.

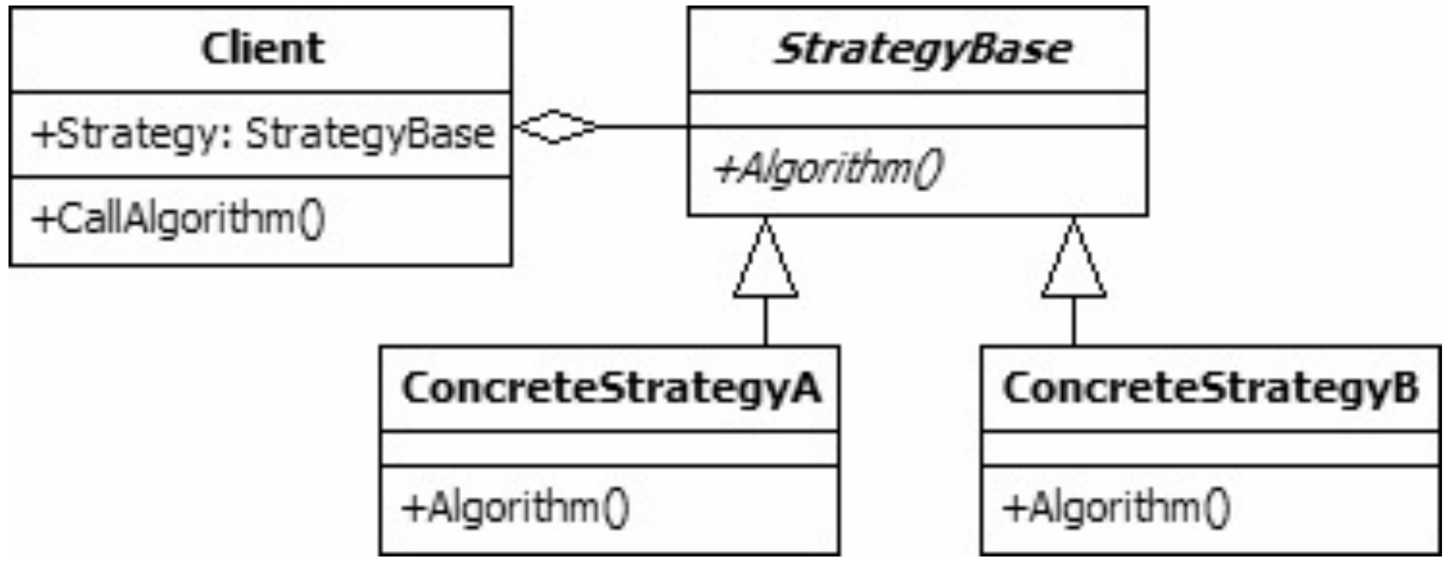
Implementation

Strategy Pattern UML Diagram



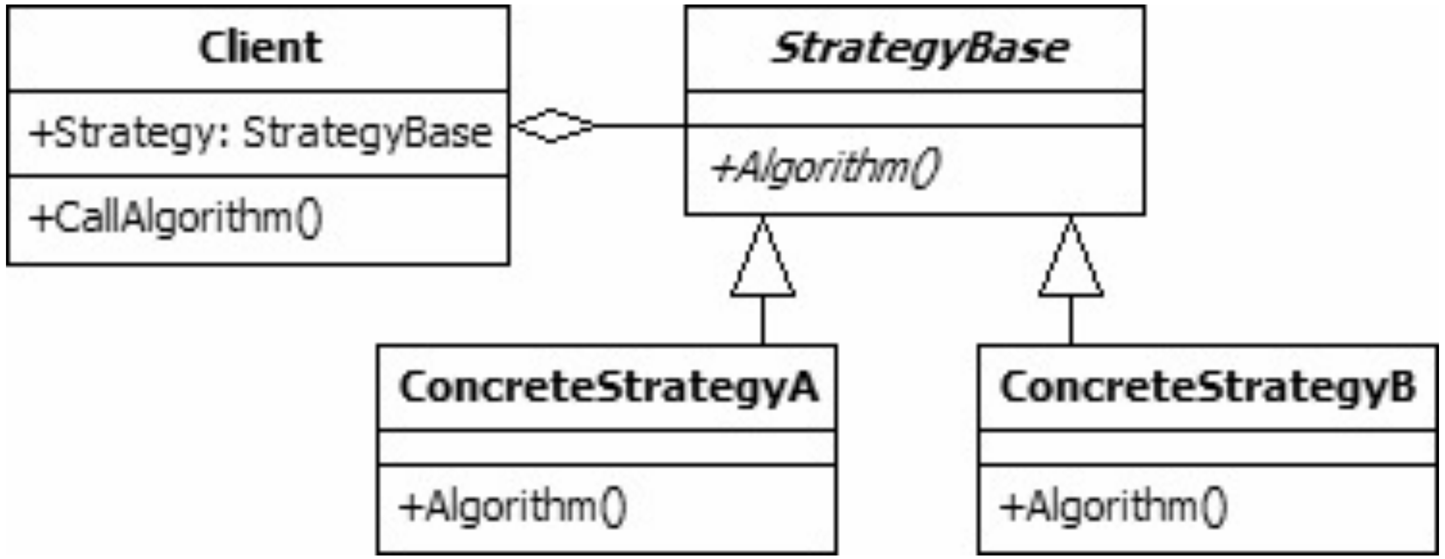
StrategyBase. This abstract class is the base class for all classes that provide algorithms. In the diagram the class includes a single method. However, there is no reason why a number of properties and methods may not be included. This class may be implemented as an interface if it provides no real functionality for its subclasses.

Strategy Pattern UML Diagram



ConcreteStrategyA/B. The concrete strategy classes inherit from the StrategyBase class. Each provides a different algorithm that may be used by the client.

Strategy Pattern UML Diagram



Client. This class is the user of an interchangeable algorithm. The class includes a property to hold one of the strategy classes. This property will be set at run-time according to the algorithm that is required.

Strategy Pattern Basic Code

```
abstract class StrategyBase
{
    public abstract string Algorithm();
}
```

Strategy Pattern Basic Code

```
class ConcreteStrategyA : StrategyBase
{
    public override string Algorithm()
    {
       return "Concrete Strategy A";
    }
}

class ConcreteStrategyB : StrategyBase
{
    public override string Algorithm()
    {
    return "Concrete Strategy B";
    }
}
```

Strategy Pattern Basic Code

```
class Client
{
   public StrategyBase Strategy;

   public Client(StrategyBase Strategy)
   {
      this.Strategy = Strategy;
   }

   public void CallAlgorithm()
   {
      Console.WriteLine(Strategy.Algorithm());
   }
}
```

Strategy Pattern Basic Code

```
class StrategyTemplateApplication
{
    static void Main()
    {
        Client First = new Client(new ConcreteStrategyA());
        Client Second = new Client(new ConcreteStrategyB());

        First.CallAlgorithm();
        Second.CallAlgorithm();

        Console.ReadLine();
    }
}
```

```
Concrete Strategy A
Concrete Strategy B
```

Any Questions?