# Strategy Pattern

A Strategy Pattern says that "defines a family of functionality, encapsulate each one, and make them interchangeable".

The Strategy Pattern is also known as Policy.

---

## Benefits:

- It provides a substitute to subclassing.
- It defines each behavior within its own class, eliminating the need for conditional statements.
- It makes it easier to extend and incorporate new behavior without changing the application.

---

## Usage:

- When the multiple classes differ only in their behaviors.e.g. Servlet API.
- It is used when you need different variations of an algorithm.

---

### Strategy Pattern in (Core Java API's) or JSE 7 API's:

- jav


- jav

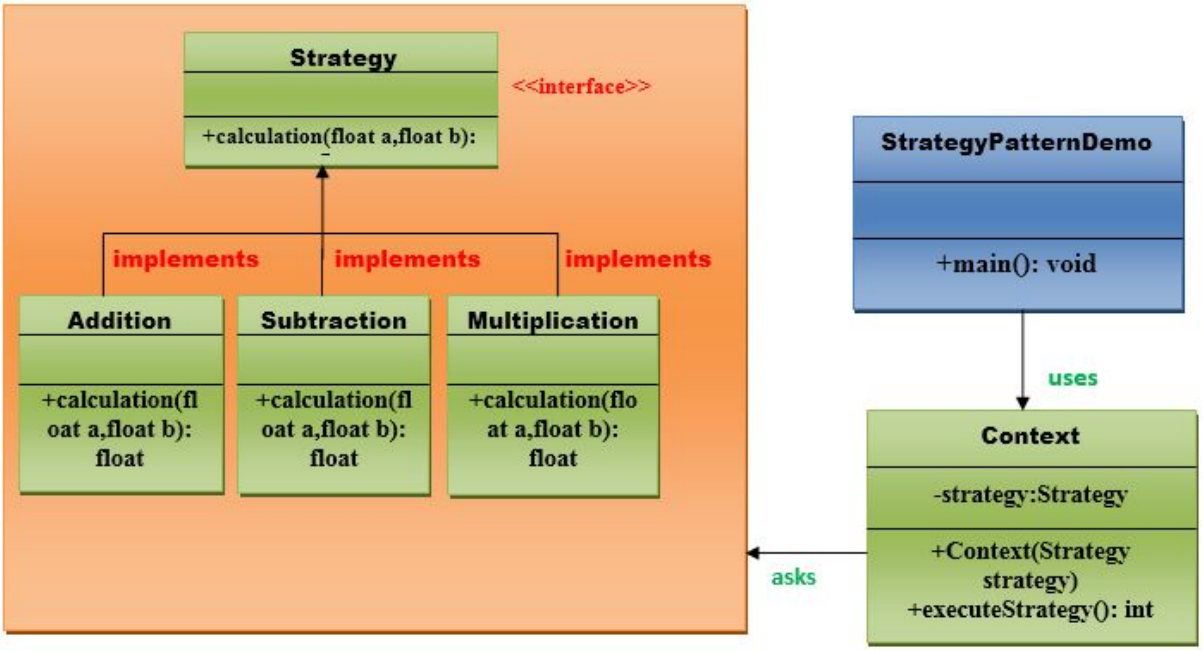

---

### Strategy Pattern in (Advance Java API's) or JEE 7 API's:

- javax.


---

## UML for Strategy Pattern:




---

## Implementation of Strategy Pattern:

**Step 1:**

Create a *Strategy* interface.

```
1. //This is an interface.
2.
3. public interface Strategy {
4.
5.     public float calculation(float a, float b);
6.
7. }// End of the Strategy interface.
```

**Step 2:**

Create a *Addition* class that will implement Startegy interface.

```
1. //This is a class.
```

```
2.  public class Addition implements Strategy{
3.
4.      @Override
5.      public float calculation(float a, float b) {
6.          return a+b;
7.      }
8.
9.  }// End of the Addition class.
```

**Step 3:**

Create a *Subtraction* class that will implement Startegy interface.

```
1.  //This is a class.
2.  public class Subtraction  implements Strategy{
3.
4.      @Override
5.      public float calculation(float a, float b) {
6.          return a-b;
7.      }
8.
9.  }// End of the Subtraction class.
```

**Step 4:**

Create a Multiplication class that will implement Startegy interface.

```
1.  //This is a class.
2.
3.  public class Multiplication implements Strategy{
4.
5.      @Override
6.      public float calculation(float a, float b){
7.          return a*b;
8.      }
9.  }// End of the Multiplication class.
```

**Step 5:**

Create a *Context* class that will ask from Startegy interface to execute the type of strategy.

```
1.  //This is a class.
```

```
2.
3.
4.  public class Context {
5.
6.      private Strategy strategy;
7.
8.      public Context(Strategy strategy){
9.        this.strategy = strategy;
10.     }
11.
12.     public float executeStrategy(float num1, float num2){
13.       return strategy.calculation(num1, num2);
14.     }
15. }// End of the Context class.
```

**Step 6:**

Create a *StartegyPatternDemo* class.

```
1.  //This is a class.
2.  import java.io.BufferedReader;
3.  import java.io.IOException;
4.  import java.io.InputStreamReader;
5.
6.  public class StrategyPatternDemo {
7.
8.     public static void main(String[] args) throws NumberFormatException,
    IOException {
9.
10.         BufferedReader br=new BufferedReader(new
    InputStreamReader(System.in));
11.         System.out.print("Enter the first value: ");
12.         float value1=Float.parseFloat(br.readLine());
13.         System.out.print("Enter the second value: ");
14.         float value2=Float.parseFloat(br.readLine());
15.         Context context = new Context(new Addition());
16.         System.out.println("Addition = " + context.executeStrategy(value1,
    value2));
17.
```

```
18.         context = new Context(new Subtraction());
19.         System.out.println("Subtraction = " + context.executeStrategy(value1,
    value2));
20.
21.         context = new Context(new Multiplication());
22.         System.out.println("Multiplication = " + context.executeStrategy(value1,
    value2));
23.       }
24.
25. }// End of the StrategyPatternDemo class.
```