Lecture 5.
Strategy Pattern (策略模式)
Behavioral

- 行为模式关心算法和对象之间的责任分配。
- 它关心的不是仅仅描述对象或类的模式，而是要更加侧重描述它们之间的通信模式。
- 行为模式刻画了很难在运行时跟踪的复杂的控制流。该模式将软件开发者的注意力从控制流转移到对象相互关联的方式方面。

Professor:
Yushan (Michael) Sun

Fall 2020

Contents of this lecture

1. Introduction to the strategy pattern through Design of a sorting program
2. Theory of the strategy pattern
3. Example of design using the strategy pattern

Introduction to the strategy pattern through Design of a sorting program

Introduction to the strategy pattern through Design of a sorting program

【例1】Design of a sorting program

Requirement: design a program to sort integers by using

- Bubble Sort
- Heap Sort
- Insertion Sort
- Quick Sort

respectively.

Introduction to the strategy pattern through Design of a sorting program

- **Design 1**: use a single class named sorting to implement all the sorting algorithms

Sorting
+main(String[ ] args) +bubbleSort (int[ ] nums) : int[ ] +heapSort (int[ ] nums) : int[ ] +insertionSort (int[ ] nums) : int[ ] +quickSort (int[ ] nums) : int[ ]

最愚蠢的设计

Introduction to the strategy pattern through Design of a sorting program

Problems:

- **增加新算法. When a new algorithm is added to the class Sorting, this whole class needs to be recompiled**
- **修改算法. When an algorithm is modified, this whole class needs to be recompiled**
- **可复用性. The Sorting class cannot be reused directly**

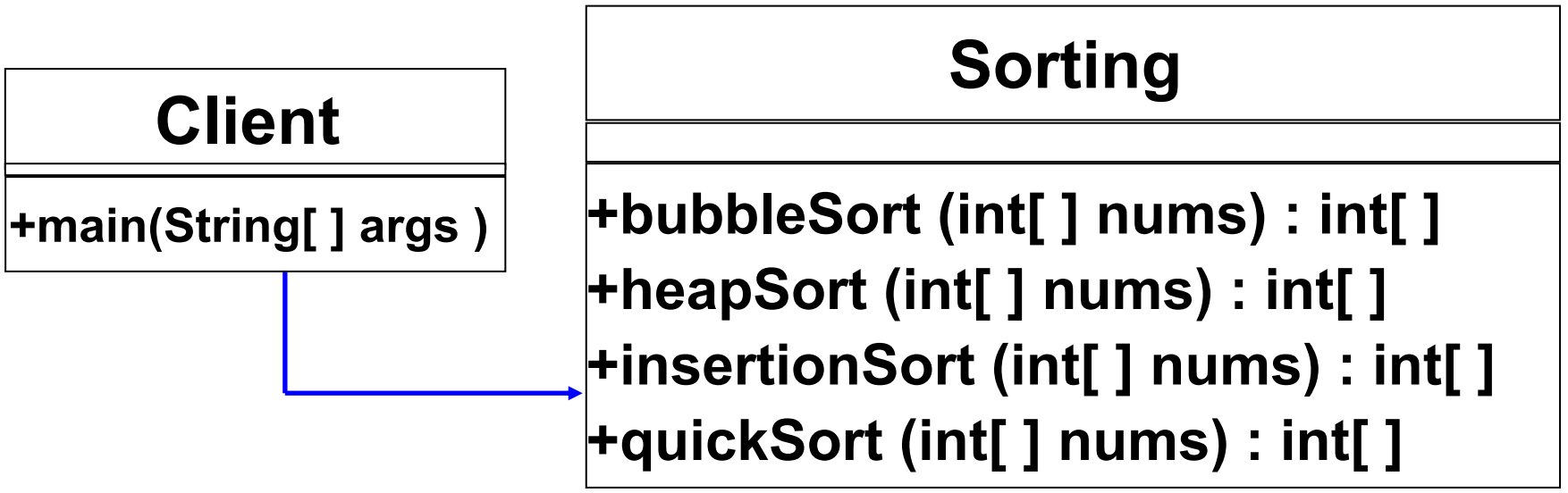
Question:

- **How to improve the design?**

Introduction to the strategy pattern through Design of a sorting program

Design 2: split the class into two classes,

- the Client class and
- the Sorting class, containing all the required algorithms, as illustrated below.



比较正常思维人的设计

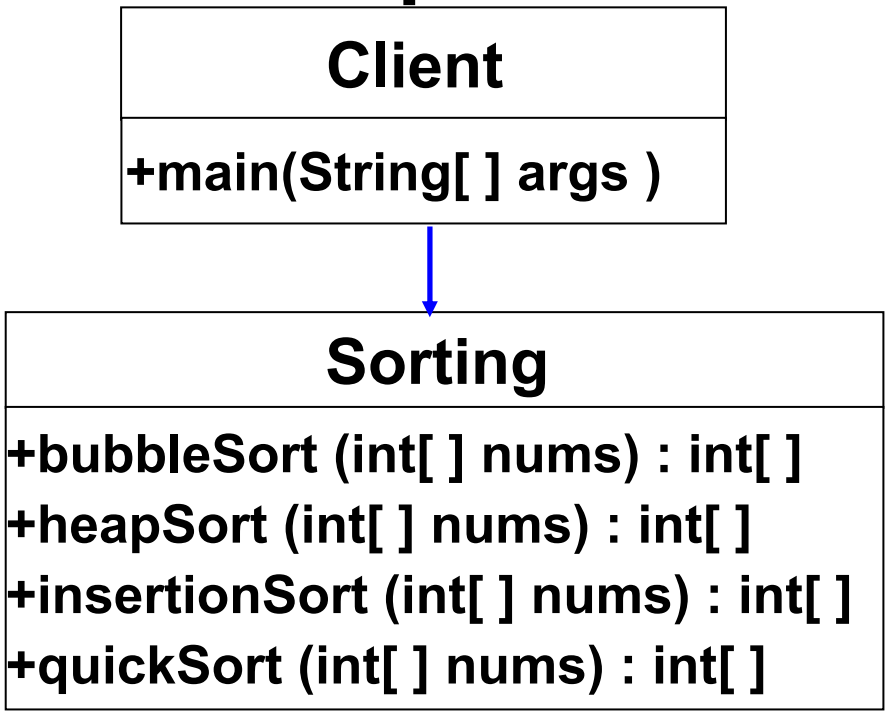
Introduction to the strategy pattern through Design of a sorting program

- **Advantage:**
- **可复用性. The Sorting class can be reused.**

Introduction to the strategy pattern through Design of a sorting program

Drawbacks:

- 扩展：When adding a new algorithm, the class Sorting still needs to be recompiled
- 修改：When modifying an algorithm, the class Sorting needs to be recompiled
- 自顶向下的依赖

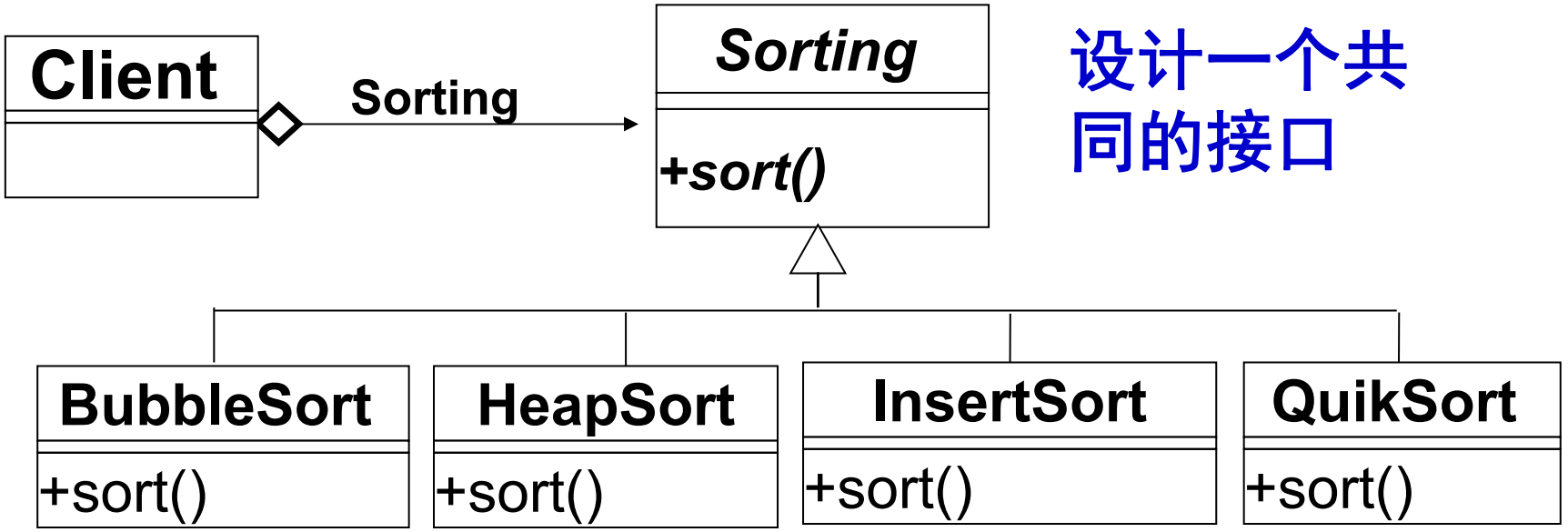


Introduction to the strategy pattern through Design of a sorting program

- 怎样改善设计
- How to improve the design further?
- Any better ideas?

Introduction to the strategy pattern through Design of a sorting program

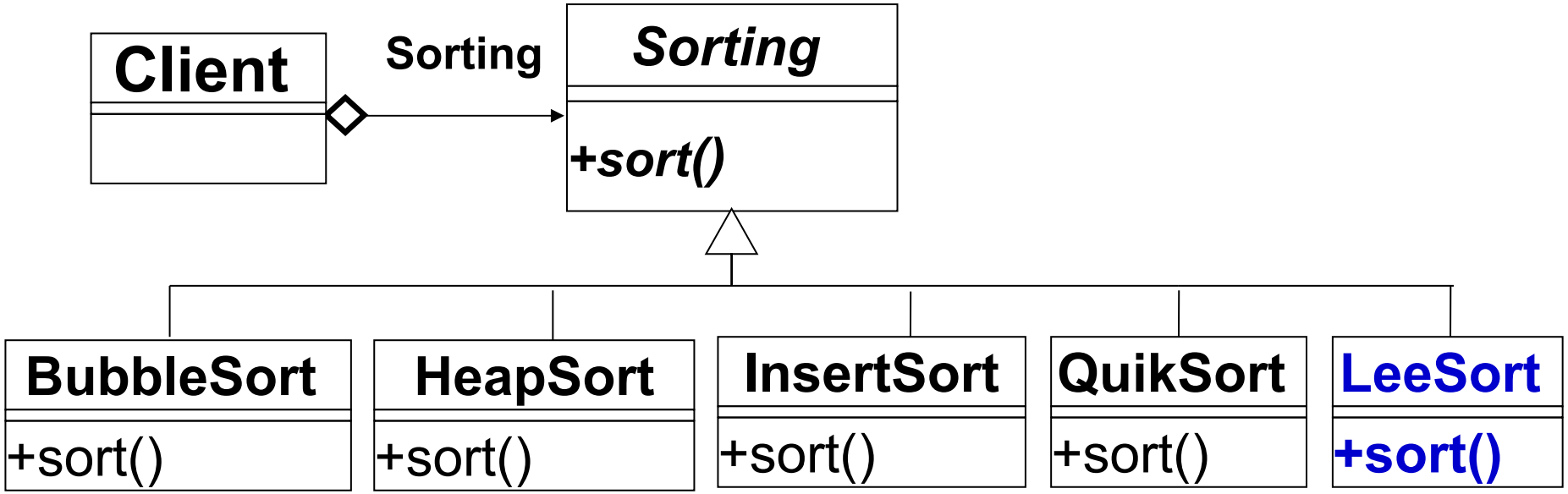
Design 3. We further split the Sorting class into 4 classes to get the following design



Introduction to the strategy pattern through Design of a sorting program

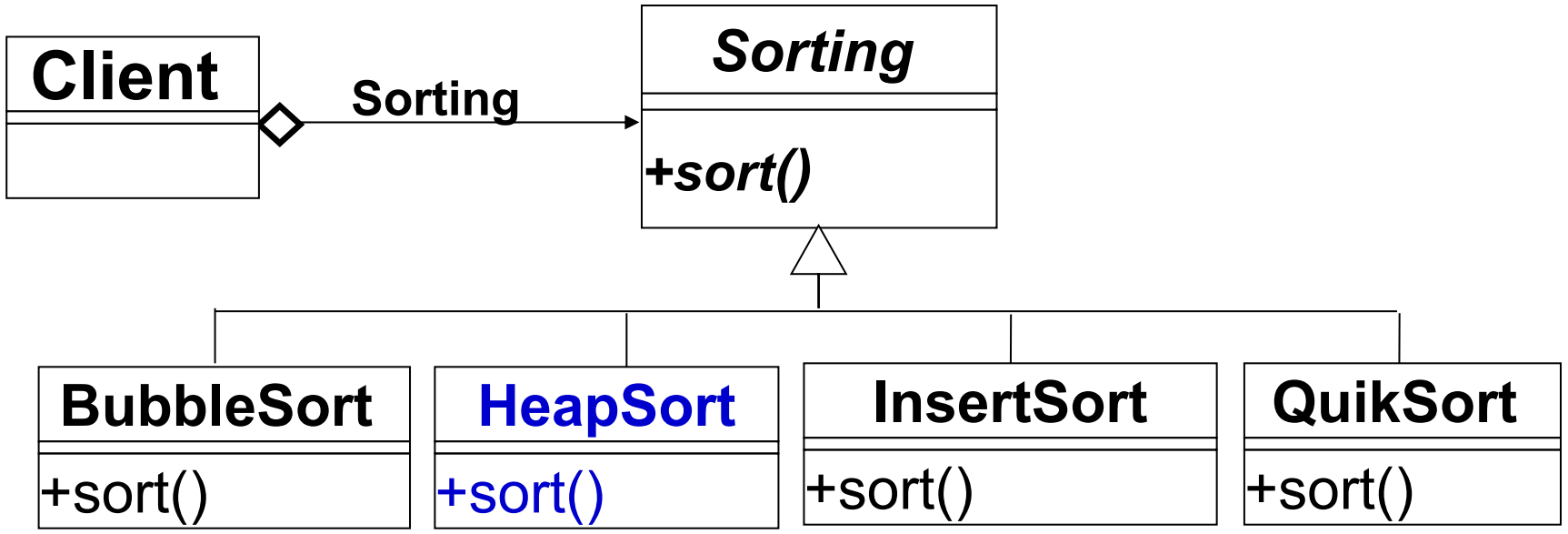
The advantages of this design

a) When a new algorithm is added, the existing Sorting class hierarchy doesn't need to be changed and recompiled



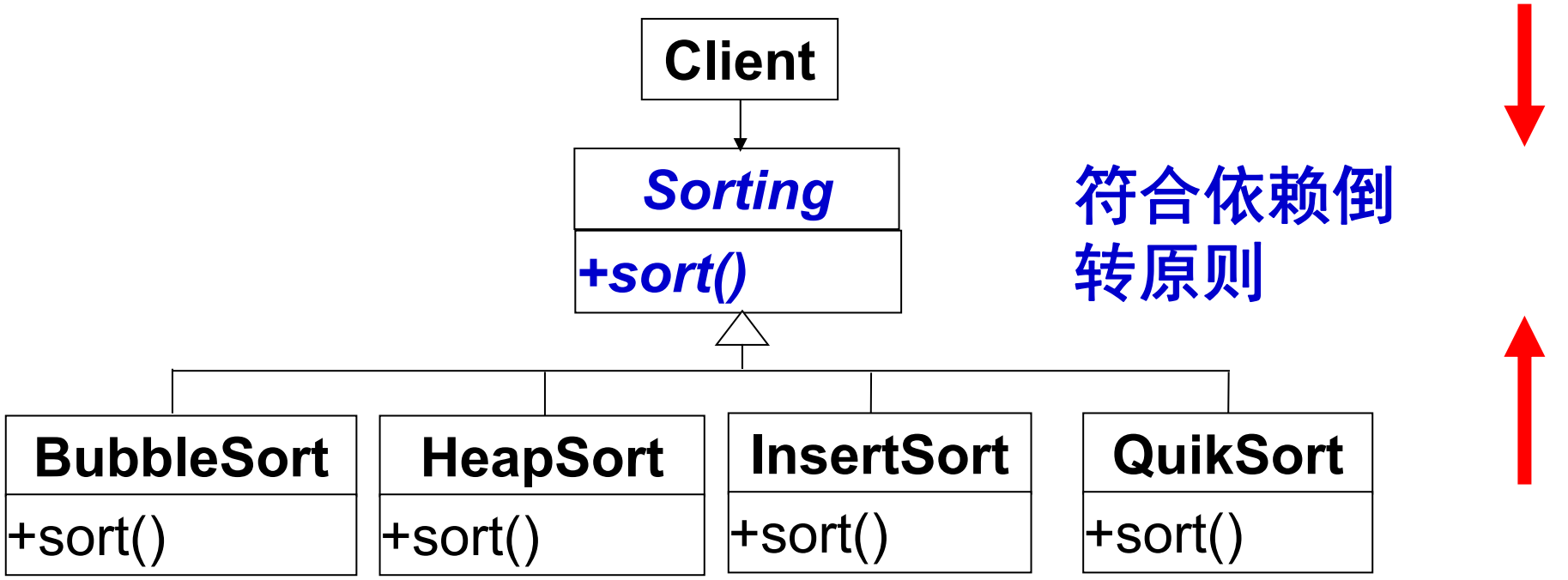
Introduction to the strategy pattern through Design of a sorting program

b) When an algorithm is modified, the remaining classes in the Sorting hierarchy don’ t need to be changed and recompiled



Introduction to the strategy pattern through Design of a sorting program

c) Both the client class and the implementation classes depends on the abstract interface



以上的设计3实际上是使用了策略模式的思想

Back

Something About Code for Sorting

Theory of the Strategy Pattern

Theory of the Strategy Pattern

Structure

Diagram of Strategy pattern

Theory of the Strategy Pattern

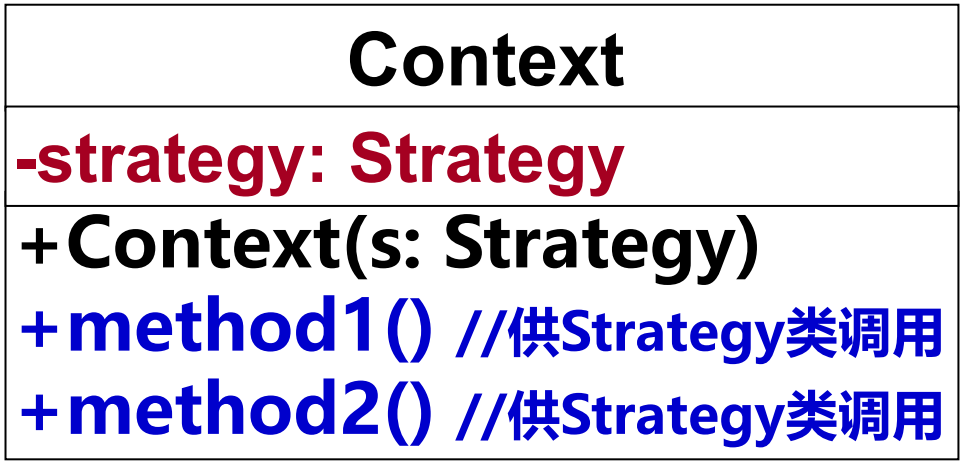
Participants

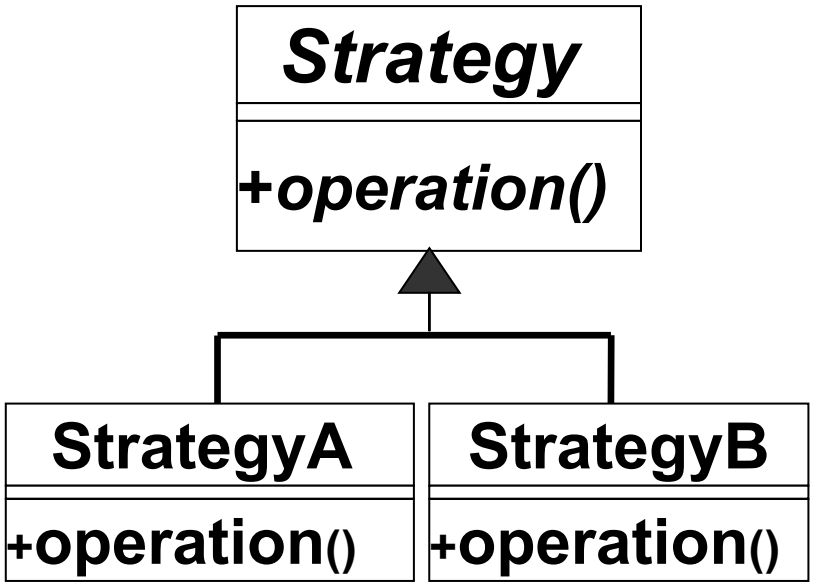
- **Strategy**
  - **declares an interface common to all supported algorithms.**
  - **Context uses this interface to call the algorithms defined by a ConcreteStrategy.**
- **ConcreteStrategy**
  - **implements the algorithm using the Strategy interface.**

Theory of the Strategy Pattern

Context

- **is configured with a ConcreteStrategy object.**
- **maintains a reference to a Strategy object.**
- **may define an interface that lets Strategy access its data.**





Theory of the Strategy Pattern

When to use this pattern?

Use the Strategy pattern in any of the following cases:

1) 当你要处理一些仅仅行为不同的相关的类的时候

When many **related** classes differ only in their behaviors. Strategies provide a way to configure a class with one of many behaviors.

2) 当你要隐藏实现细节的时候 When you need to hide complex algorithm details

When an algorithm uses data that clients shouldn't know about. Use the Strategy pattern to avoid exposing complex, algorithm-specific data structures.

Theory of the Strategy Pattern

When to use this pattern?

3) 当你要取消许多条件语句的时候 When you want to cancel many conditional statements

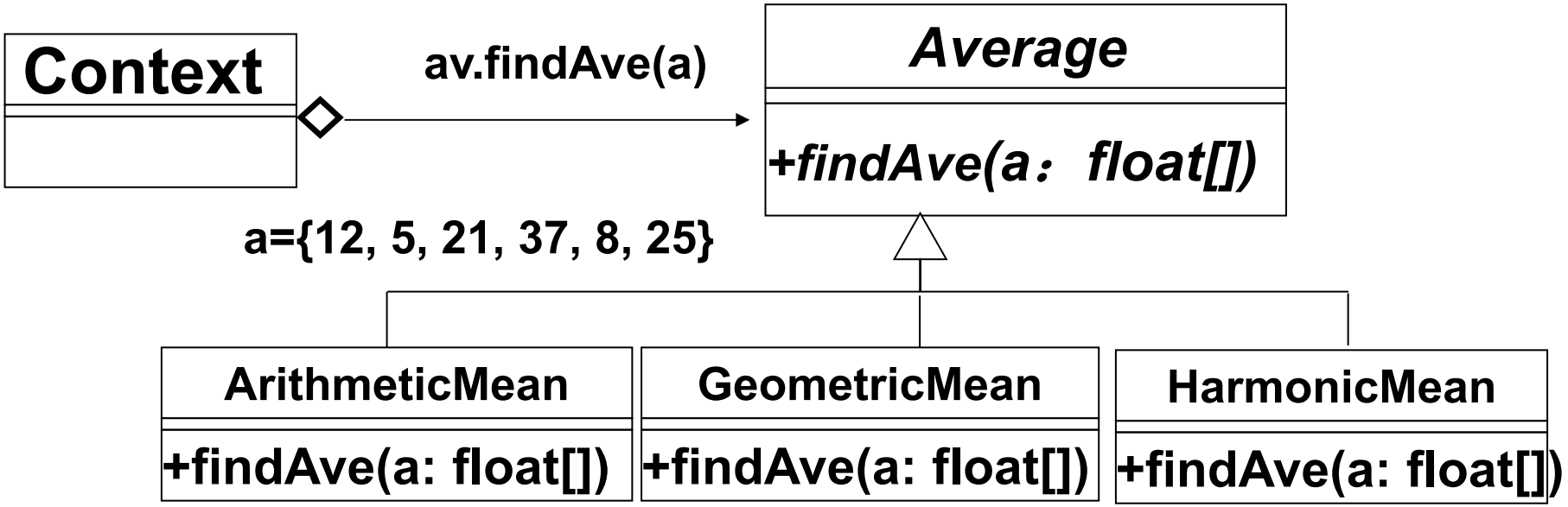
- When a class defines many behaviors, and these appear as multiple conditional statements in its operations.
- Instead of many conditionals, move related conditional branches into their own Strategy class.

Theory of the Strategy Pattern

Context的实现方式:

a) **Pass data to Strategy**

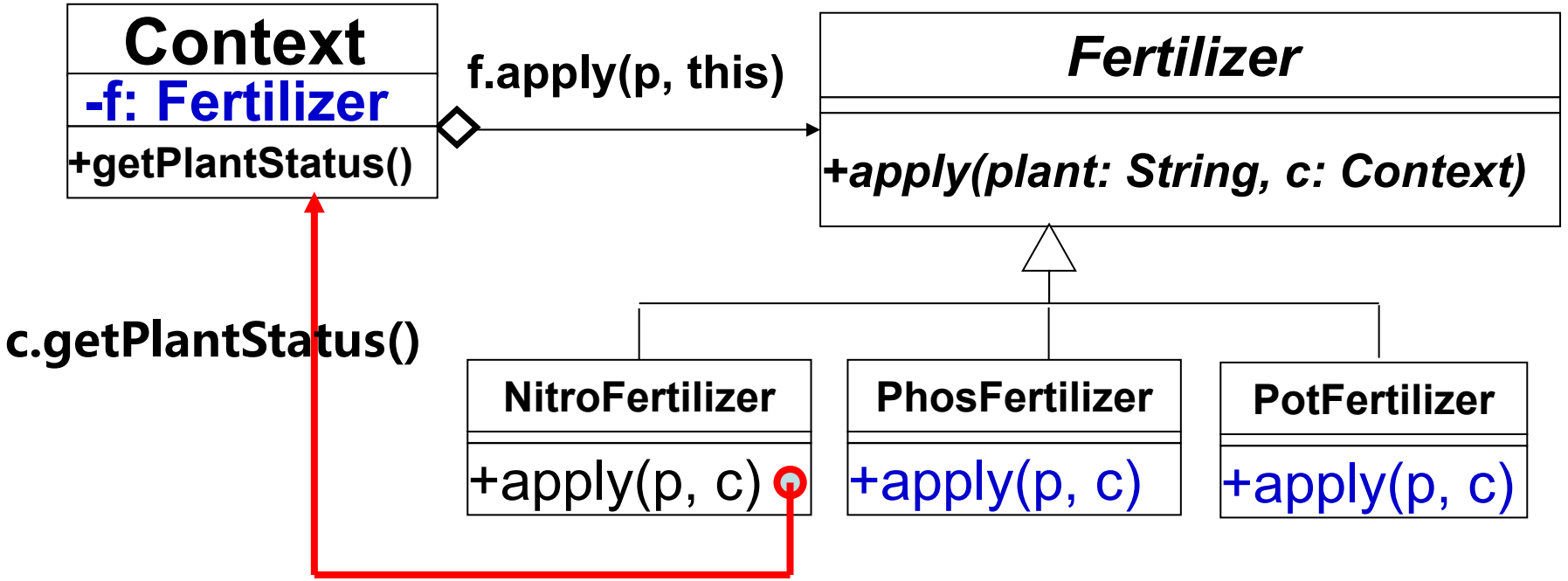
A context may pass all data required by the algorithm to the strategy when the algorithm is called. 当某算法被调用时，Context类将Strategy类所需要的所有数据一次性地传递完毕



Theory of the Strategy Pattern

b) Pass itself as argument to Strategy Functions

A context may pass itself as an argument to Strategy operations. That lets the strategy call back on the context as required. **Context将自己作为一个对象传递给Strategy类，然后Strategy类再利用该对象反过来调用Context类，以便获得某些数据与功能**



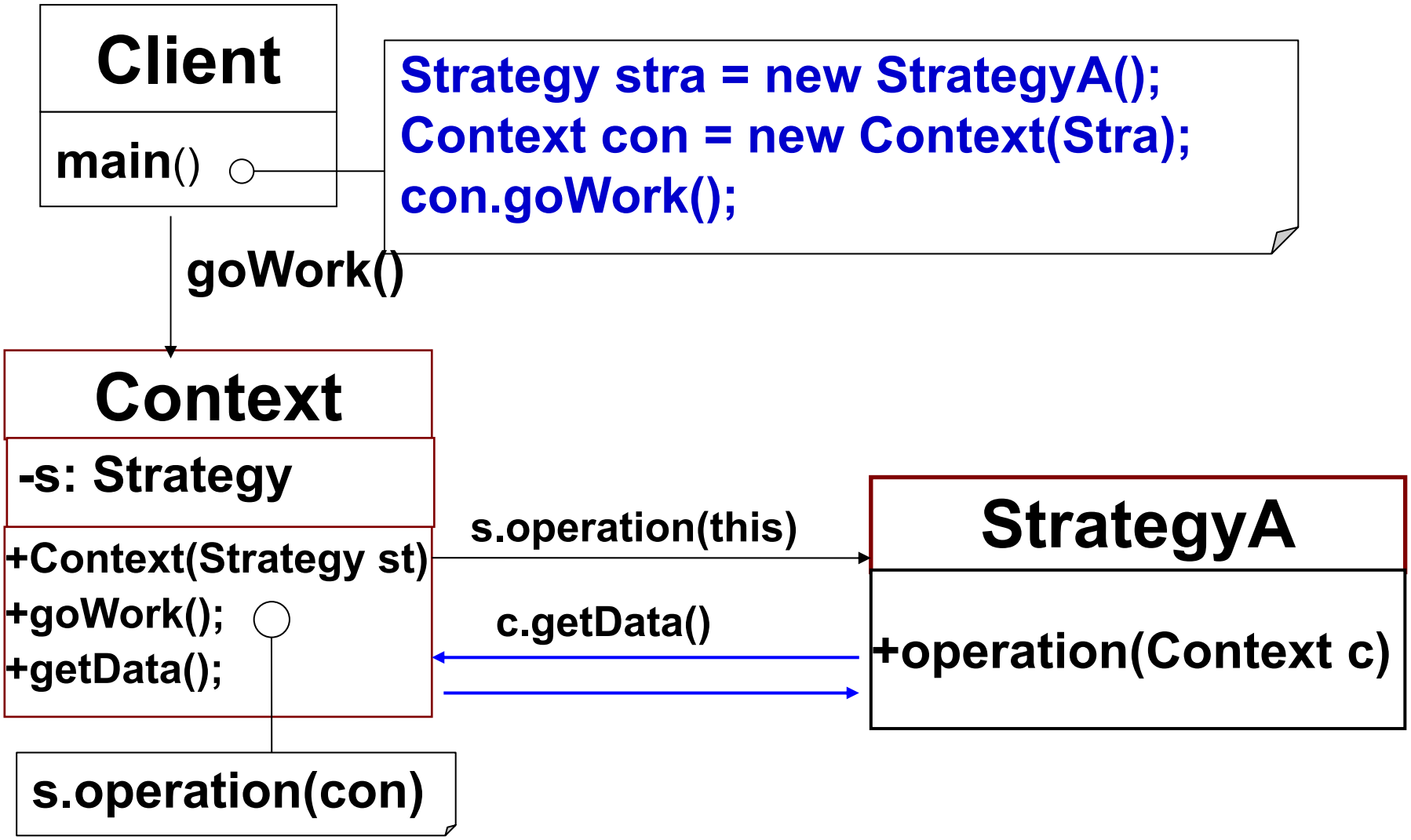
Theory of the Strategy Pattern

通常情况下策略模式的交互

Working mechanism of the strategy pattern:

- **Firstly**, Clients usually create an object of ConcreteStrategy and pass the object to the context;
- **Secondly**, a context forwards requests from its clients to its strategy.
- **Thereafter**, clients interact with the context exclusively (目的).

Theory of the Strategy Pattern



Working mechanism of the strategy pattern

Theory of the Strategy Pattern

策略模式优点 (Advantages of Strategy Pattern)

1. **Define related algorithms for reuse**
   - **Families of related algorithms**. Hierarchies of Strategyclasses define a family of algorithms or behaviors for contexts to reuse.
   - **Inheritance can help factor out common functionality of the algorithms.**

Theory of the Strategy Pattern

2. Easy extension

Encapsulating the algorithm in separate Strategy classes lets you change the algorithm independently of its context (独立于Context), making it easier to

- **understand,**
- **switch (change to a more efficient algorithm)**
- **extend.**

- **When an existing algorithm implementation is changed or a new algorithm is added to the group, both the existing strategy class hierarchy and the context remain unaffected.**

Theory of the Strategy Pattern

策略模式的缺点 (Disadvantages of the strategy pattern)

Clients must know different Strategies.

- A client must understand how Strategies differ before it can select the appropriate one.

Back

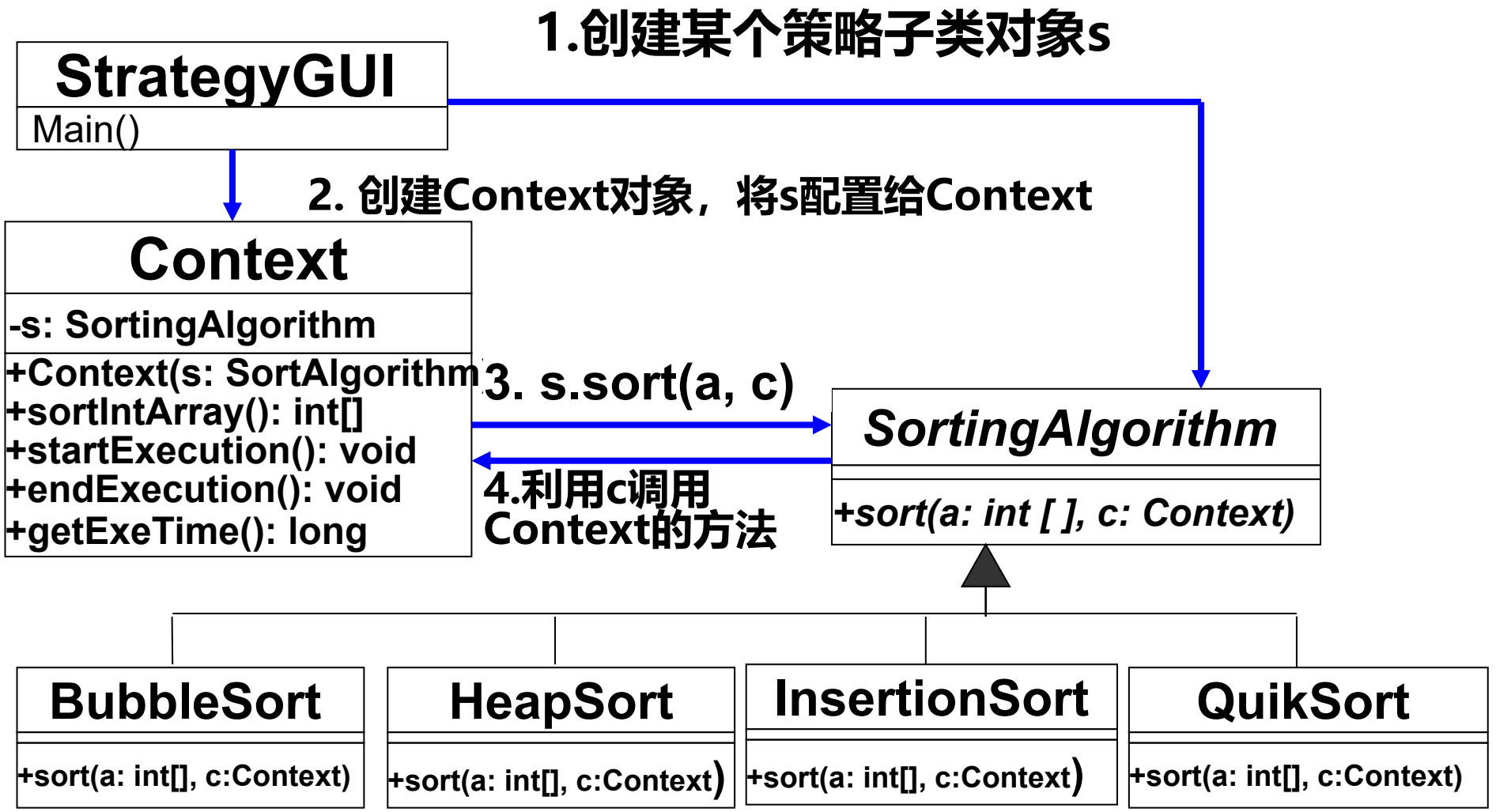
Design Example Using the Strategy Pattern

Example of design using strategy pattern

【例2】Sorting Problem – 利用策略模式的新设计

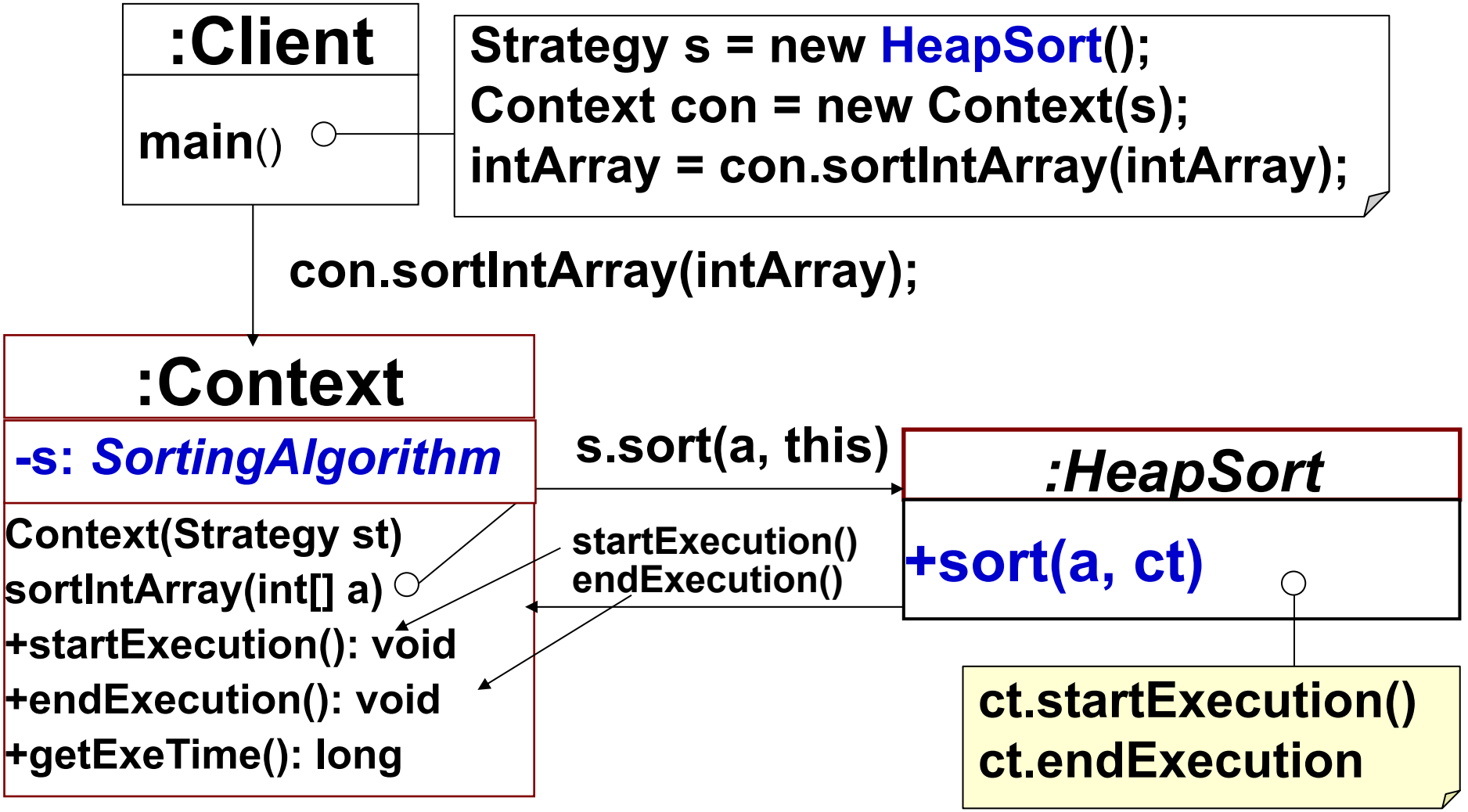
- Design a program using several sorting algorithms to sort integer arrays. The program includes
  - GUI. Graphical User Interface：for user’s input and output
  - 排序算法封装类. Classes to encapsulate the sorting algorithms
  - 计算每个算法的执行时间. to get the execution time taken for each sorting algorithm. The result will be shown on the GUI

Example of design using strategy pattern



应用策略模式的带有Context类的整数序列排序程序设计类图

Example of design using strategy pattern



The typical interactions of the system

Example of design using strategy pattern

```
class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent ae) {
        if (ae.getActionCommand().equals(SORT)) {
            if (type.equals(BUBBLE))
                sa = new BubbleSort();
            if (type.equals(HEAP))
                sa = new HeapSort();
            if (type.equals(INSERT))
                sa = new InsertSort();
            if (type.equals(QUICK))
                sa = new QuickSort();

            context = new Context(sa);
            intArray = context.sortIntArray(intArray);
            long eTime = context.getExeTime();
    }}}
```

1. 创建策略子类对象sa
2. 创建Context的对象时，将sa作为参数传递给Context
3. 使用Context对象调用sortIntArray(intArray)

getExeTime()方法

Example of design using strategy pattern

```
class Context {
        SortAlgorithm alg;
        private long startTime;
        private long endTime;

        public Context(SortAlgorithm alg) {
            this.alg = alg;
        }
        public int[] sortIntArray(int[] a) {
            return this.alg.sort(a, this);
        }
        public void startExecution(){
          startTime = System.currentTimeMillis();
        }
        public void endExecution(){
          endTime = System.currentTimeMillis();
        }
        public long getExeTime(){
          long exeTime=0;
          exeTime = endTime-startTime;
          return exeTime;
        }
}
```

1. 策略子类对象alg由构造方法传入
2. 在sortIntArray方法中，直接使用alg调用策略类的sort方法
3. 同时，将Context对象传递给策略类
4. 策略类利用该Context对象调用计算程序执行时间的方法 startExecution() endExecution()

Example of design using strategy pattern

```
public interface SortAlgorithm {
    int[] sort(int[] nums, Context ct);
}
```

Example of design using strategy pattern

```
public class BubbleSort implements
                                    SortAlgorithm {
   public int[] sort(int[] intArray, Context ct){
      ct.startExecution();
      // 排序代码
      ct.endExecution();
      return intArray;
   }
}
```

其余算法的实现与此算法类似，省略。

Example of design using strategy pattern

【例3】 在政府或商业网站上，许多数据需要用图表解 释，用户可选择不同的图表以便方便地显示一些数据，例如显示全国各个行业的外贸出口比例等等。同一组数据，可以有不同的图表显示：

- 条形图(Bar)，
- 折线图(Line)，
- 饼图(Pie)，
- XY散点图(XY (scatter))，
- 面积图(area)

等等图形。

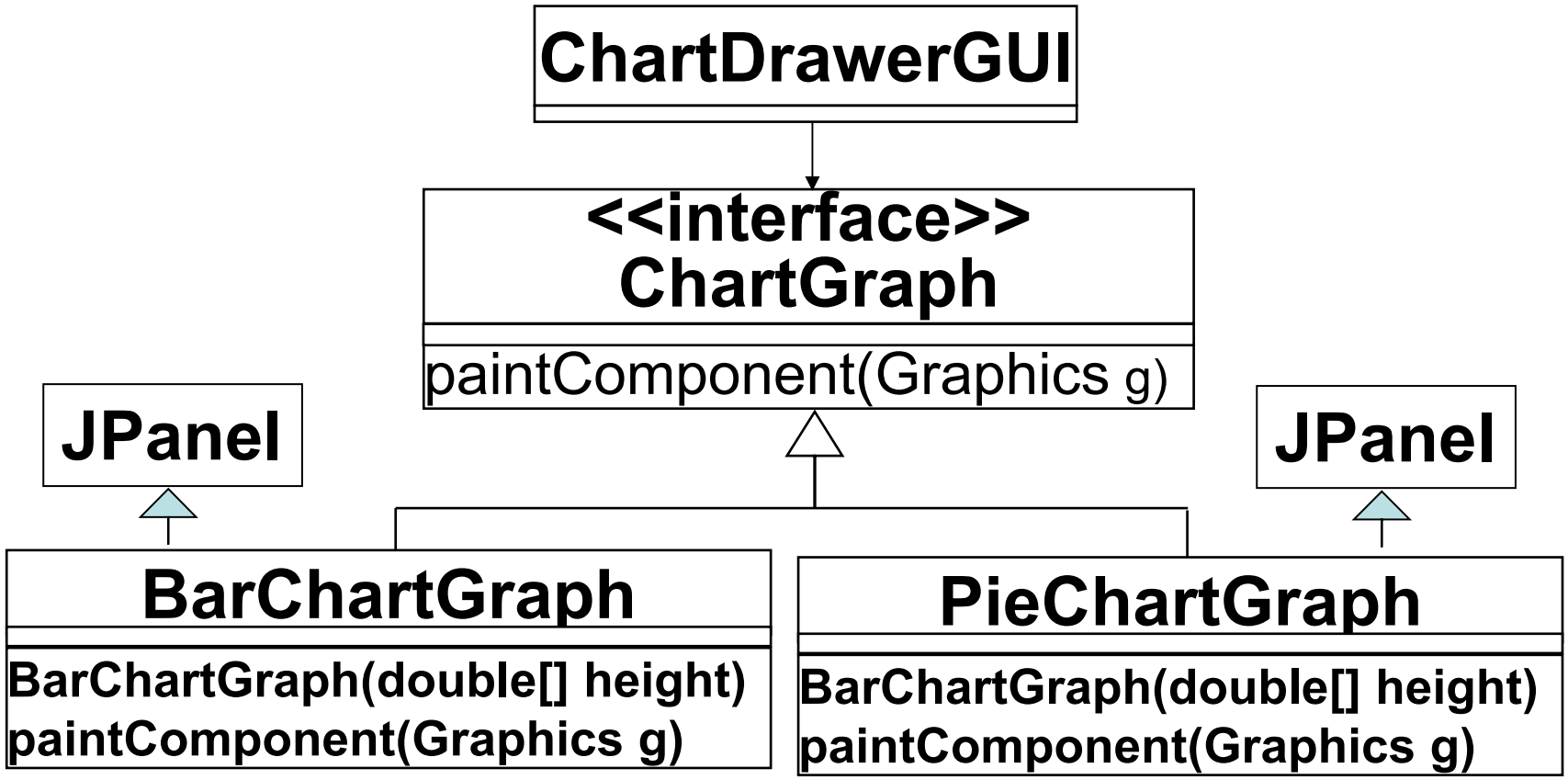
Example of design using strategy pattern

为什么要使用策略模式进行设计?

- 假设在用户在输入一组数据以后，程序将为你根据需要画出你所需要的图表类型。
- 各种图表，虽表现形式不同，但都显示同一组数据，功能是相同的。可以使用策略模式实现各种图表的显示。
- 将该根据用户输入的一组数据，利用各种不同的图标显示。为简单起见，我们在本例子中，只使用
  - 条形图(Bar)，和
  - 饼图(Pie)。

Example of design using strategy pattern

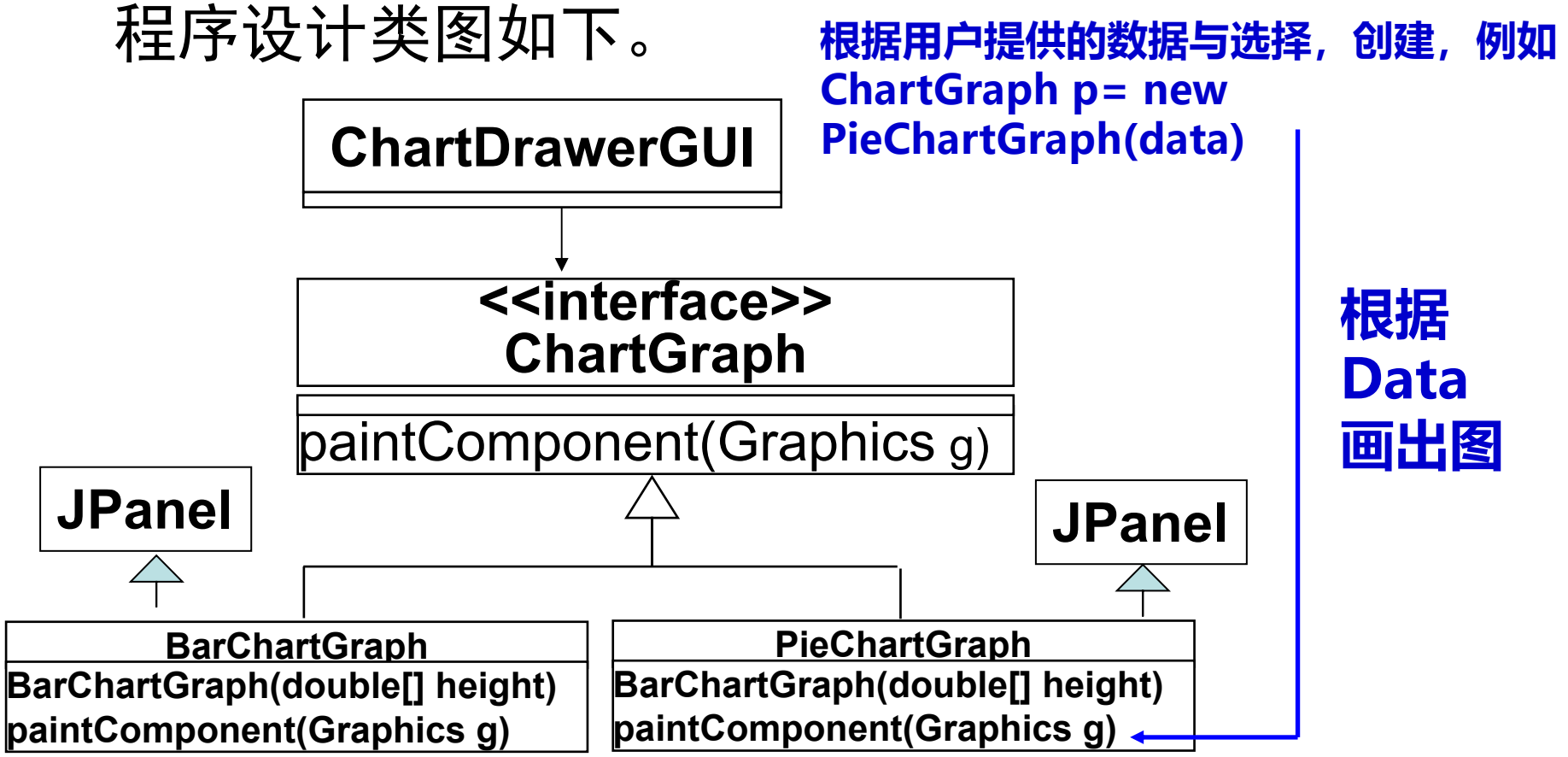
程序设计类图如下。



使用策略模式设计的绘图软件-省略Context类的情况

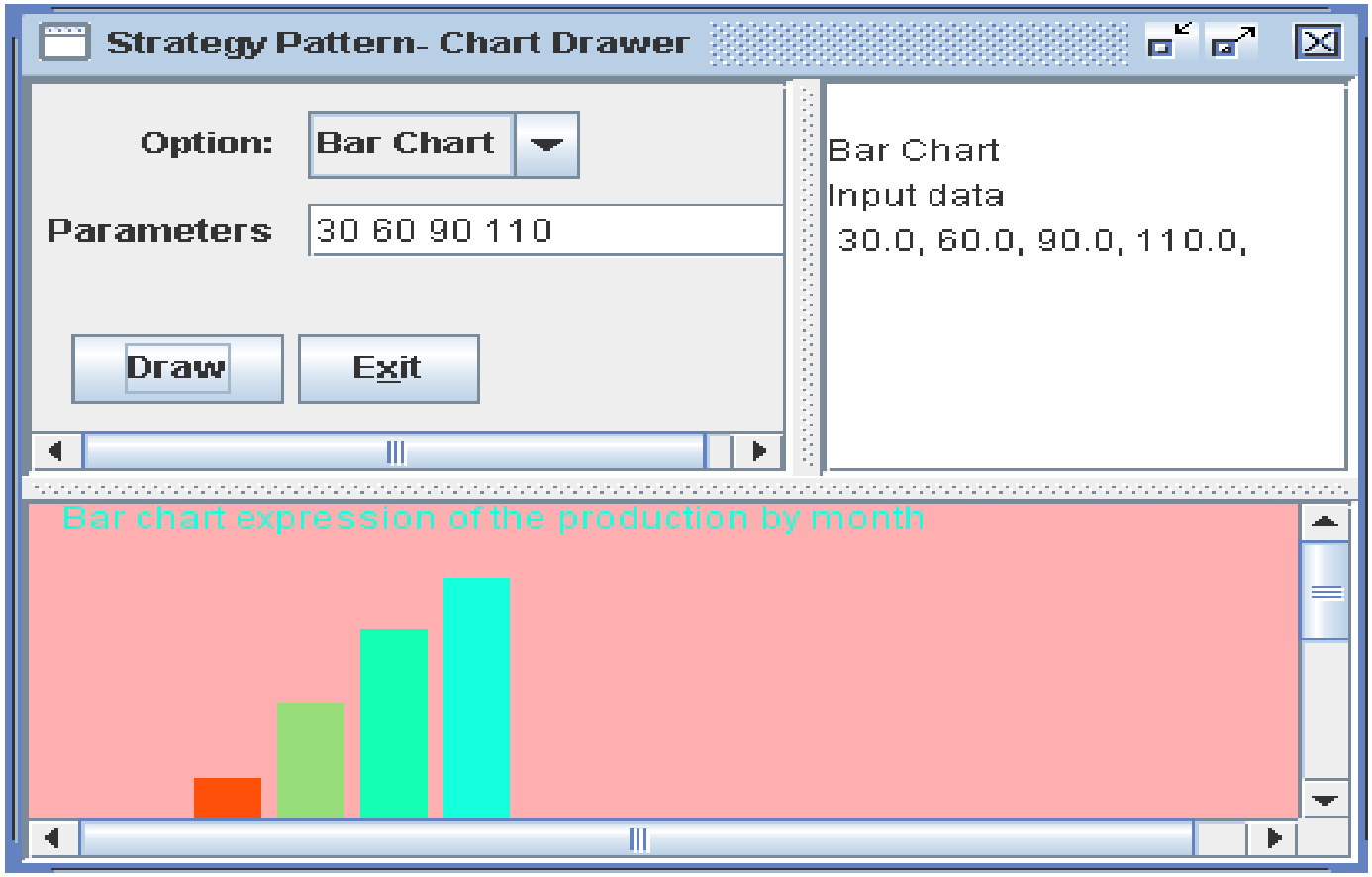
Example of design using strategy pattern

程序设计类图如下。



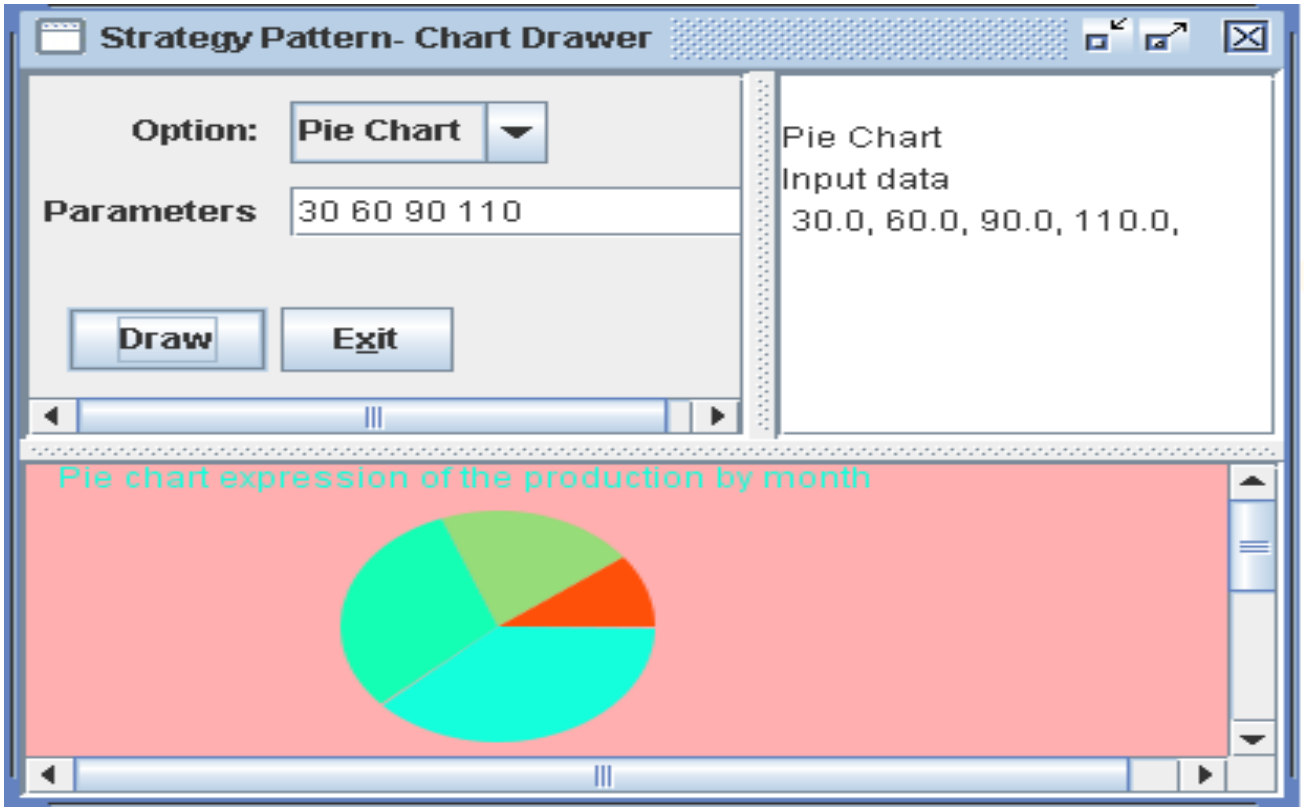
使用策略模式设计的绘图软件–交互情况

Example of design using strategy pattern



用户图形界面-当用户选取了Barchart的情况

Example of design using strategy pattern



用户图形界面-当用户选取了Piechart的情况

Example of design using strategy pattern

为什么在本设计中省略了Context类?

- 两个类BarChartGraph和PieChartGraph中，都有一个paintComponent(Graphics g)方法。在该方法中，调用其超类JPanel的paintComponent(Graphics g)方法。
- 在创建BarChartGraph或PieChartGraph的对象时，paintComponent(Graphics g)方法将自动运行。
- 即，BarChartGraph或PieChartGraph的对象代表一个JPanel，且相应的图形，例如Bar chart，已被绘制在该JPanel上。
- 为了将图形显示在GUI上，只要创建BarChartGraph或PieChartGraph对象并将其贴在GUI即可。
- 故省略了context类。

Back