Strategy Pattern Tutorial

Definition

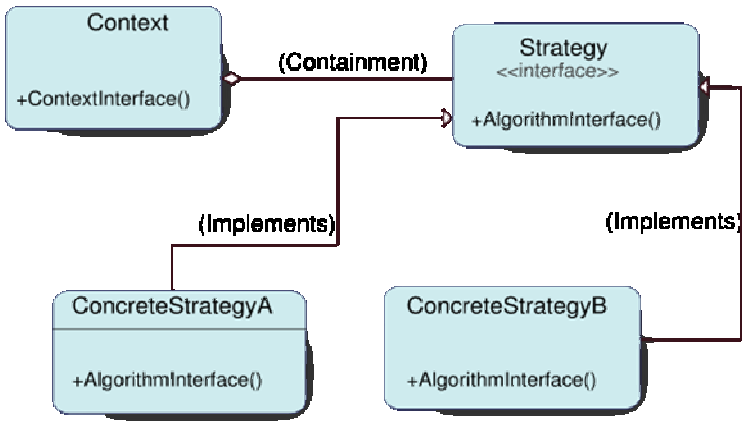
Define a family of algorithms, encapsulate, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it **(http://en.wikipedia.org/wiki/Strategy_pattern)**

Simply Speaking

Strategy is what we group the many algorithms that do the same things and make it interchangeable at run-time

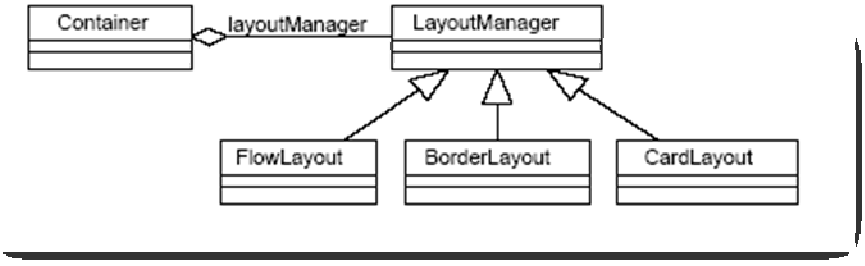
UML Example #1

The UML below illustrates the idea of Strategy pattern.



- It means that **Context** class has the **Strategy** object.
- And ConcreateStrategyA, ConcreteStrategyB is the choice of algorithm that does the same purpose.
- When you used at runtime, you can switch between **ConcreateStrategyA** and **ConcreteStrategyB** to use.

UML Example #2



- This UML, We want to set the layout for the GUI Components. So, in order to set the layout, we have 3 choices of setting layout which are

- FlowLayout
- BorderLayout
- CardLayout

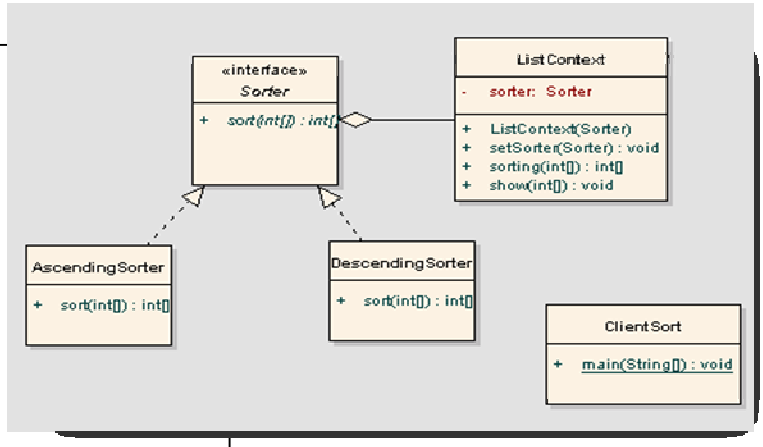
So at run-time, you can be able to switch each type of layout.

UML Example #3 – Let's try…

So for more understanding, let's see Sort Example. Let's try to implement by yourself.

Just start from understand the purpose and its UML first, and then try to code in Eclipse or your IDE.

Purpose: Write the **sort** program which support multiple sort algorithms, and make it changeable at runtime




```
package Strategy;

public class ClientSort
{
	public static void main(String[] args)
	{
		// Sort for AscendingSorter
ListContext listContext = new ListContext(new
AscendingSorter());
		int[] listNumber = new int[5];
		listNumber[0] = 9;
		listNumber[1] = 1;
		listNumber[2] = 5;
		listNumber[3] = 2;
		listNumber[4] = 8;
		listNumber = listContext.sorting(listNumber);
		System.out.println("Ascending Sorting");
		listContext.show(listNumber);

		// Sort for DescendingSorter
		listContext.setSorter(new DescendingSorter());
		listNumber = listContext.sorting(listNumber);
		System.out.println("Descending Sorting");
		listContext.show(listNumber);

	}
}
```

```
package Strategy;

public class ListContext
{
	private Sorter sorter;

	public ListContext(Sorter sorter)
	{
		this.sorter = sorter;
	}

	public void setSorter(Sorter sorter)
	{
		this.sorter = sorter;
	}

public int[] sorting(int[] listNumber)
	{
	return this.sorter.sort(listNumber);
	}

	/**
	 * Printing for show the result.
	 *
	 */
	public void show(int[] listNumber)
	{
```

```
package Strategy;

public interface Sorter
{
public int[] sort(int[] listNumber);
}
```

```
package Strategy;
public class AscendingSorter implements
Sorter
{
public int[] sort(int[] listNumber)
	{
	// add ascending Sort Algorithm
	return listNumber;
	}

}
```

```
package Strategy;
public class DescendingSorter implements
Sorter
{
public int[] sort(int[] listNumber)
	{
	// add descending Sort Algorithm
	return listNumber;
	}
```