InterBit
Training & Consulting Ltd.

Strategy Pattern

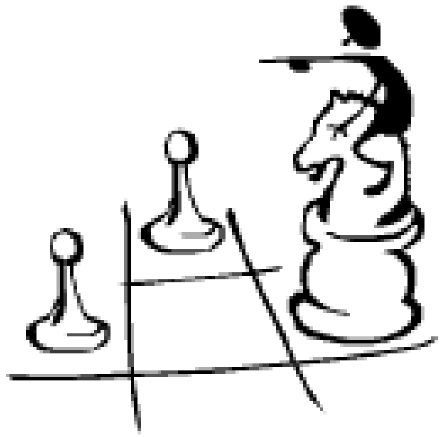
Sun microsystems | AUTHORIZED MySQL EDUCATION CENTER | solaris | FORTINET | symantec.

Chapter Content

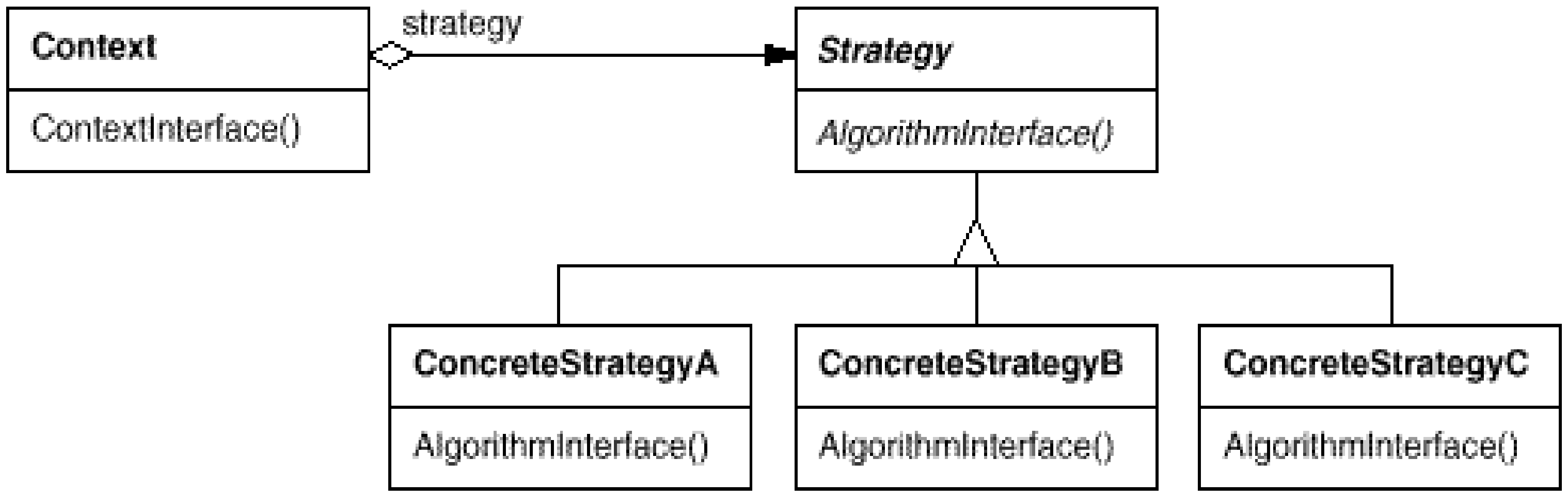
- Strategy Pattern Overview
- Strategy Pattern UML Diagram
- Strategy - comparators
- Strategy - layout managers
- Strategy - victory conditions
- Strategy - access considerations
- Similarity to other patterns

Strategy Pattern Overview

- The Strategy Pattern allows you to “plug” **different algorithms** (strategies) into the same general code.



Strategy Pattern UML Diagram



Context - contains a reference to a strategy object. It may define an interface that lets strategy accessing its data.

The Context objects contains a reference to the ConcreteStrategy that should be used. When an operation is required then the algorithm is run from the strategy object.

The context object receives requests from the client and delegates them to the strategy object. Usually the ConcreteStartegy is created by the client and passed to the context. From this point the clients interacts only with the context.

Strategy: Comparators

- Java can do sorting with different comparison strategies:

```
// Employees can be sorted either by name or by salary:
class Employee {
    String name;
    double salary; ...
}
class NameComparator implements Comparator {
    public int compare(Object obj1, Object obj2){
        Employee e1=(Employee)obj1 , e2=(Employee)obj2;
        return (e1.getName().compareTo(e2.getName()));
    }
}
```

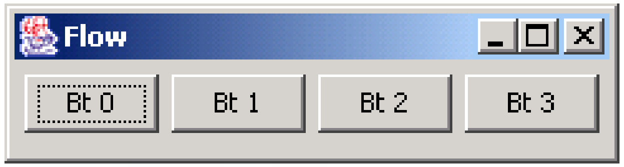
Strategy: Comparators (cont.)

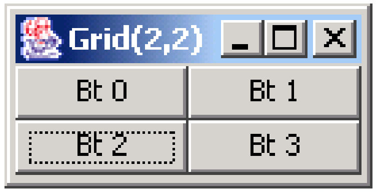
```
class SalaryComparator implements Comparator {
    public int compare(Object obj1, Object obj2){
     Employee e1=(Employee)obj1 , e2=(Employee)obj2;
     double diff = e1.getSalary() - e2.getSalary();
     if (diff == 0) return 0;
     else return (diff > 0 ? 1 : -1);
    }
}

Usage:
    TreeSet tree = new TreeSet(new NameComparator());  // or SalaryComparator
    tree.add(new Employee("Yossi", 9998));
    tree.add(new Employee("Miri", 9999));
    for(Iterator iter=tree.iterator(); iter.hasNext();)
      System.out.println(iter.next());
```

Strategy – layout managers

- Every swing/awt container has a layout manager, which determines a strategy for arranging components inside:

```
public class MyFrame extends JFrame{
    public MyFrame(){
        Container cp = getContentPane();
        cp.setLayout(new GridLayout(2,2));  // or: new FlowLayout()
        for(int i=0; i<4; i++)  cp.add(new JButton("Bt "+i));
    }
}
```





Strategy – victory conditions

InterBit Training & Consulting Ltd.

- Configuring a game with different victory conditions:

```
class TicTacToeGame {
    JFrame gameFrame;  // graphical display
    char[] board;          // board state
    Player[] players;  // 2 players
    VictoryChecker vic ;    // checks for victory !
    ...
    public TicTacToeGame(VictoryChecker vic) { this.vic=vic;}
}

Interface VictoryChecker {
    boolean isVictorious(Player p, char[] board);
}
```

vic is responsible for deciding when a player is considered to have won

Victory Conditions (cont.)

```
// Any row, column or diagonal is considered a victory:
class DefaultVictoryChecker implements VictoryChecker{
    private TicTacToeGame game;
    public boolean isVictorious(Player p, char[] board) {
        ...  // look for victory in rows, columns or diagonals
    }
}

// Row and column are considered a victory, but diagonals are not:
class NoDiagVictoryChecker implements VictoryChecker{
    private TicTacToeGame game;
    public boolean isVictorious(Player p, char[] board) {
       ...  // look for victory in rows/columns but not in diagonals
    }
}
```

Access considerations

- Careful consideration is required regarding communication between the strategy class and the encapsulation class.
  - In particular, strategy may be required to access some data that shouldn’t be visible to other classes.

Problems and implementation

Passing data to/from Strategy object

- Usually each strategy need data from the context and have to return some processed data to the context. This can be achieved in 2 ways.
  - Creating some additional classes to encapsulate the specific data.
  - passing the context object itself to the strategy objects. The strategy object can set returning data directly in the context.

Problems and implementation

Passing data to/from Strategy object

- When data is passed the drawbacks of each method should be analyzed.
  - For example, if some classes are created to encapsulate additional data, a special care should be paid to what fields are included in the classes.
  - The current implementation may requires fields that are added, but in the future some new strategy will require data which is not available in fields.
  - It's also very likely that some of the strategy concrete classes will not use field passed to the in the additional classes.
- On the other side, if the context object is passed to the strategy then we have a tighter coupling between strategy and context.

Problems and implementation

Optionally Concrete Strategy Objects

- It's possible to implement a context object that carries an implementation for default or a basic algorithm.
  - While running it, it checks if it contains a strategy object. If not it will run the default code and that's it.
  - If a strategy object is found, it is called instead (or in addition) of the default code.
- This is an elegant solution to exposing some customization points to be used only when they are required. Otherwise the clients don't have to deal with Strategy objects.

Similarity to other patterns

- **How is strategy different from state?**
  - Object is likely to frequently change its state, while strategy rarely changes.
- **How is it different from command?**
  - There are similarities (C++/STL programmers would categorize both as functors).
  - Strategy makes more assumptions about the code to be executed (specific algorithm), while command is general (whatever it is, execute it).

Strategy Summary

- The strategy design pattern splits the behavior (there are many behaviors) of a class from the class itself.
- This has some advantages, but the main draw back is that a client must understand how the Strategies differ.
- Since clients get exposed to implementation issues the strategy design pattern should be used only when the variation in behavior is relevant to them.