

Software Design and Construction 1

SOFT2201 / COMP9201

Assignment Project Exam Help

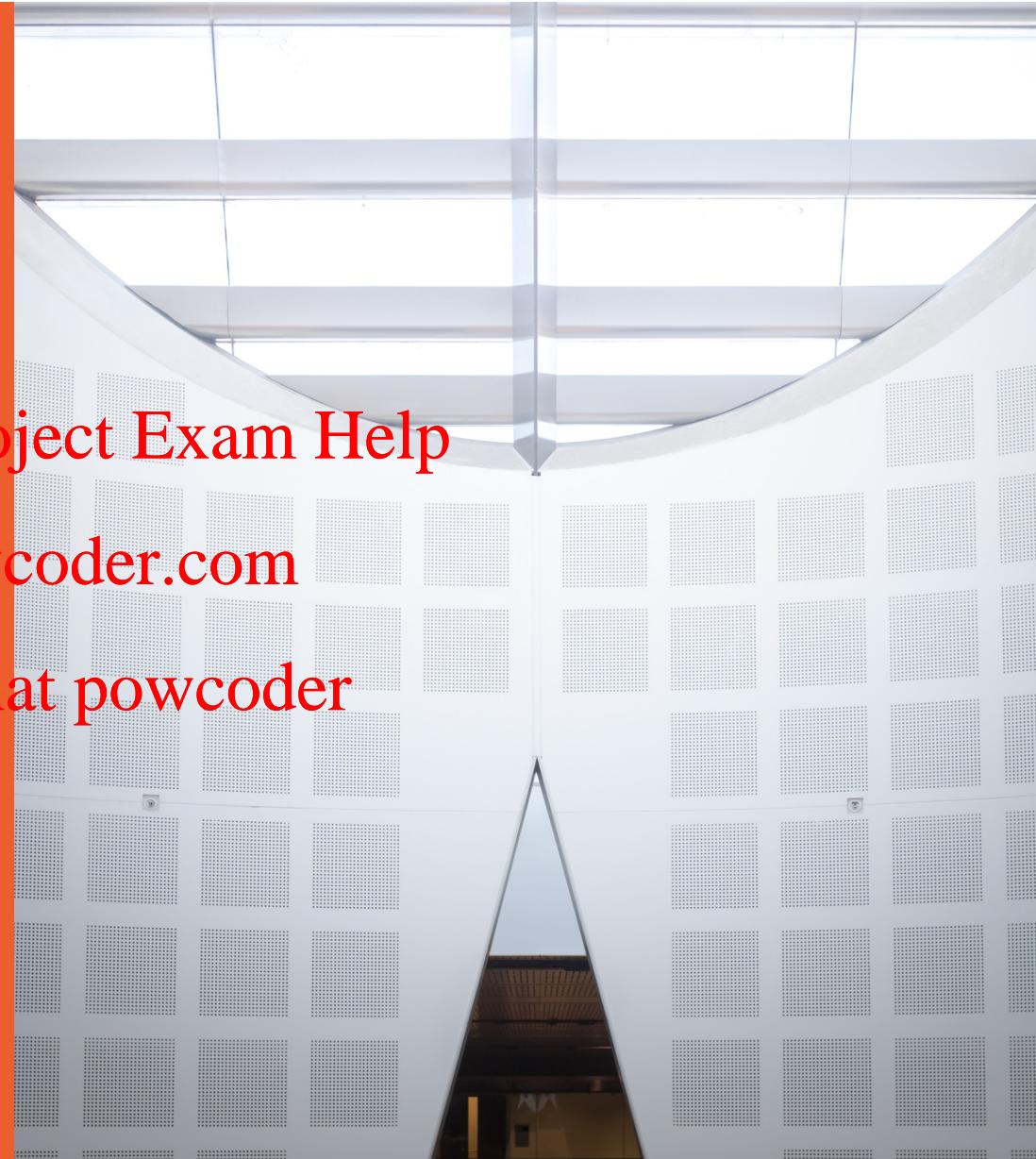
Behavioural Design Patterns <https://powcoder.com>

Dr. Xi Wu

School of Computer Science



Add WeChat powcoder



Copyright warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

Assignment Project Exam Help

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (the Act).

Add WeChat powcoder

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Agenda

- Behavioral Design Patterns
 - Strategy
 - State

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Behavioural Design Patterns

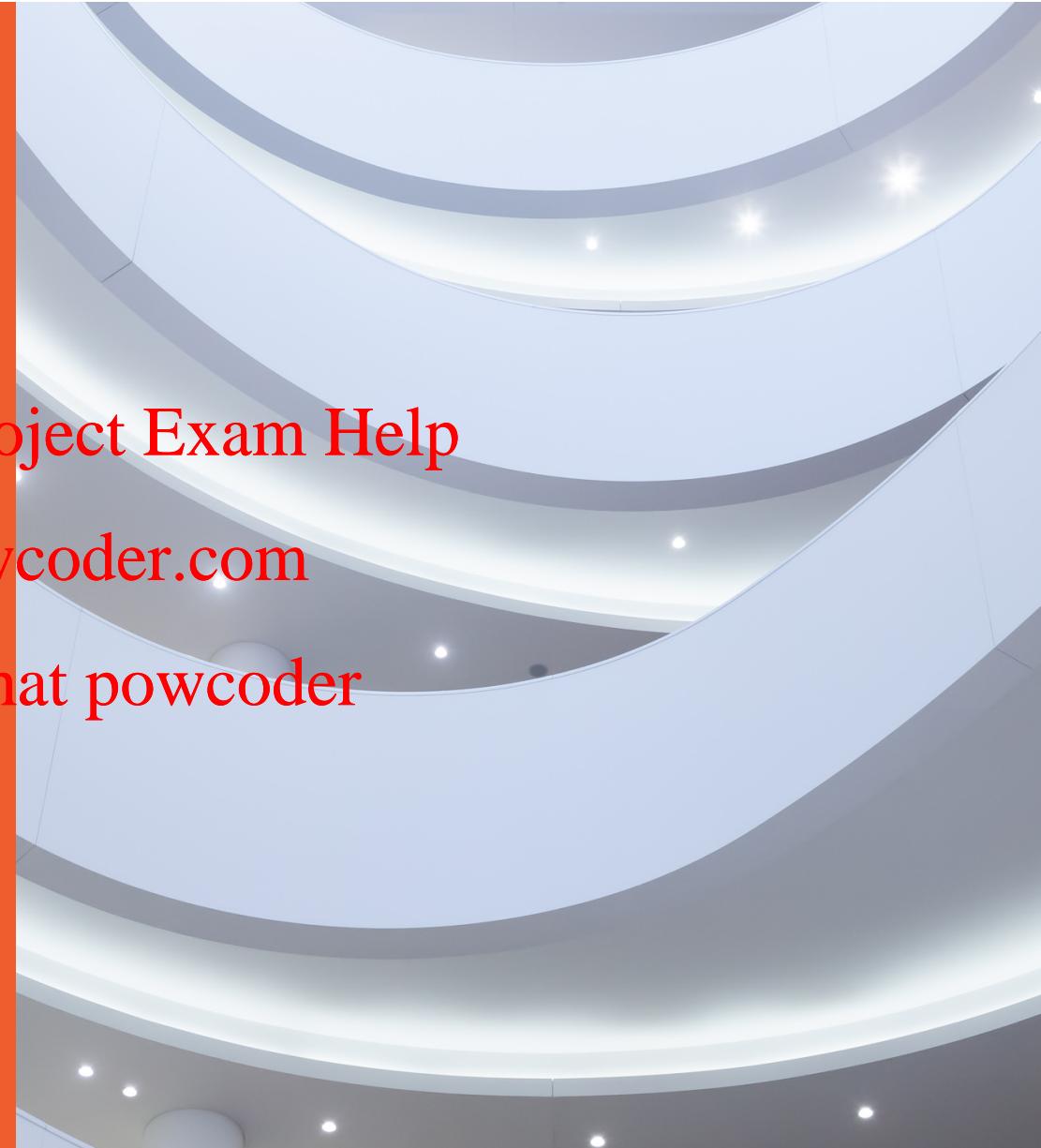
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



THE UNIVERSITY OF
SYDNEY



Behavioural Patterns

- Concerned with algorithms and the assignment of responsibilities between objects
- Describe patterns of objects and class, and communication between them
 - Simplify complex control flow that's difficult to follow at run-time
 - Concentrate on the way objects are interconnected
 - **Behavioural Class Patterns (SOFT3202)**
 - Use inheritance to distribute behavior between classes (algorithms and computation)
 - **Behavioural Object Patterns**
 - Use object composition, rather than inheritance. E.g., describing how group of peer objects cooperate to perform a task that no single object can carry out by itself
 - Question: how peer objects know about each other?

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Behavioural Patterns (GoF)

Pattern Name	Description
Strategy	Define a family of algorithms, encapsulate each one, and make them interchangeable (let algorithm vary independently from clients that use it)
Observer	Define a one-to-many dependency between objects so that when one object changes, all its dependents are notified and updated automatically
Memento	Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later
Command	Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations
State	Allow an object to alter its behaviour when its internal state changes. The object will appear to change to its class
Visitor	Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates
Other patterns	Interpreter, Iterator, Mediator, Chain of Responsibility, Template Method

Strategy Design Pattern

Assignment Project Exam Help

Object Behavioural Pattern

Algorithm design through encapsulation

<https://powcoder.com>

Add WeChat powcoder



Strategy Design Pattern

- **Purpose/Intent**
 - Define a family of algorithms, encapsulate each one, and make them interchangeable
 - Let the algorithm vary independently from clients that use it
- **Known as** <https://powcoder.com>
 - Policy
- **Motivation Scenario** Add WeChat powcoder
 - Design for varying but related algorithms that are suitable for different contexts
 - Ability to change these algorithms

Strategy – Applicability

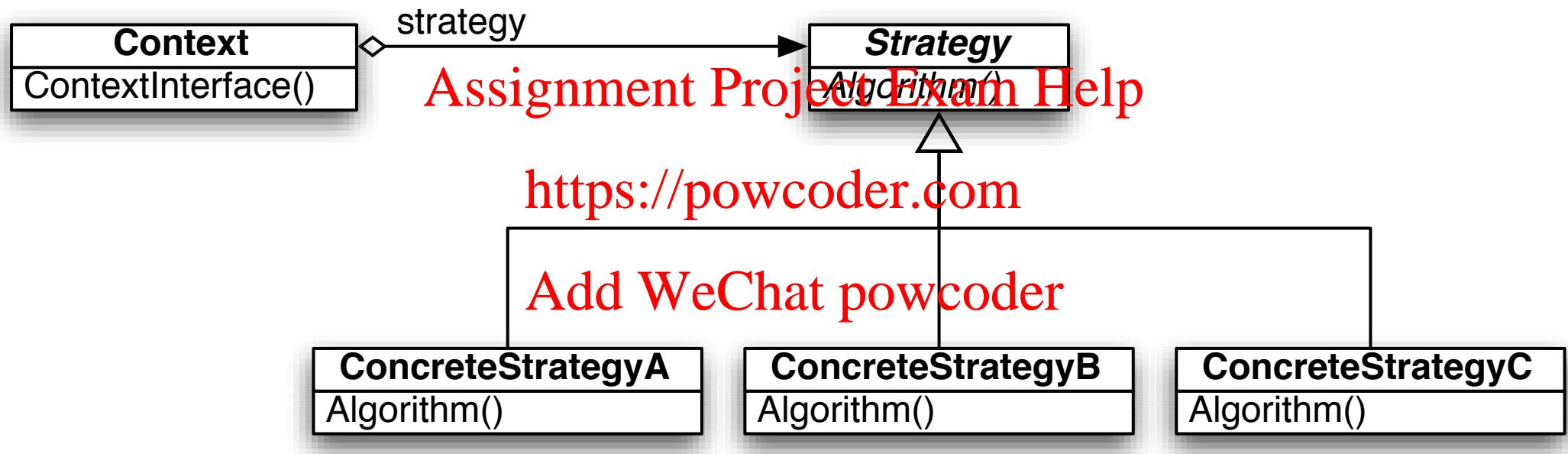
- Many related classes differ only in their behavior
- You need different variants of an algorithm
 - An algorithm uses data that should be hidden from its clients
 - A class defines many behaviors that appear as multiple statements in its operations

Strategy – Example (Text Viewer)

- Many algorithms for breaking a stream of text into lines
- Problem: hard-wiring all such algorithms into the classes that require them
 - More complex and harder to maintain clients (more line breaking algorithms)
 - Not all algorithms will be needed at all times

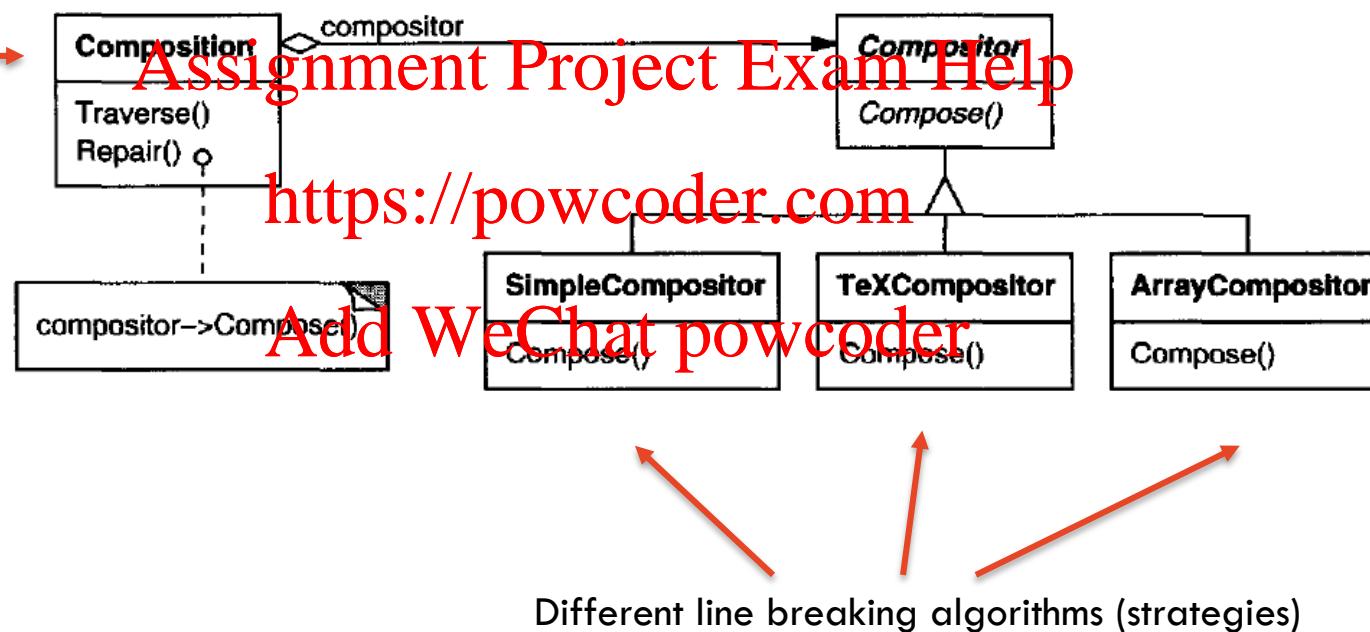
Add WeChat powcoder

Strategy – Structure



Strategy – Example (Text Viewer)

Maintain &
update the line
breaks of text



Strategy – Participants

Participant	Goals
Strategy (Compositor)	Declares an interface common to all supported algorithms Used by Context to get the algorithm defined by ConcreteStrategy
ConcreteStrategy (SimpleComposer, TeXComposer, etc)	https://powcoder.com Implements the algorithm using the Strategy interface Add WeChat powcoder
Context (Composition)	Is configured with a ConcreteStrategy object Maintains a reference to a Strategy object May define an interface that lets Strategy access its data

Strategy – Collaborations

- Strategy and Context interact to implement the chosen algorithm
 - A context may pass all data required by the algorithm to the Strategy
 - The context can pass itself as an argument to Strategy operations

Assignment Project Exam Help

- A context forwards requests from its clients to its strategy
 - Clients usually create and pass a ConcreteStrategy object to the context; thereafter, clients interact with the context exclusively

Add WeChat powcoder

Strategy – Consequences (Benefits)

- Family of related algorithms (behaviors) for context to reuse
- Alternative to sub-classing
 - Why not sub-classing a Context class directly to give it different behaviors?
- Strategies eliminate conditional statements
- Provide choice of different implementation of the same behavior

Strategy – Consequences (Drawbacks)

- Clients must be aware of different strategies
 - Understand how strategies differ
- Communicate overhead between Strategy and Context
 - Strategy interface is shaped by all ConcreteStrategy classes whether the algorithms they implement are trivial or complex

Assignment Project Exam Help

<https://powcoder.com>

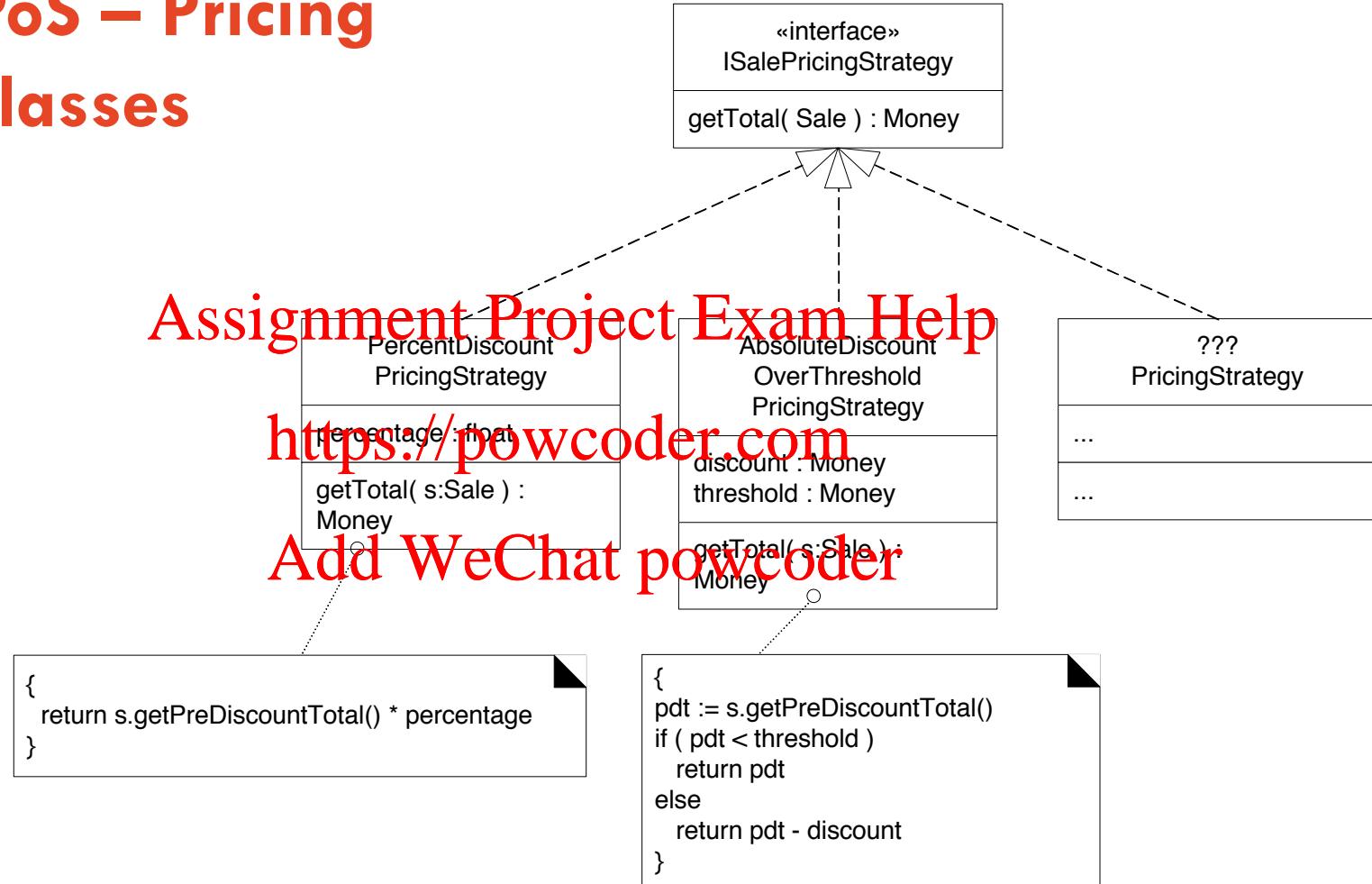
Add WeChat powcoder

- Increased number of objects in an application
 - Can be reduced by implementing strategies as stateless objects that context can share
 - Strategy objects often make good flyweight (sharing strategies)

Strategy – NextGen PoS System

- **Design problem:** how to provide more complex pricing logic, e.g., store-wide discount for the day, senior citizen discounts
- The pricing strategy (or policy) for a sale can vary:
 - 10% of all sales during a specific period
 - \$10 off if the total sale is greater than \$200
 - Other variations
- How do we design our system to accommodate such varying pricing policies?

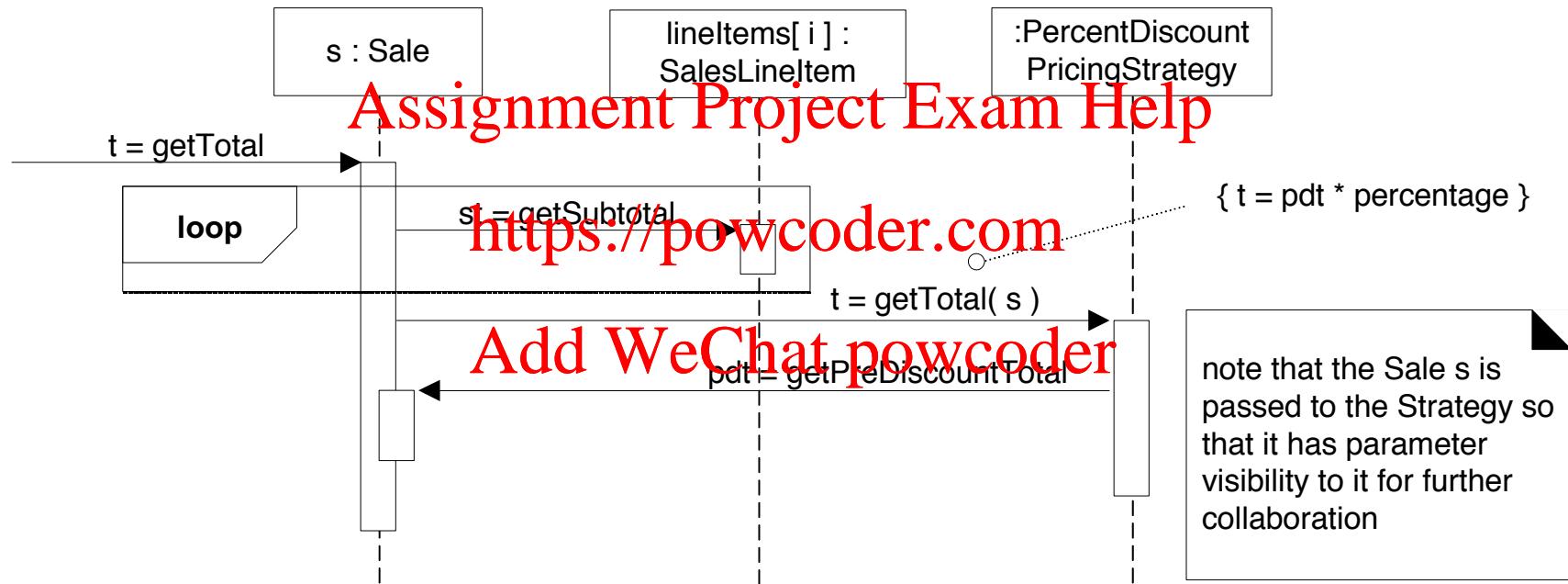
NextGen PoS – Pricing Strategy Classes



Strategy Pattern – NextGen PoS System

- Create multiple *SalesePricingStrategy* classes, each with a polymorphic *getTotal* method
- Each *getTotal* method takes the *Sale* object as a parameter
 - The pricing strategy object can find the pre-discount price from the *Sale*, and they apply the discounting policy
- The implementation of each *getTotal* method will be different
 - E.g., *PercentDiscountPricingStrategy* will discount by a percentage

PoS System – Strategy POS Collaboration



PoS System – Strategy POS Collaboration

- A strategy is attached to the *Sale* object (context object)
- The *Sale* object delegates some of the work to its strategy object
 - The message to the context and strategy objects is not required to be the same (e.g., `getTotal`)
 - The *Sale* object passes a reference to itself on to the strategy object

Add WeChat powcoder

State Design Pattern

Object Behavioural Pattern

Taking control of objects from the inside

A structured way to control the internal behaviour
of an object

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



State Design Pattern

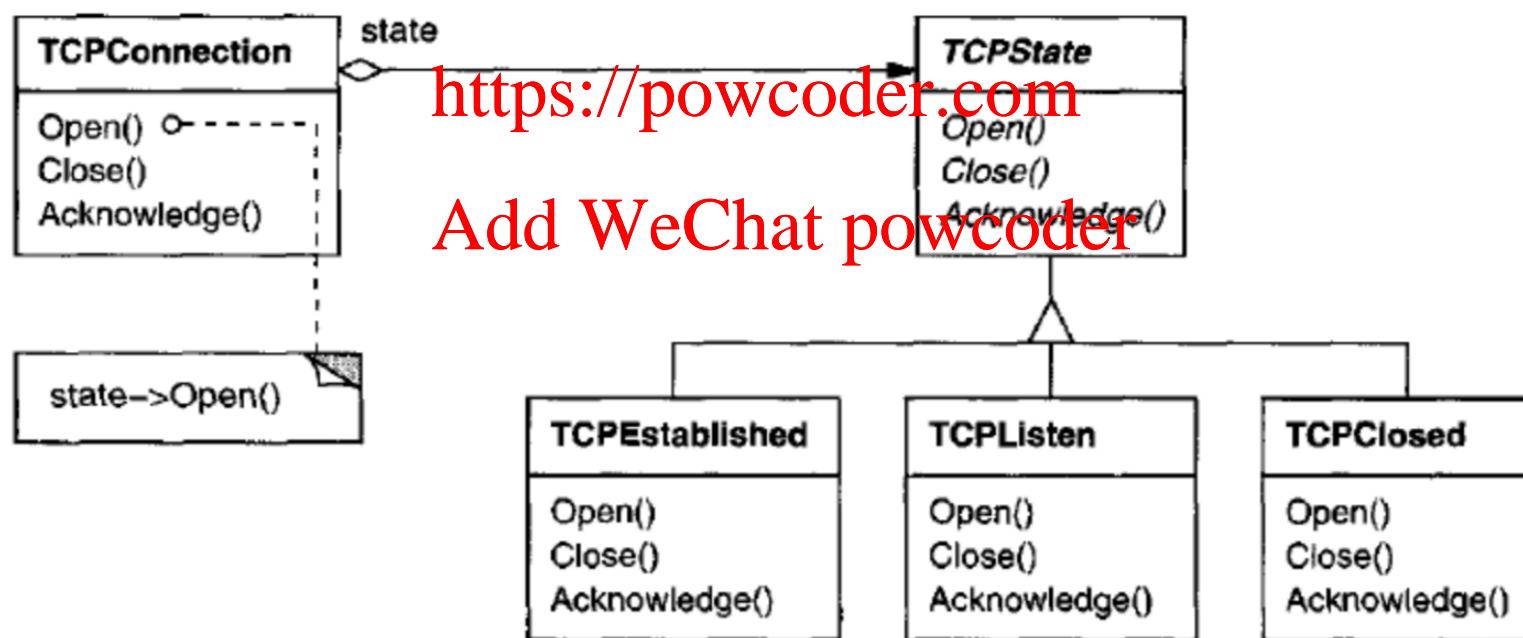
- **Purpose/Intent**
 - Allow an object to change its behaviour when its internal state changes
 - The object will appear to change its class when the state changes

Assignment Project Exam Help

- **Known as**
 - Objects for States <https://powcoder.com>
- **Motivating Scenario** Add WeChat powcoder
 - We can use subtypes of classes with different functionality to represent different states, such as for a TCP connection with Established, Listening, Closed as states

Motivating Scenario

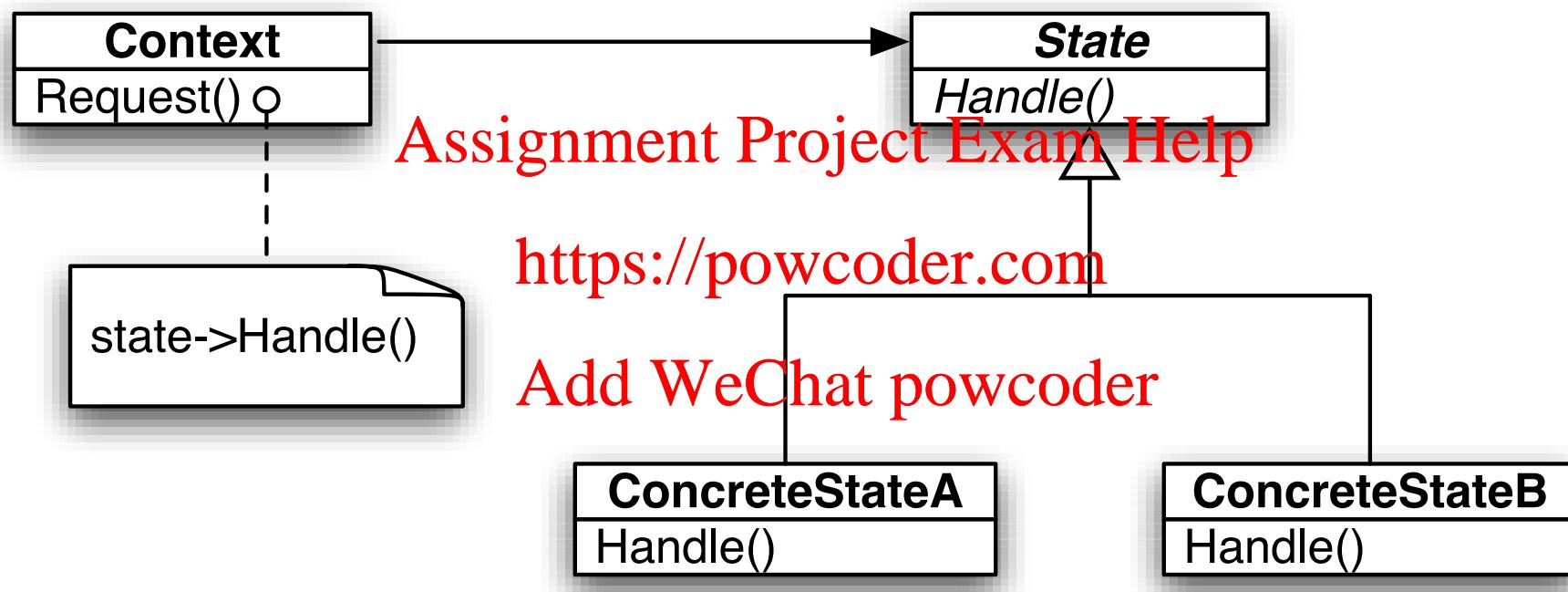
- TCP network connection
- States: Established, Listening and Closed
- TCP connection responds based on its current state



State Design Pattern

- Applicability
 - Any time you need to change behaviours dynamically, i.e., the state of an object drives its behavior and change its behavior dynamically at run-time
 - There are multi-part checks of an object's state to determine its behaviour, i.e., operations have large, multipart conditional statements that depend on the object's state
- Benefits
 - Removes case or if/else statements depending on state, and replaces them with function calls; makes the state transitions explicit; permits states to be shared
- Limitations
 - Does require that all the states have to have their own objects

State Pattern – Structure



State Pattern – Participants

- Context (TCPConnection)
 - Defines the interface of interest to clients
 - Maintains an instance of a ConcreteState subclass that defines the current state
- State (TCPState)
 - Defines an interface for encapsulating the behaviour associated with a certain state of the Context
- ConcreteState subclasses (TCPEstablished, TCPListen, TCPClosed)
 - Each subclass implements a behaviour associated with a state of the Context

State Pattern – Collaborations

- Context delegates state – specific requests to the current ConcreteState object
- A context may pass itself as an argument to the State object handling the request, so the State object access the context if necessary
<https://powcoder.com>
- Context is the primary interface for clients
 - Clients can configure a context with State objects, so its clients don't have to deal with the State objects directly
- Either Context or the ConcreteState subclasses can decide which state succeeds another and under what circumstances

State Pattern – Code Example

```
1 interface State {  
2     void writeName(StateContext context, String name);  
3 }  
4  
5 class LowerCaseState implements State {  
6     @Override  
7     public void writeName(StateContext context, String name) {  
8         System.out.println(name.toLowerCase());  
9         context.setState(new MultipleUpperCaseState());  
10    }  
11 }  
12  
13 class MultipleUpperCaseState implements State {  
14     /* Counter local to this state */  
15     private int count = 0;      Add WeChat powcoder  
16  
17     @Override  
18     public void writeName(StateContext context, String name) {  
19         System.out.println(name.toUpperCase());  
20         /* Change state after StateMultipleUpperCase's writeName() gets invoked twice */  
21         if(++count > 1) {  
22             context.setState(new LowerCaseState());  
23         }  
24     }  
25 }
```

Assignment Project Exam Help

<https://powcoder.com>

State Pattern – Code Example

```
27 class StateContext {  
28     private State state;  
29  
30     public StateContext() {  
31         state = new LowerCaseState();  
32     }  
33  
34     /**  
35      * Set the current state.  
36      * Normally only called by classes implementing the State interface.  
37      * @param newState the new state of this context  
38      */  
39     void setState(State newState) {  
40         state = newState;  
41     }  
42  
43     public void writeName(String name) {  
44         state.writeName(this, name);  
45     }  
46 }
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

State Pattern – Code Example

```
48 public class StateDemo {  
49     public static void main(String[] args) {  
50         var context = new StateContext();  
51  
52         context.writeName("Monday");  
53         context.writeName("Tuesday");  
54         context.writeName("Wednesday");  
55         context.writeName("Thursday");  
56         context.writeName("Friday");  
57         context.writeName("Saturday");  
58         context.writeName("Sunday");  
59     }  
60 }
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

- What is the output of StateDemo?

State Pattern – Consequences

- Localizes state-specific behaviour for different states
 - Using data values and context operations make code maintenance difficult
 - State distribution across different subclasses useful when there are many states
 - Better code structure for state-specific code (better than monolithic)
- It makes state transition explicit
 - State transitions as variable assignments
 - State objects can protect the context from inconsistent state
- State objects can be shared
 - When the state they represent is encoded entirely in their type

State Pattern – Implementation (1)

- Defining the state transitions
 - Let the state subclasses specify their successor state to make the transition (decentralized)
 - Achieves flexibility – easy to modify and extend the logic
 - Introduces implementation dependencies between subclasses

<https://powcoder.com>

- Table-based state transitions
 - Look-up table that maps every possible input to a succeeding state
 - Easy to modify (transition data not the program code) but:
 - Less efficient than a functional call
 - Harder to understand the logic (transition criteria is less explicit)
 - Difficult to add actions to accompany the state transitions

State Pattern – Implementation (2)

- When to create and destroy state objects?
 - Pre-create them and never destroy them
 - Useful for frequent state changes (save costs of re-creating states)
 - Context must keep reference to all states
 - Only when they are needed and destroyed them thereafter
 - States are not known at run-time and context change states frequently

<https://powcoder.com>

Add WeChat powcoder

State Pattern – Implementation (3)

- Using dynamic inheritance
 - Changing the object's class at run-time is not possible in most OO languages **Assignment Project Exam Help**
 - delegation-based languages allow changing the delegation target at run-time
<https://powcoder.com>

Add WeChat powcoder

References

- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
1995. *Design Patterns: Elements of Reusable Object-Oriented Software*.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

<https://powcoder.com>

Add WeChat powcoder

Task for Week 6

- Submit weekly exercise on canvas before 23.59pm Sunday
- Well organize time and continue assignment 2
- Attend Helpdesk session if you have any questions/difficulties on implementation perspective
[Assignment Project Exam Help
https://powcoder.com](https://powcoder.com)

Add WeChat powcoder

What are we going to learn on week 7?

- Testing
- Not next week but the week after next week
 - Next week is mid-break week <https://powcoder.com>
 - no lecture and tutorials during mid-break week but Helpdesk session runs as normal at AddWeChat [powcoder](https://powcoder.com)