Design Patterns

宋 杰

Song Jie

东北大学 软件学院

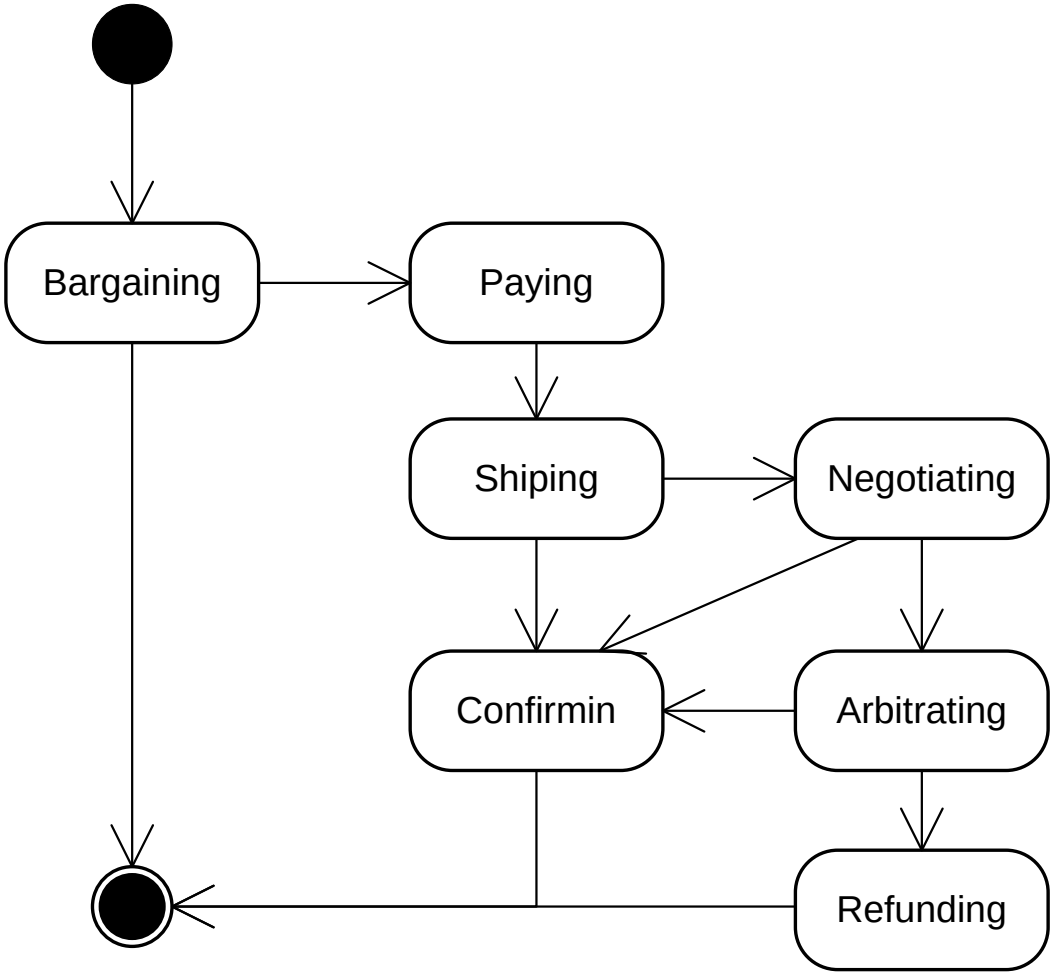
Software College, Northeastern University

14. State Pattern

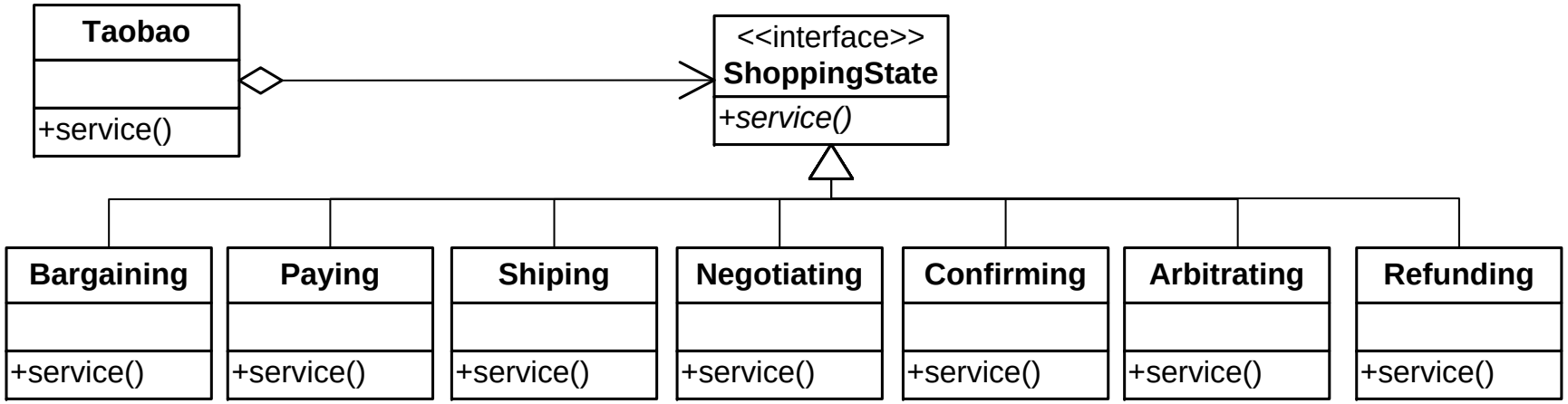
Intent

- Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
- 状态模式允许一个对象在其内部状态改变的时候改变其行为。这个对象看上去就像改变了它的类一样。

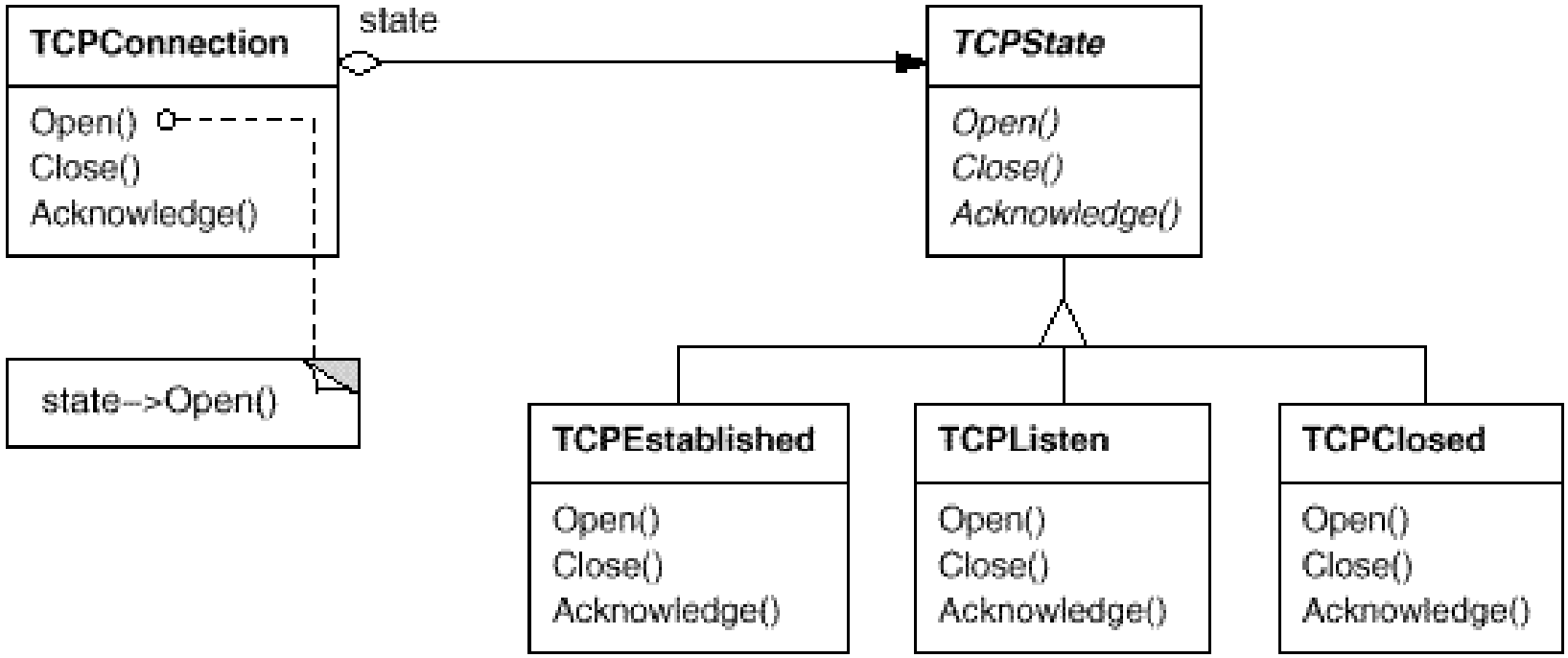
Example: Shopping in Taobao



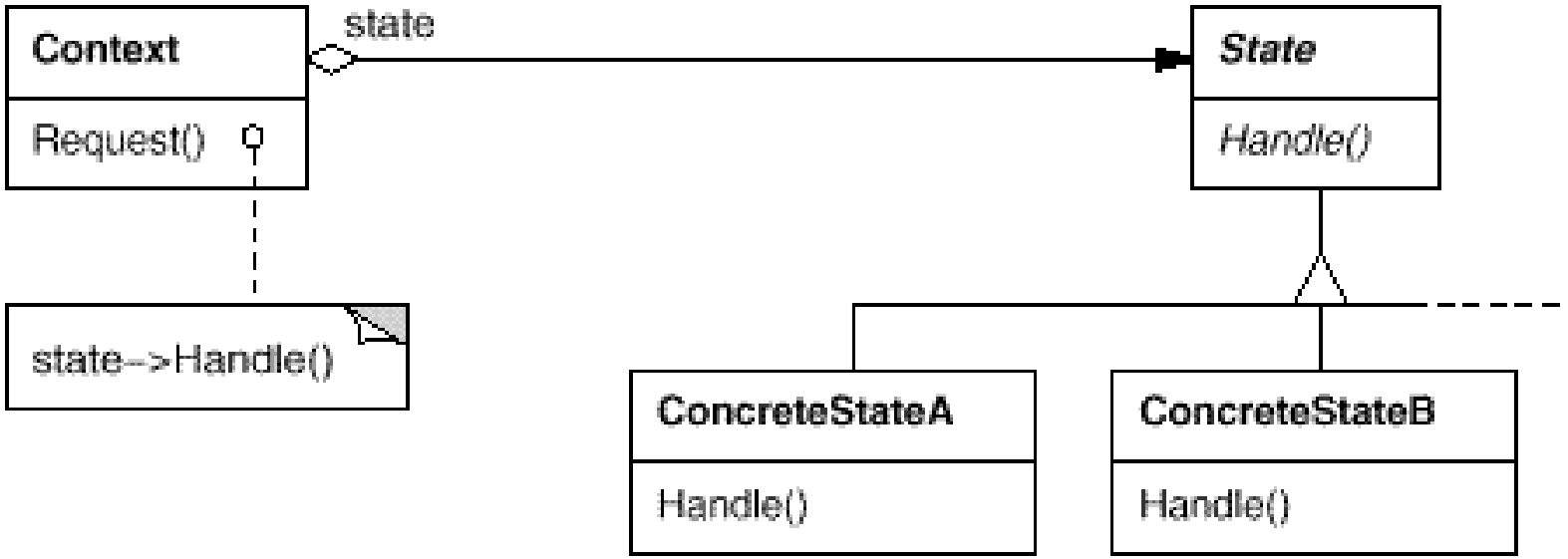
Example: Shopping in Taobao



Example: TCP Connection



Structure


Context
Request()
state
State
Handle()
state->Handle()
ConcreteStateA
Handle()
ConcreteStateB
Handle()

Participants

- Context: Defines the interface of interest to clients; maintains an instance of a ConcreteState that defines the current state.
- State: Defines an interface for encapsulating the behavior associated with a particular state of the Context.
- ConcreteState: each implements a behavior associated with a state of the Context.

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. This lets the State object access the context if necessary.
- Context is the primary interface for clients. State objects can be configured to context. Once a context is configured, its clients don't have to deal with the State objects directly.
- Either Context or the ConcreteState can decide which state succeeds another and under what circumstances.

Consequences

- It localizes state-specific behavior and partitions behavior for different states.
  - New states and transitions can be added easily by defining new subclasses;
  - Avoiding large conditional statements which are undesirable.
  - It increases the number of classes and is less compact than a single class. But such distribution is actually good if there are many states.

Consequences

- It makes state transitions explicit.
  - When an object defines its current state by internal data values, its state transitions have no explicit representation; they only show up as assignments to some variables.
- State objects can be shared (Flyweight).

Applicability

- An object's behavior depends on its state, and it must change its behavior at run-time depending on that state.
- Operations have large, multipart conditional statements that depend on the object's state.

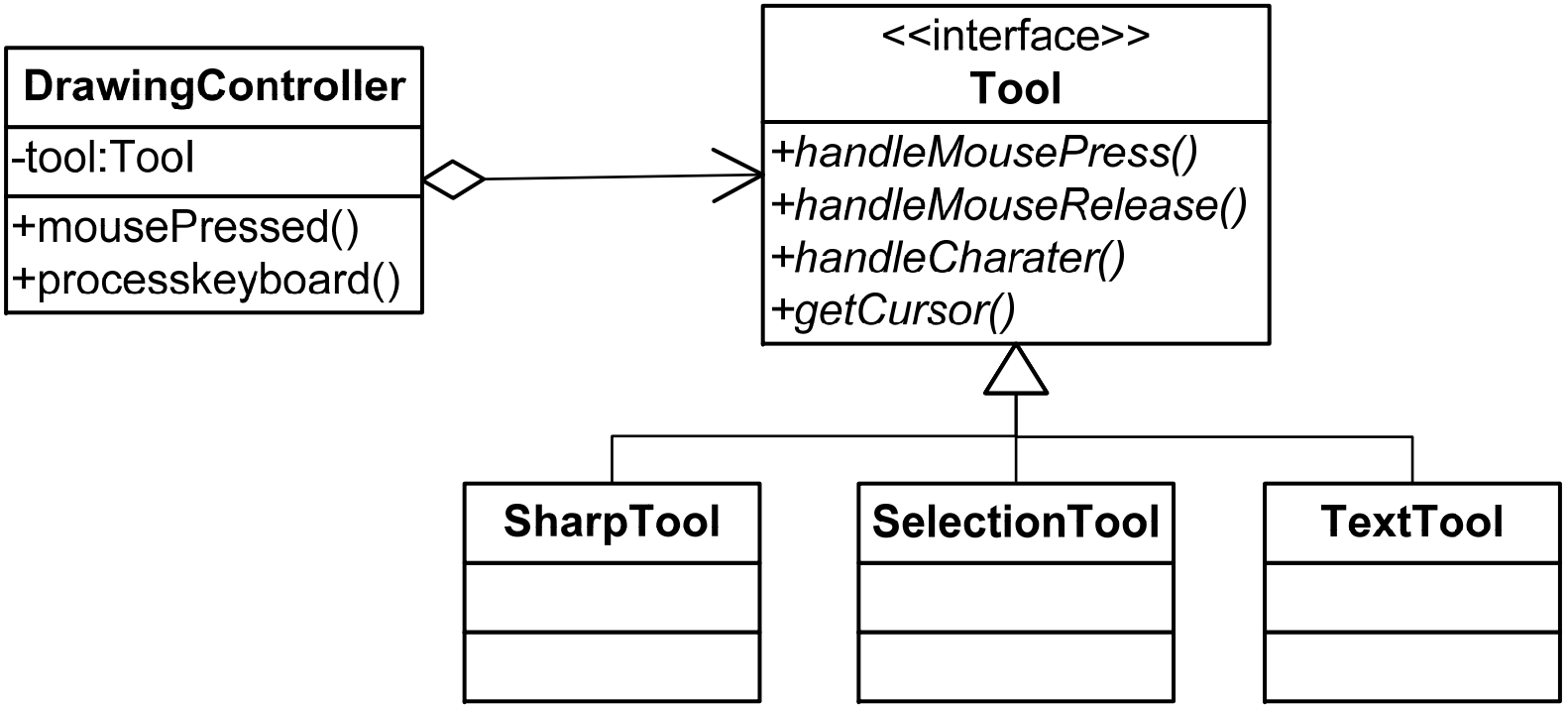
Implementation 1: Who defines the state transitions?

- The State pattern does not specify which participant defines the criteria for state transitions.
- If the criteria are fixed, then they can be implemented entirely in the Context.
- It is generally more flexible and appropriate to let the State subclasses themselves specify their successor state and when to make the transition.
  - It is easy to modify or extend the logic by defining new State subclasses.
  - A disadvantage is State subclass will have knowledge of at least one other, which introduces implementation dependencies between subclasses.

Implementation 2: Creating and destroying State objects.

- A common implementation trade-off worth considering is whether:
  - Lazy: to create State objects only when they are needed and destroy them thereafter.
    - When the states that will be entered aren't known at run-time, and contexts change state infrequently.
  - Eager: creating them ahead of time and never destroying them.
    - When state changes occur rapidly

Examples


DrawingController
-tool:Tool
+mousePressed()
+processkeyboard()
<<interface>>
Tool
+handleMousePress()
+handleMouseRelease()
+handleCharater()
+getCursor()
SharpTool
SelectionTool
TextTool

Extension: Table-driven approach

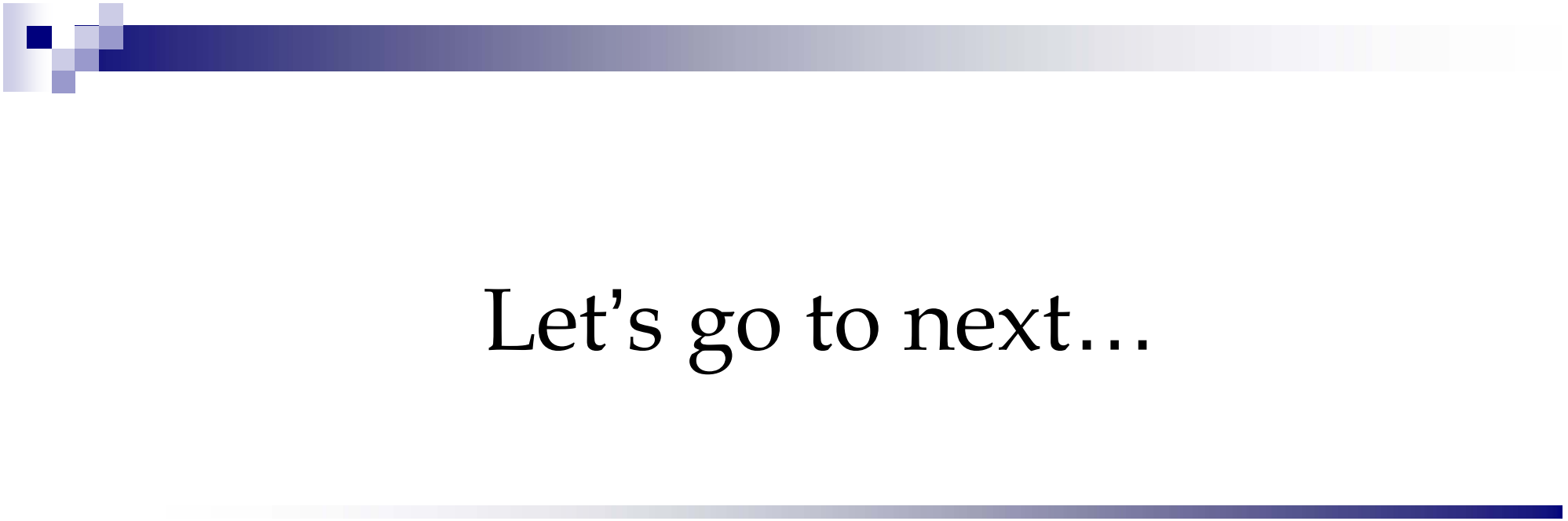
- Using tables to map inputs to state transitions. For each state, a table maps every possible input to a succeeding state.
  - This approach converts conditional code into a table look-up.
- The main advantage of tables is their regularity: You can change the transition criteria by modifying data instead of changing program code.

Extension: Table-driven approach

- Disadvantages
  - A table look-up is often less efficient than a function call.
  - Less explicit and harder to understand.
  - It's usually difficult to add actions to accompany the state transitions.
- The key difference between table-driven and the State pattern
  - The State pattern models state-specific behavior
  - the table-driven approach focuses on defining state transitions.

Related Patterns

- Strategy: One state with many algorithms;
- State: many States with different behaviors.



Let’s go to next…