CS F213 - Object Oriented Programming

J. Jennifer Ranjani

email: jennifer.ranjani@pilani.bits-pilani.ac.in

Chamber: 6121 P, NAB

Consultation: Appointment by e-mail

https://github.com/JenniferRanjani/Object-Oriented-Programming-with-Java

BITS Pilani

Pilani Campus

innovate achieve lead

BITS Pilani
Pilani Campus

Behavioural Design Patterns

- It is about identifying common communication patterns between objects.

Iterator Design Pattern

Intent

- Provides a standard way to access the elements of an aggregate object without exposing its underlying representation.
- Logic for iteration is embedded in the collection itself and it helps the client program to iterate over them easily.
- Eg. Collection framework iterator, Scanner

Steps

- Create a iterator() to the “Collections class” and grant iterated class privileged access
- Design the iterator that can encapsulate the traversal of the collection class
- Clients ask the collection object to create an iterator object
- Clients uses the iterator methods to access the elements of the collection class.

Analogy


sky guide
TV GUIDE
BOX OFFICE
SERVICES
INTERACTIVE
1 ALL CHANNELS
2 ENTERTAINMENT
3 LIFESTYLE & CULTURE
4 MOVIES
5 SPORTS
6 NEWS
7 DOCUMENTARIES
8 KIDS
9 MUSIC
0 MORE...
Planner
A-Z
Favourites
inform/tv.com

State Design Pattern

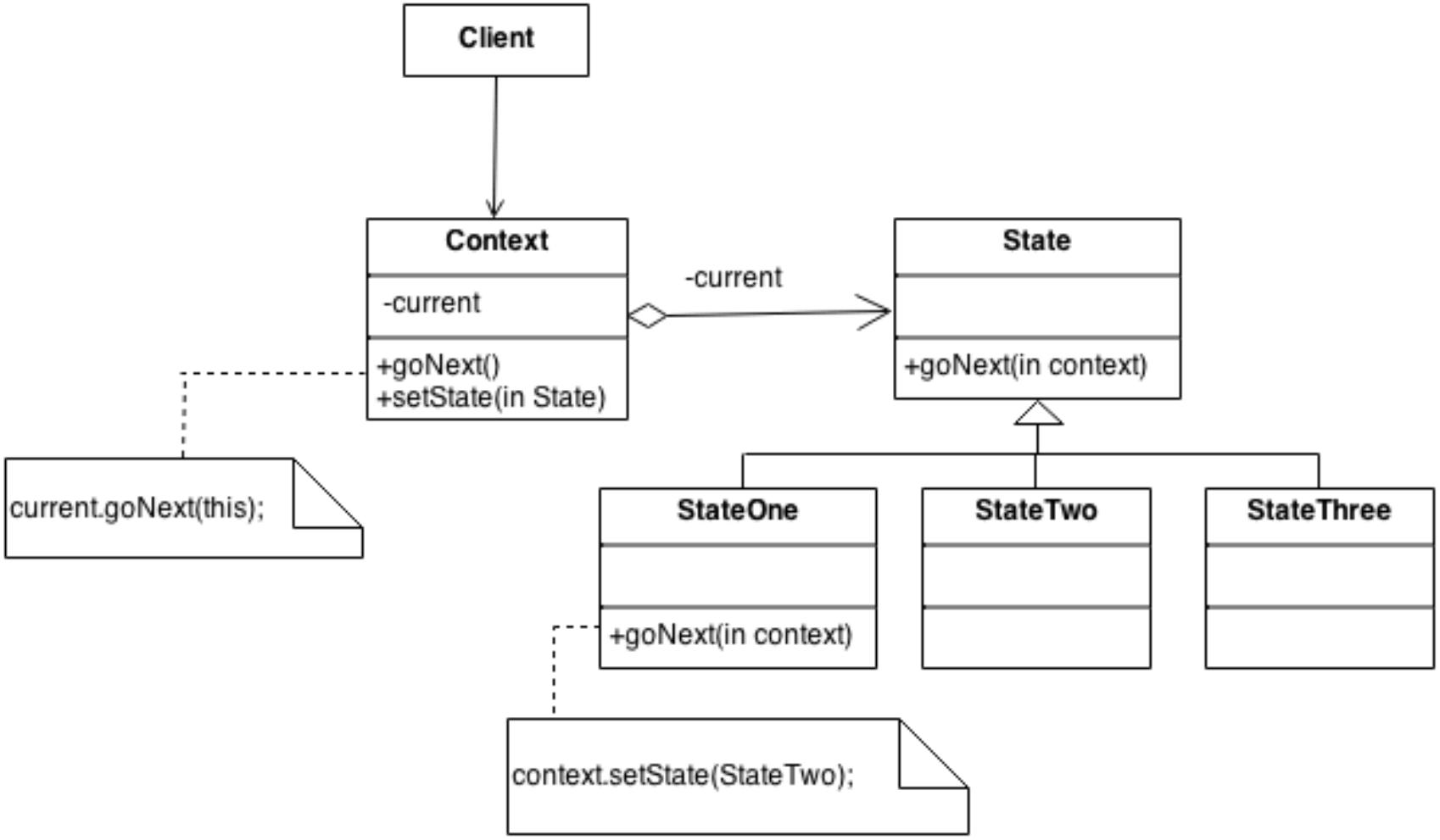
Intent

- Object's behavior is a function of its state.
- Depending on the state the object changes its behavior.
- State pattern allows an object to alter its behavior when its internal state changes.
- Intent is to provide an object oriented state machine.
- State pattern eliminates the client from remembering specific values used for setting the state.

Steps

- Identify an existing class or create a new class, that serves as an state machine for the client
- Create a state base class that replicates the methods of the state machine interface. These methods take instance of the wrapper class (context) as a parameter.
- Create a state derived class for each domain state.
- The wrapper class maintains the current state object.
- The client requests to the wrapper class are delegated to the current state object
- The state methods change the current state appropriately.

State Representation



State Design Pattern

- **When to use?**
  - When object has a relatively complex set of possible states, with many different rules on how transitions occur and what should happen when the state changes.
  - For real world problems with compound workflow, the state structure can be a decent option.
- **Advantages:**
  - Minimizes conditional complexity and eliminates the need for if and switch statements.
  - If the state of the object can be represented using FSM, it is easier to convert the diagram to a state design model.
- **Disadvantages:**
  - Code size is large for a state schema and is dependent on the number of states and the number of transition methods.

Strategy Design Pattern

Analogy

Bulk Cargo

Corrugated Pads

Easy-Fold Mailers

32 ECT Lightweight

Multi-Depth

Storage File

Moving Boxes

Side Loaders

Haz Mat

Wine Shippers

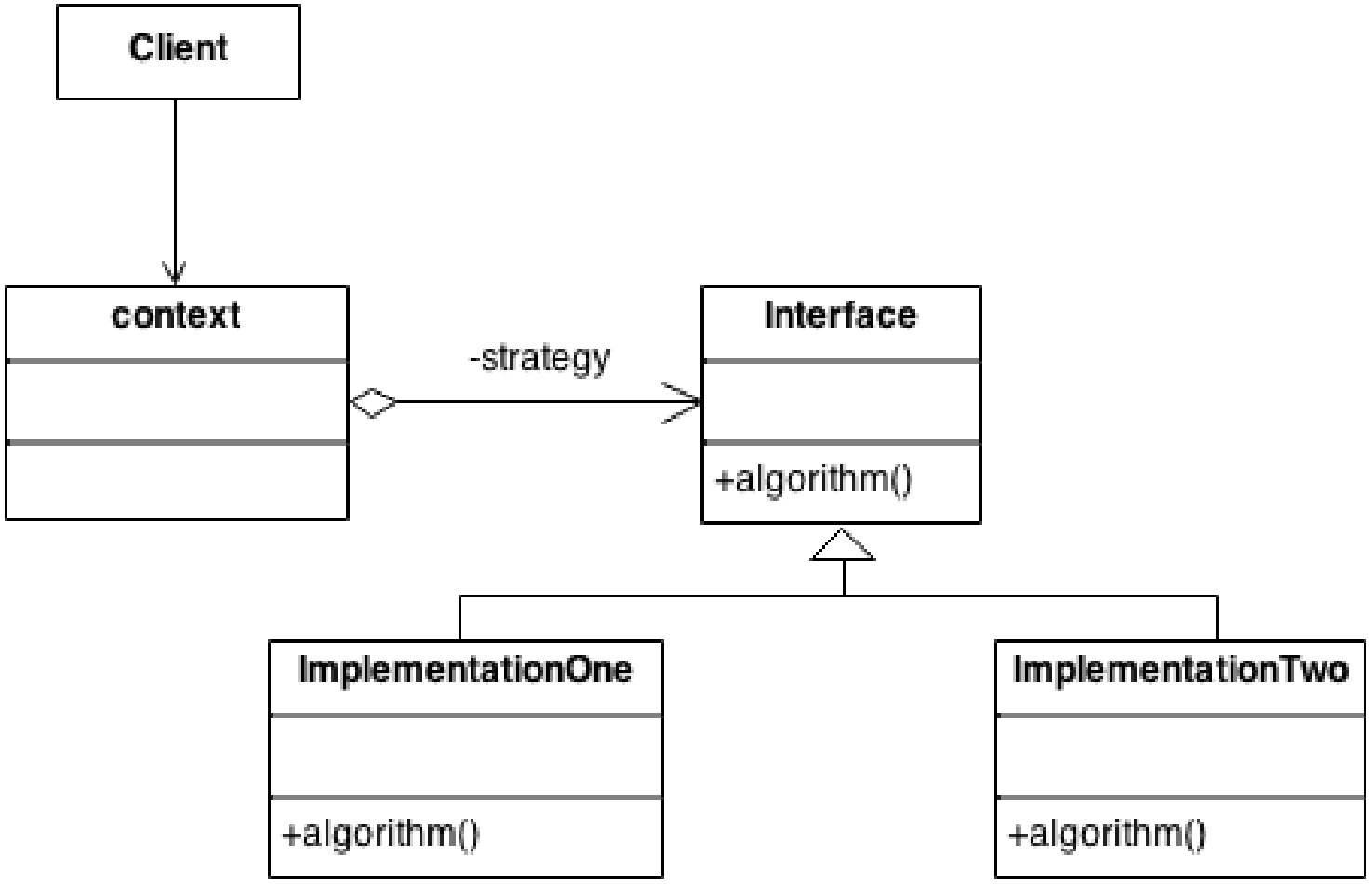
Insulated Shippers

Wood Crates

Intent

- It is ideal when code should programmatically determine which algorithm, function or method should be executed at runtime.
- It lets the algorithm vary independently from the clients that use it.
- Capture the abstraction in an interface, bury implementation details in derived class.

Structure


Client
context
-strategy
Interface
+algorithm()
ImplementationOne
+algorithm()
ImplementationTwo
+algorithm()

Pros and Cons

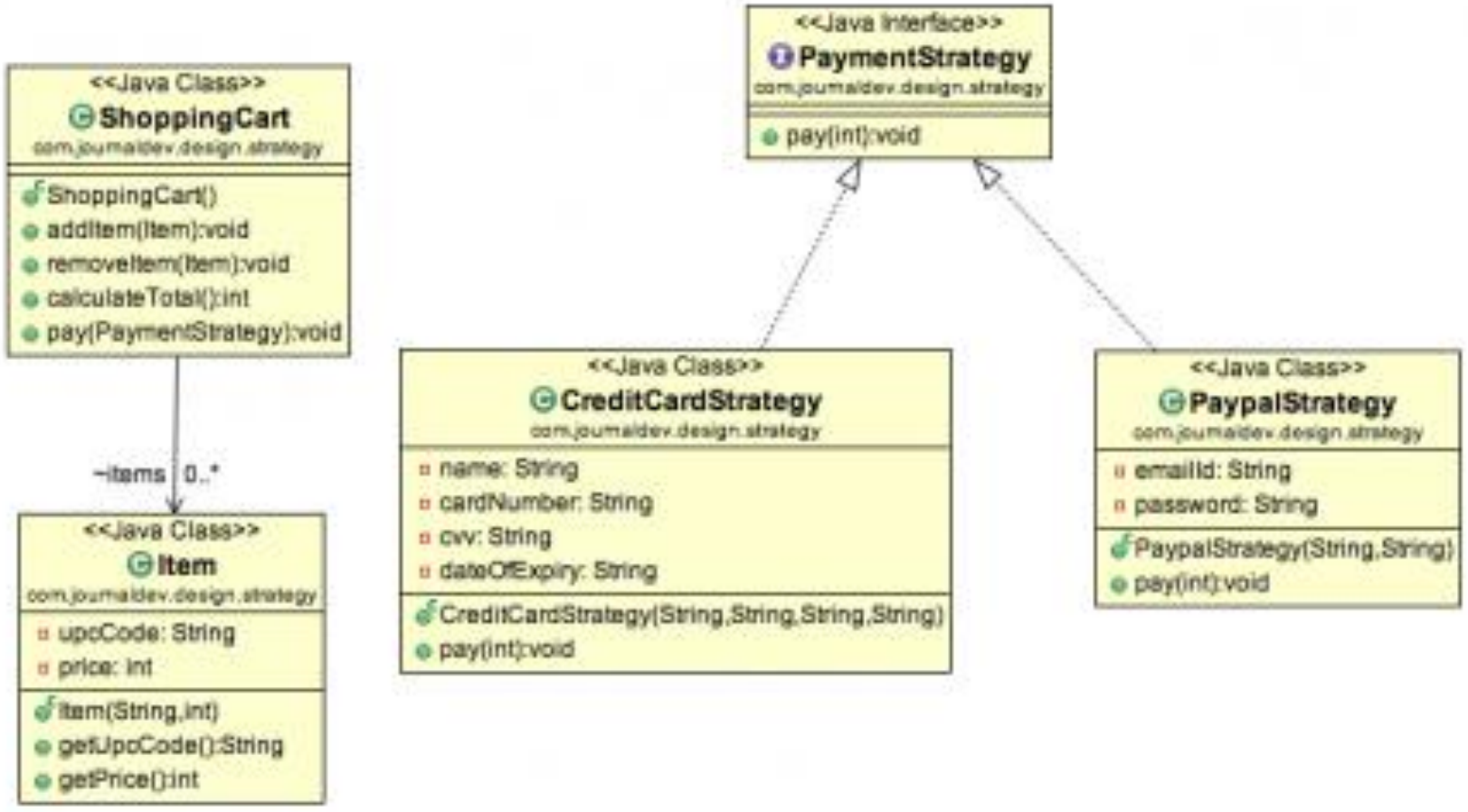
Pros:

- Prevents the conditional statements
- Algorithms are loosely coupled with the context entity. They can be changed/ replaced without changing the context entity.
- Easily extendable.

Cons:

- Clients must know the existence of different strategies and the client must understand how the strategies differ.
- It increases the number of objects.

Example – Class Diagram



Command Design Pattern

Introduction

- It is used to separate a request for an action from the object that actually performs the action.
- It decouples the invoker and the receiver objects and it provides a uniform way to perform different types of action.
- It provides an object oriented callback

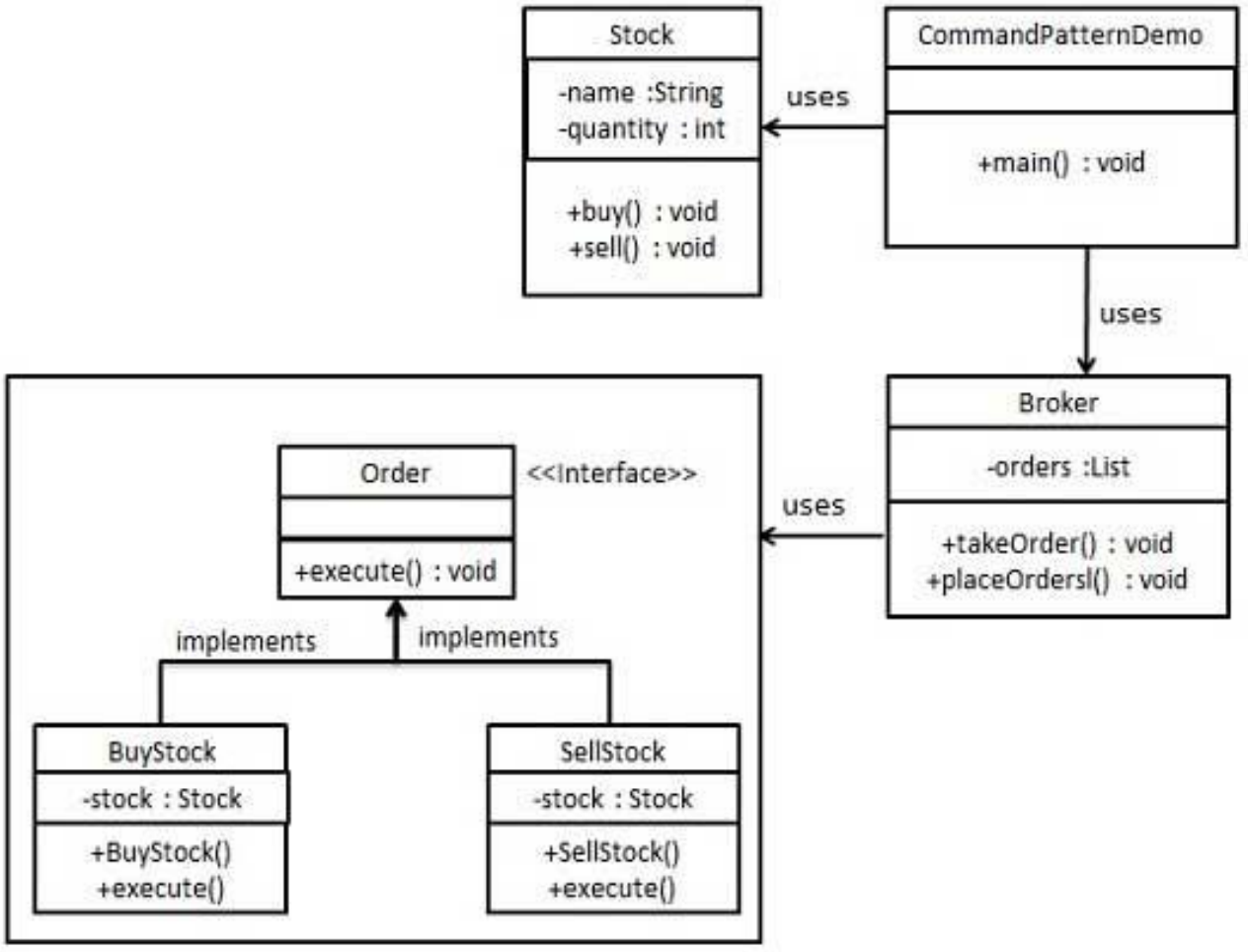
Basic Components

- **Receiver**
  - Receives actions via commands
- **Command**
  - Binds action with the receiver
- **Invoker**
  - Handles a collection of commands and determines when commands are executed.
- **Client**
  - Manages interactions between receiver/command and command/ invoker

Analogy – Postal Service

- **Receiver**
  - Recipient waiting to get some letter
- **Command**
  - Letters await for their time to be executed when delivered to the appropriate recipient.
- **Invoker**
  - Postman who handles the collection of letters and determines when they are delivered
- **Client**
  - Post office as it determines which letters are assigned to which postman

Example – Class Diagram



Observer Design Pattern

Introduction

One of more **Observers** are interested in the state of a **Subject** and register their interest with the subject by **attach**ing themselves. When something changes in the **Subject** the interested **Observers** will be notified, which calls the **update** method. The observer can **detach** himself if he is no longer interested.

Analogy: ?

Sequence diagram