Behavioral Patterns

Sang Shin
"Code with Passion!"

Behavioral Patterns

Patterns that are concerned with communication between objects

- Strategy pattern
- Template method pattern
- Visitor pattern
- Command pattern
- Chain of responsibility pattern
- Observer pattern
- Mediator pattern
- Null object pattern
- Iterator pattern

Strategy Pattern

Strategy Pattern (Policy Pattern)

- What is it?
  - In Strategy pattern, behavior (algorithm) can be changed at runtime
- How to do it?
  - Create objects which represent various strategies and a context object whose behavior varies as per its strategy object
- When to use it?
  - Use it when different behavior (algorithm) needs to be selected during runtime

Example Scenario

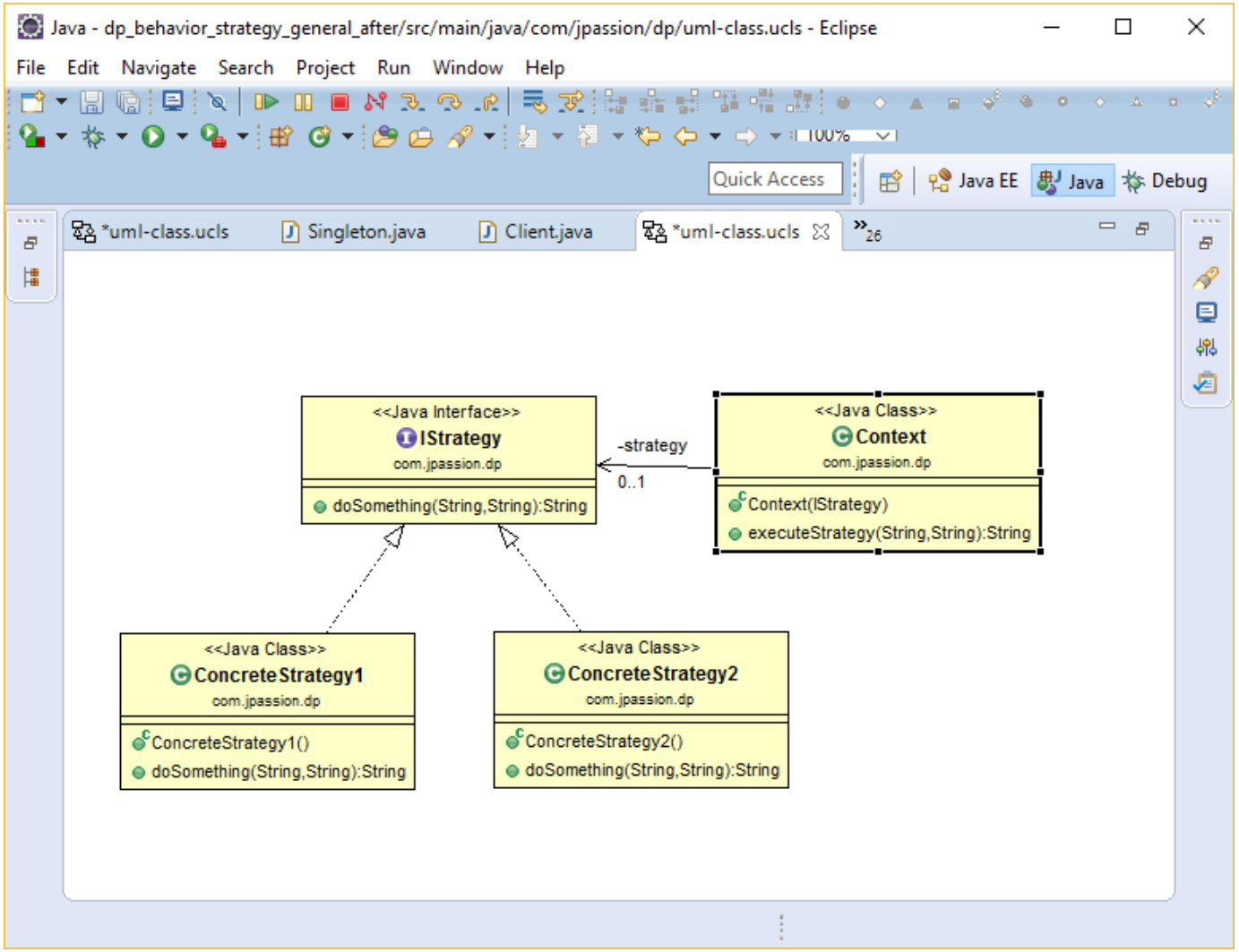
- We are online shopping site
  - We want our customers to be able to use whatever payment scheme of their choice – paypal, credit card, cash

- Future changes (without violating Open-Closed principle)
  - We want to be able to handle new payment scheme: ApplePay, Square, etc

Participants

- Strategy (abstraction – Java interface or abstract class)
  - Declares an interface common to all supported behaviors (algorithms)
  - Context uses this interface to call the behavior (algorithm) implemented by a ConcreteStrategy
- ConcreteStrategies (concrete classes)
  - Implement the concrete behavior (algorithm)
- Context
  - Maintains a reference to a Strategy object
- Client
  - Select a particular concrete strategy during runtime and create a context object with it

Strategy Pattern



Lab:

Exercise 1: Strategy Pattern 9004_dp_behavior.zip

Template Method Pattern

Template method pattern

- What is it?
  - An abstract class that exposes a defined way to execute its methods (i.e., a predetermined sequence of methods) in the form of a template method
  - Its subclasses can override the method implementation as per need, but the invocation is to be in the same way as defined by the template method in the abstract class

- When to use it?
  - Use it to implement the non-changing parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary

Example Scenarios of Template Method

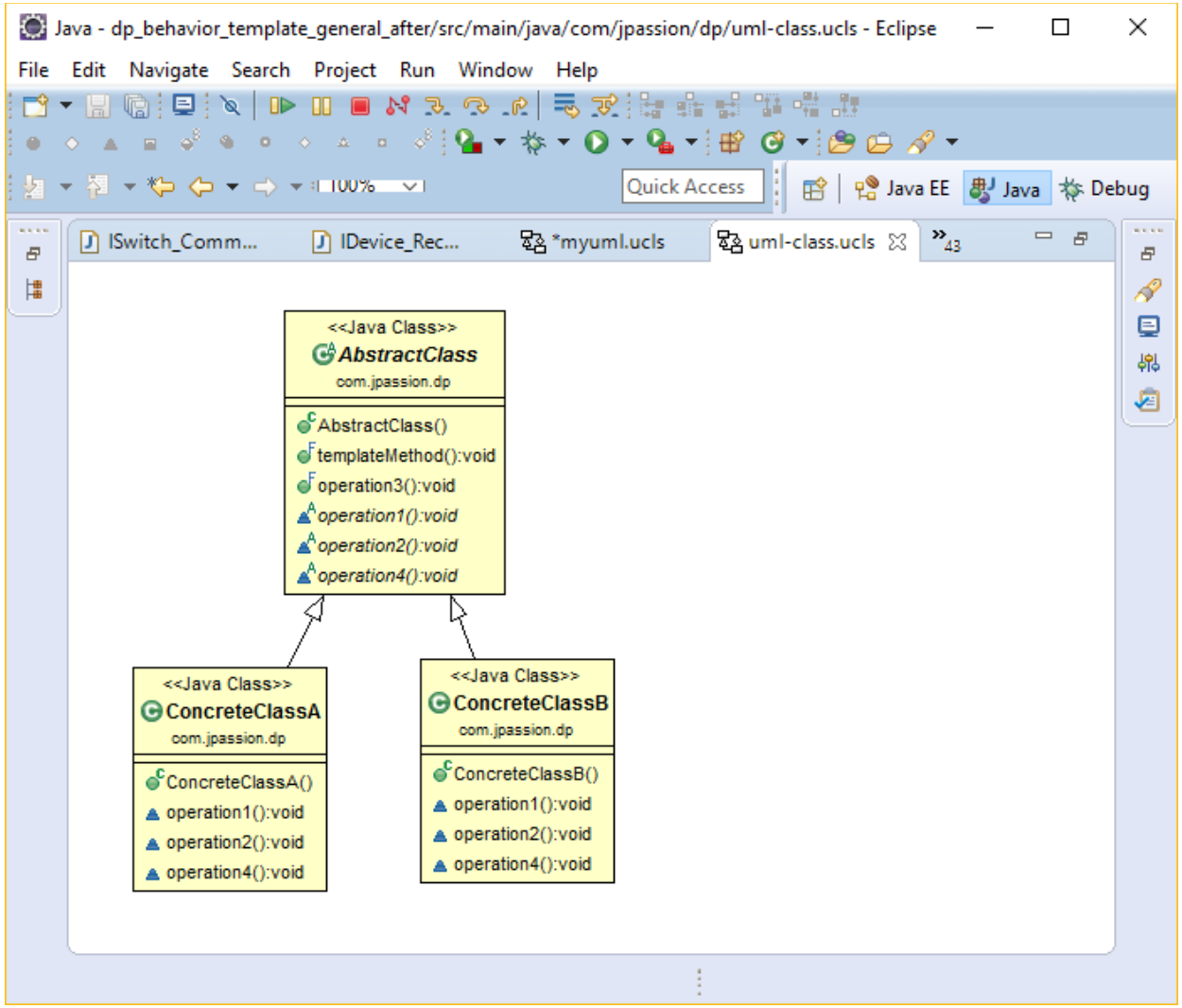
- We are a Pizza company – we have regional Pizza franchises
  - We want to make sure all regional pizza franchises to follow a same sequence of steps (operations) for handling pizza order at the store
  - We want to give them some freedom in handling some steps (operations) – maybe regional franchises want to use regionally flavored ingredients
  - We want to make sure all regional franchises to handle a certain set of operations the same – maybe taking payment should be implemented the same since they are connected to corporate financial system

- Future changes (without violating Open Close principle)
  - We would like to add new regional franchises

Participants

- AbstractClass
  - Define a template method in which a sequence of methods are invoked in a specified way
  - Defines abstract methods that need to be implemented by subclasses
  - Could define concrete methods that can be shared by subclasses
- ConcreteClasses
  - Implement abstract methods of the AbstractClass

Template method pattern



Lab:

Exercise 2: Template Pattern 9004_dp_behavior.zip

Visitor Pattern

Visitor Pattern

- What is it?
  - Visitor pattern allows an operation to be performed (represented by Visitor) on elements of an object structure (represented by Visitable)
  - Visitor lets you define a new operation without changing the classes of the elements on which it operates (Visitable)

- When to use it?
  - Use it when different set of operations need to be performed to elements of an object structure

Example Scenario

- We are Car manufacturer
- Car is made of many parts
  - Engine
  - Body
  - Four wheels
- We need to perform various set of operations against these parts
  - We need to purchase each of them
  - We need to assemble each of them
  - We need to record the price of each of them
- Future changes (without violating Open Close principle)
  - We want to perform another operation to these parts

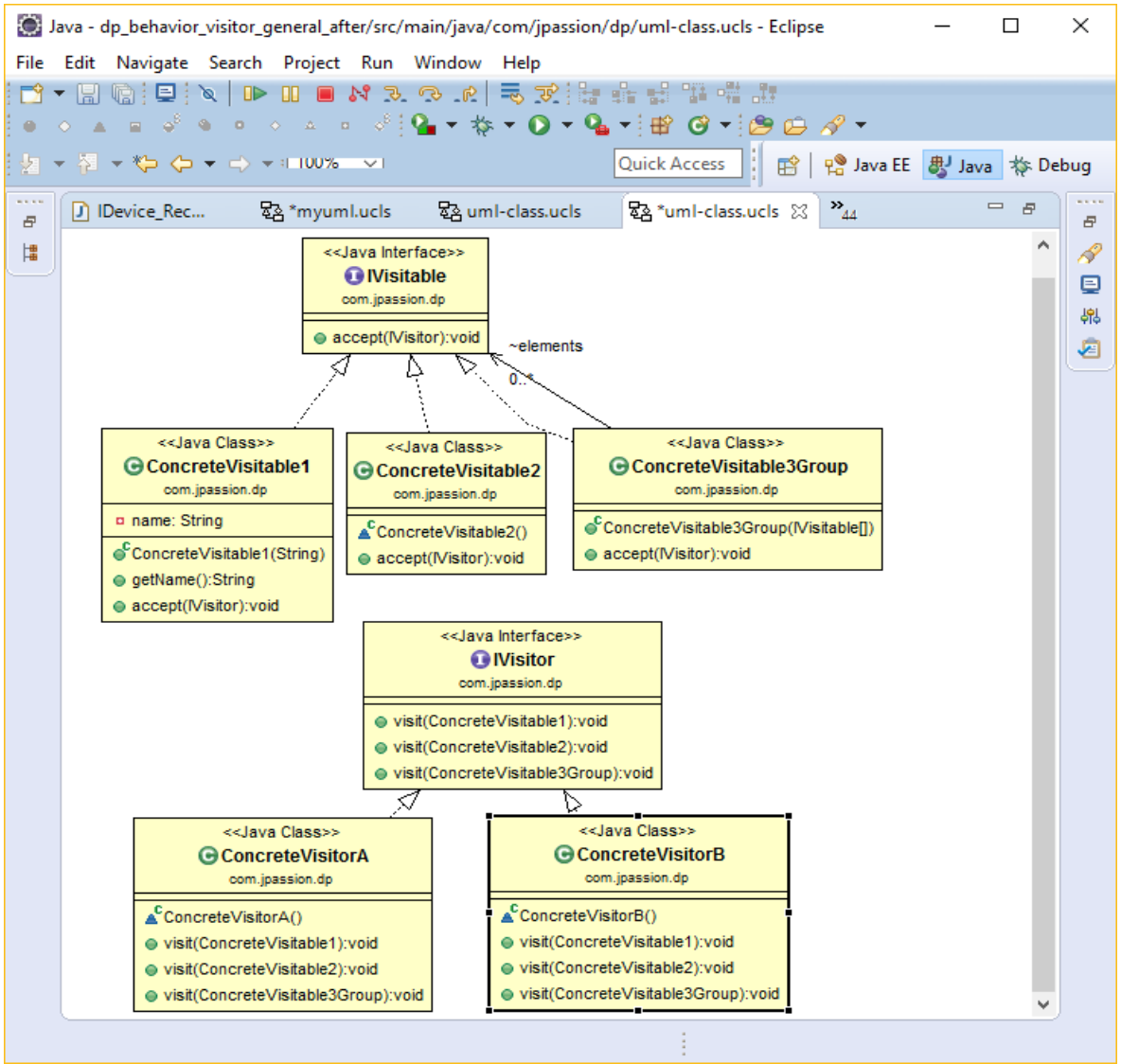
Participants (1)

- Visitor (abstraction)
  - > Declare the visit operations
  - > Usually the name of the operation is the same and the operations are differentiated by the method signature
- ConcreteVisitor's (Concrete class)
  - > Each concrete visitor has a set of operations – different visitor has a different set of operations
  - > A concrete visitor is passed to the object structure (Visitable)
  - > A new concrete visitor can be created and passed to the object structure (Visitable) without forcing change in the object structure – open close principle

Participants (2)

- Visitable (an abstraction)
  - An abstraction which declares the *accept* operation. This is the entry point which enables an object to be "visited" by the visitor object
  - Each object from a collection implements this abstraction in order to be able to be visited
- ConcreteVisitable (Concrete class)
  - Defines the accept operation, through which visitor object is passed
- ConcreteVisitableGroup (Concrete class)
  - This is a class containing all the objects that can be visited. It offers a mechanism to iterate through all the elements
  - This structure is not necessarily a collection. In can be a complex structure, such as a composite object.

Visitor Pattern



Lab:

Exercise 3: Visitor Pattern
9004_dp_behavior.zip

Command Pattern

Command Pattern

- What is it?
  - Encapsulate a request in an object (command object) - In command pattern, a command object is used to encapsulate all information needed to perform an action, which includes the method name, the object that owns the method and values for the method parameters.
  - Allows bookkeeping of the requests
- When to use it?
  - When requests need to be executed in a controlled manner – i.e. as a batch, at a scheduled time, based on conditions, undo a request

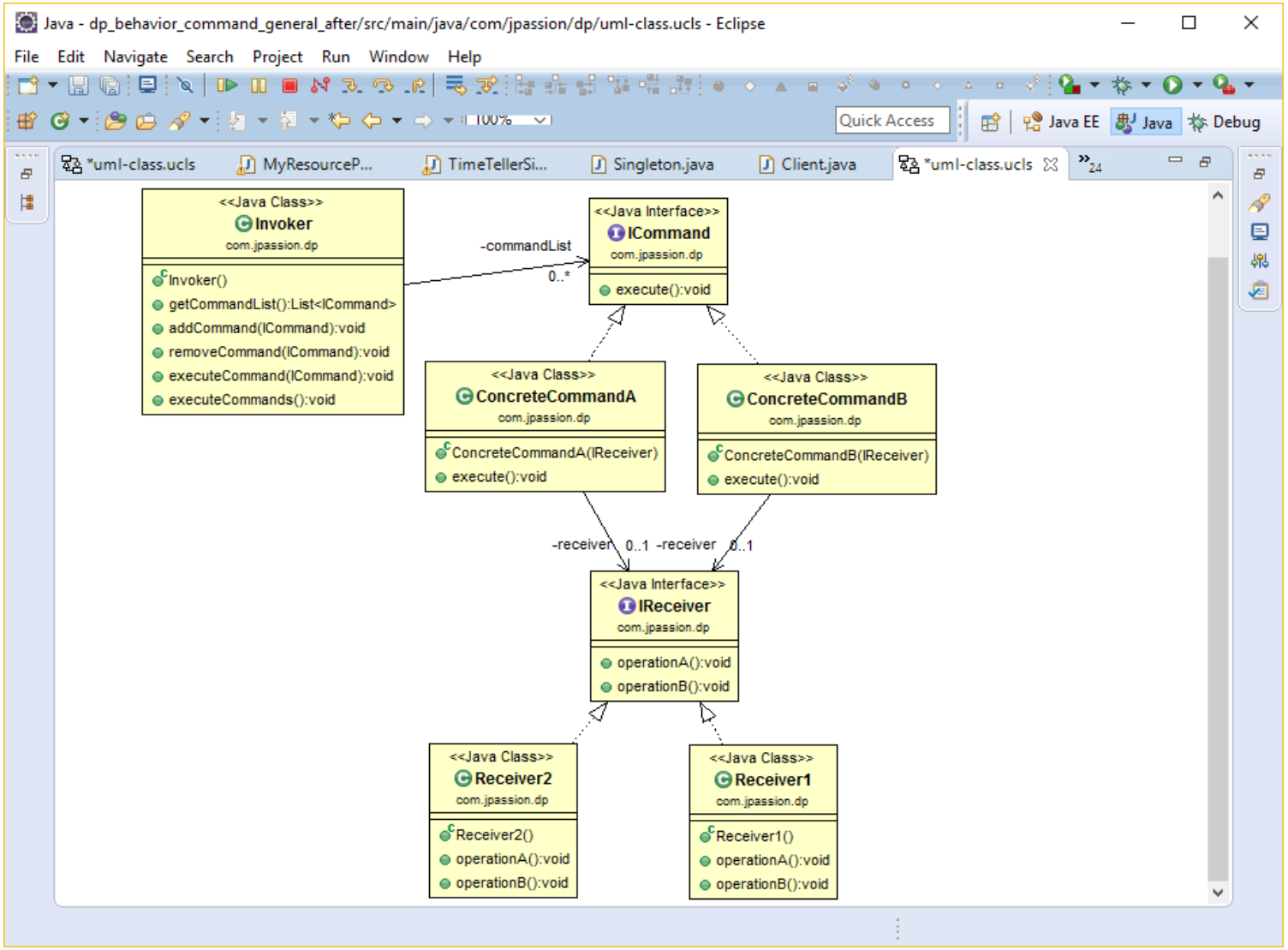
Example Scenario of Command Pattern

- Waiter (or Restaurant) has to take several orders from the customers
  - Drink orders
  - Food orders
- Some orders can be canceled
- Waiter might want to take the drink order first and then food order later or s/he might take all orders at the same time
  - Or restaurant want to prepare these orders individually or in a batch
- In other words, these orders need to be collected first and then might get executed in different execution schemes
- Future changes (without violating Open Close principle)
  - Different schemes (such as mobile orders) need to be handled differently than in-store orders

Participants

- Command (Interface)
  - Declares an interface for executing an operation – execute() method
- ConcreteCommand's (Concrete classes)
  - Encapsulates a concrete request
  - References a Receiver
  - Implementing the *execute()* method of Command interface – the execute(0 method contains code invoking the corresponding operations of the Receiver
- Receiver (Concrete class)
  - Knows how to perform the operations
- Invoker (Concrete class)
  - Does bookkeeping of commands – it plays a role of “command manager”
  - Has a method that asks the command to carry out the request calling execute() method

Command Pattern



Lab:

Exercise 4: Command Pattern 9004_dp_behavior.zip

Chain of Responsibility Pattern

Chain of Responsibility pattern

- What is it?
  - Decouples the sender of a request from its receiver by giving more than one object a chance to handle the request
  - Chain the receiving objects and pass the request along the chain until an object handles it
- When to use it?
  - Use it when a request can be handled by multiple handlers in a chain

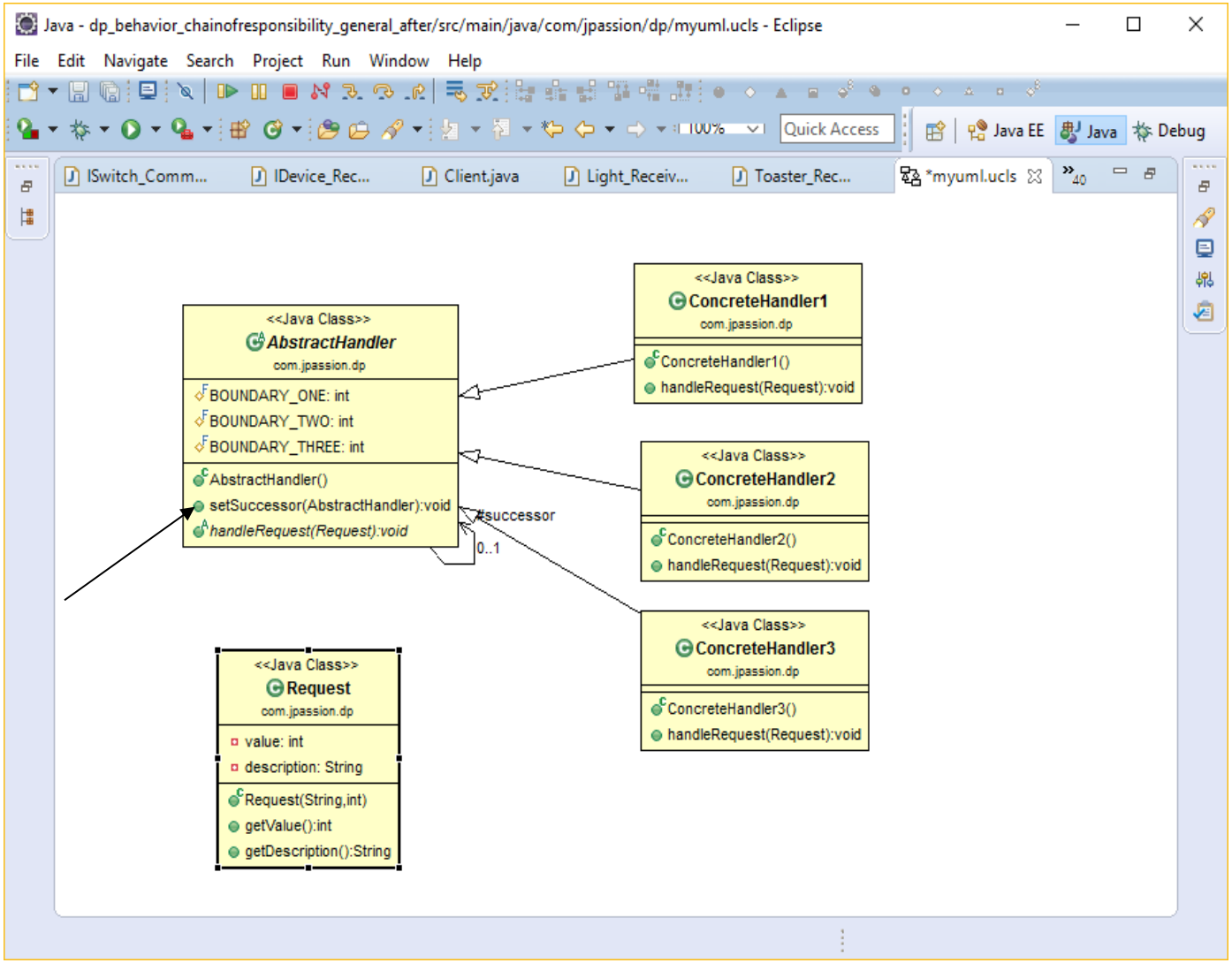
Example Scenario

- A purchase order request should be handled by different roles in a corporation depending on the amount
  - A purchase order of $1000 or less can be handled by 1st level manager
  - A purchase order of $5000 or less can be handled by a Director
  - A purchase order of $10000 or less can be handled by Vice-president
- Future changes (without violating Open Close principle)
  - Add new roles

Participants

- AbstractHandler
  - Defines an interface for handling requests
- ConcreteHandler's
  - Configured with a successor
  - If it can handle the request it does so, otherwise it sends the request to its successor
- Client
  - Sends requests to the first handler in the chain that may handle the request

Chain of Responsibility



Lab:

Exercise 5: Chain of Responsibility Pattern 9004_dp_behavior.zip

Observer Pattern

Observer pattern

- What is it?
  - Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically
- When to use it?
  - Use it to implement distributed event handling systems

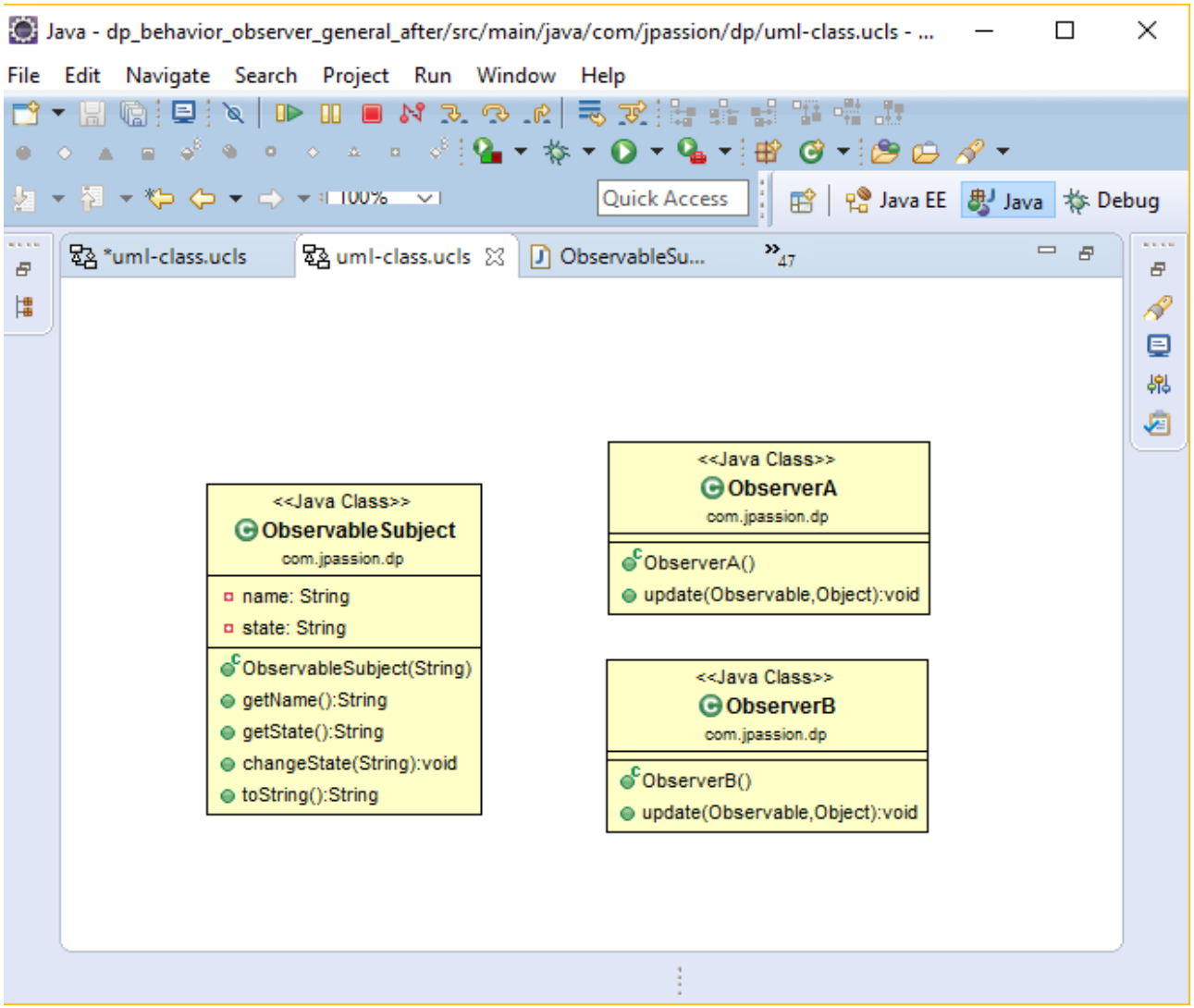
Example Scenario

- When a new employee is hired, various parties need to be notified
  - Payroll department need to be notified
  - Facility department need to be notified so that office and phone can be arranged
- Future changes (without violating Open Close principle)
  - Add another department to be notified

Participants

- Observable
  - Interface or abstract class defining the operations for attaching and deattaching observers to the client.
- ConcreteObservable
  - Concrete Observable class. It maintain the state of the object and when a change in the state occurs it notifies the attached Observers
  - This is called Subject
- Observer
  - Interface or abstract class defining the operations to be used to notify this object
- ConcreteObserver classes
  - Concrete Observer implementations

Observable Pattern



Lab:

Exercise 6: Observer Pattern
9004_dp_behavior.zip

Mediator Pattern

Mediator pattern

- What is it?
  - Define an object (mediator) that encapsulates how a set of objects interact
- When to use it?
  - Use it when loose coupling is desired between communicating objects by keeping objects from referring to each other explicitly and it lets you vary their interaction independently

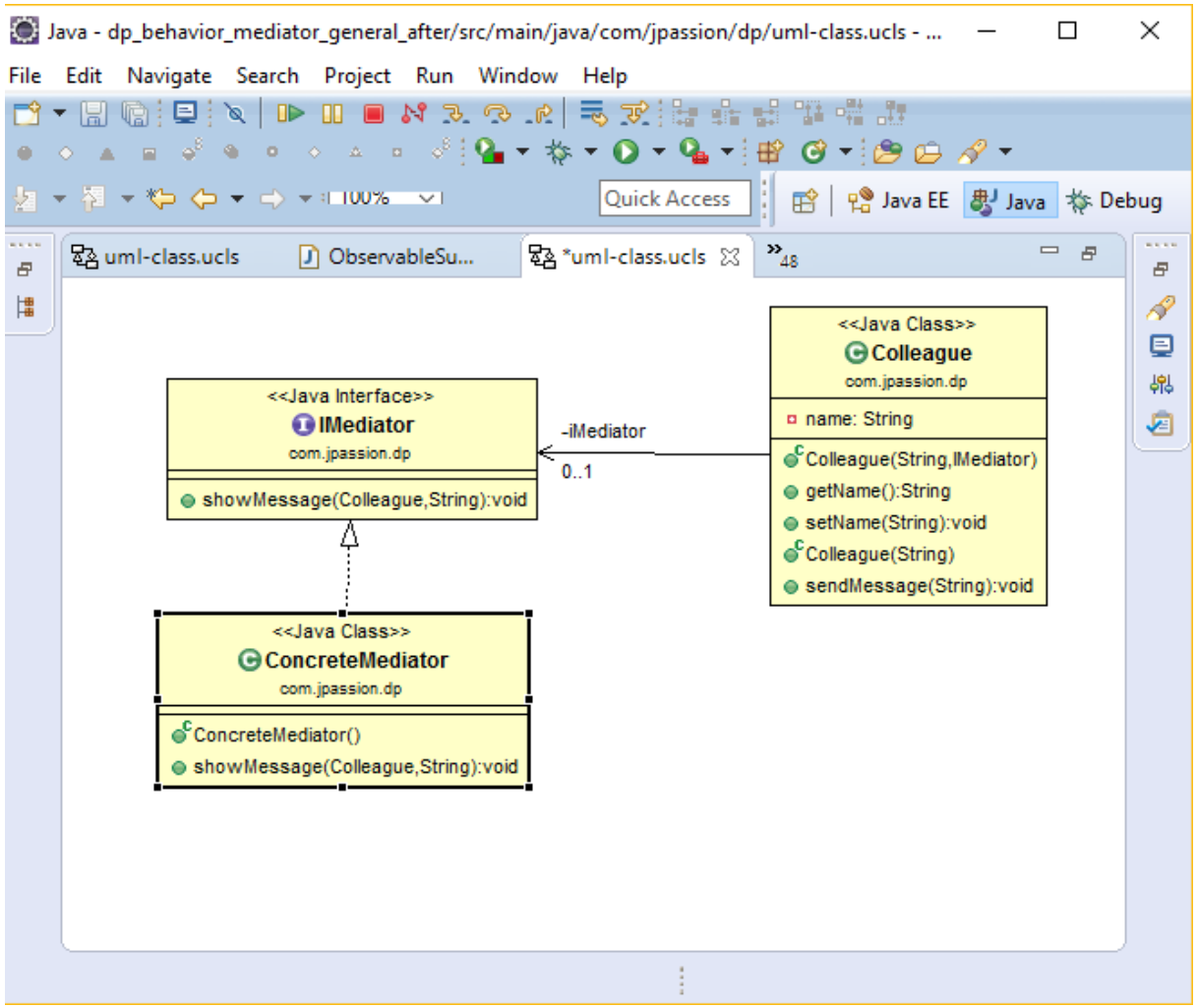
Example Scenario

- Chatting among people can be handled through a ChatRoom as a mediator
  - ChatRoom is responsible for receiving and posting messages between chatters
  - ChatRoom can impose some policies - filter out some vulgar messages
  - ChatRoom can be also responsible for collecting various information – number of messages, etc
- Future changes (without violating Open Close principle)
  - Add more chatter types
  - Add more policies (without forcing changes to the Chatter code)

Participants

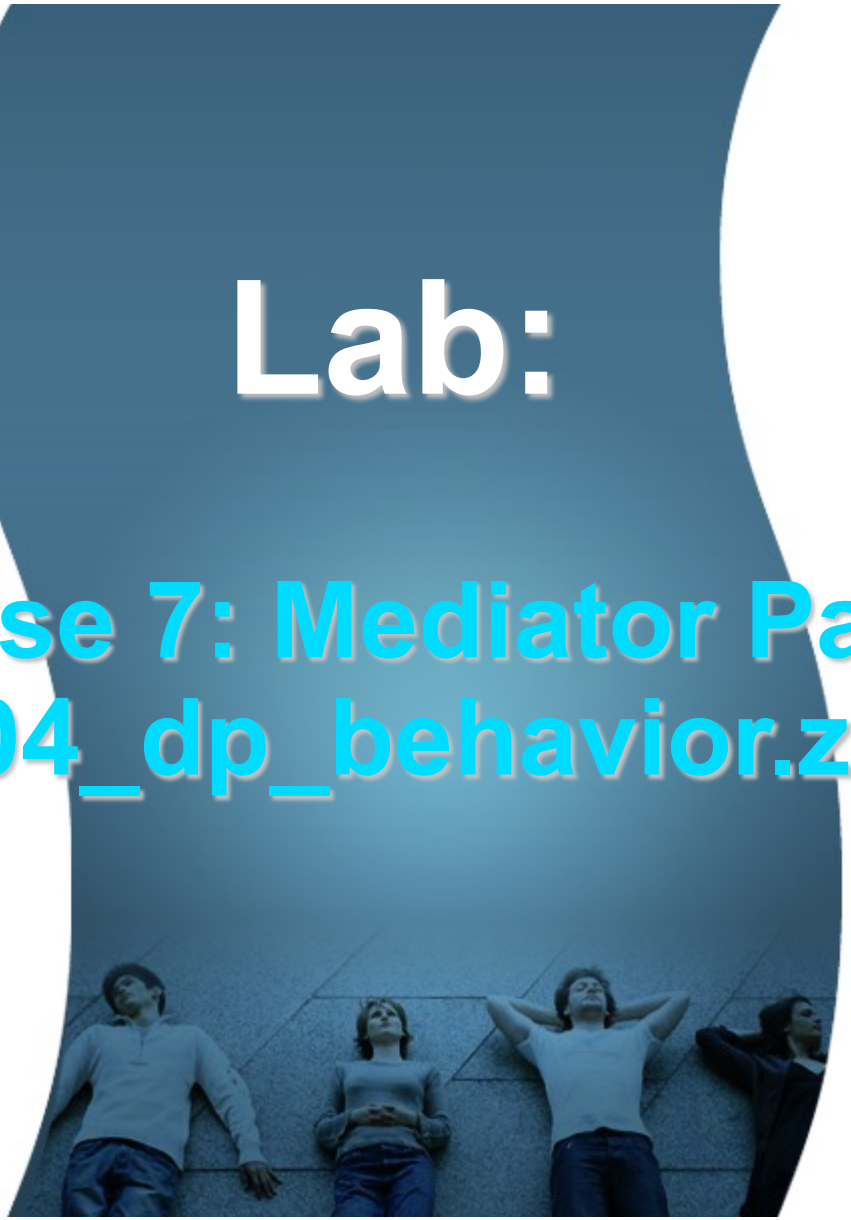
- Mediator
  - Defines the interface for communication between Colleague objects
- ConcreteMediator
  - Implements the Mediator interface and coordinates communication between Colleague objects
  - Is aware of all the Colleagues and their purpose with regards to inter communication.
- Colleague classes
  - Keep a reference to its Mediator object
  - Communicates with other Colleagues through its Mediator

Mediator Pattern



Lab:

Exercise 7: Mediator Pattern
9004_dp_behavior.zip

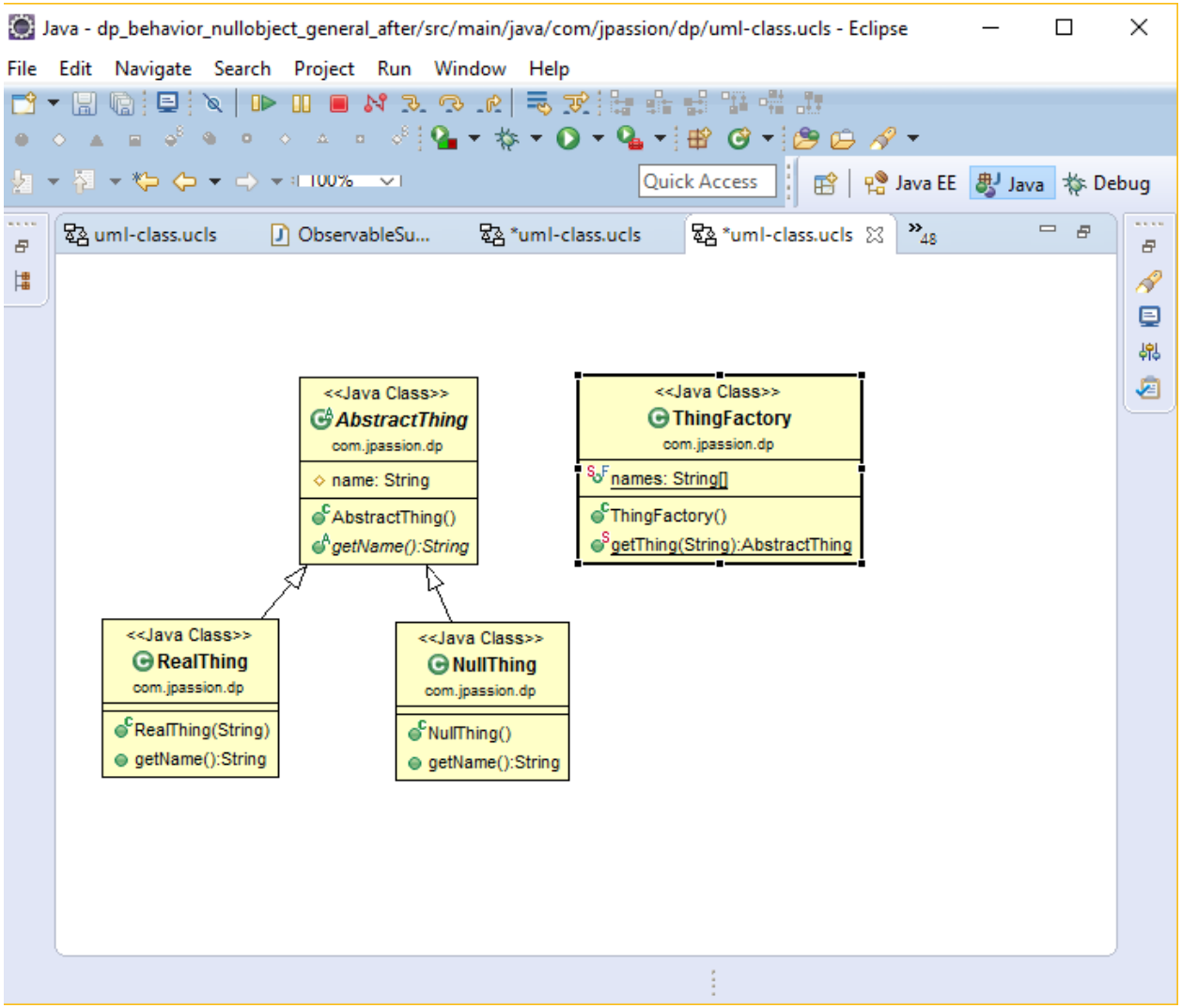


Null object Pattern

Null Object pattern

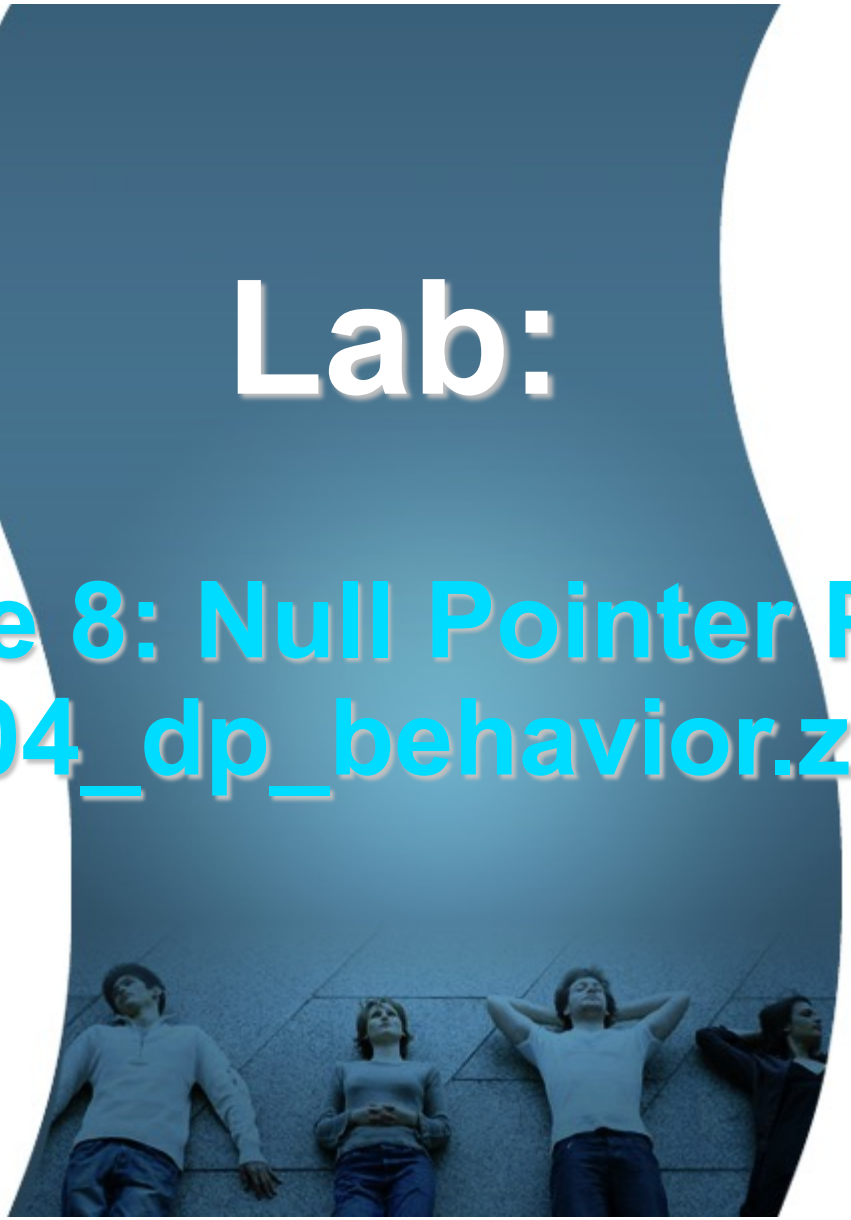
- What is it?
  - Avoid null references by providing a default object
- When to use it?
  - Use it when checking null reference is not desirable

Null Object pattern



Lab:

Exercise 8: Null Pointer Pattern 9004_dp_behavior.zip



Iterator Pattern

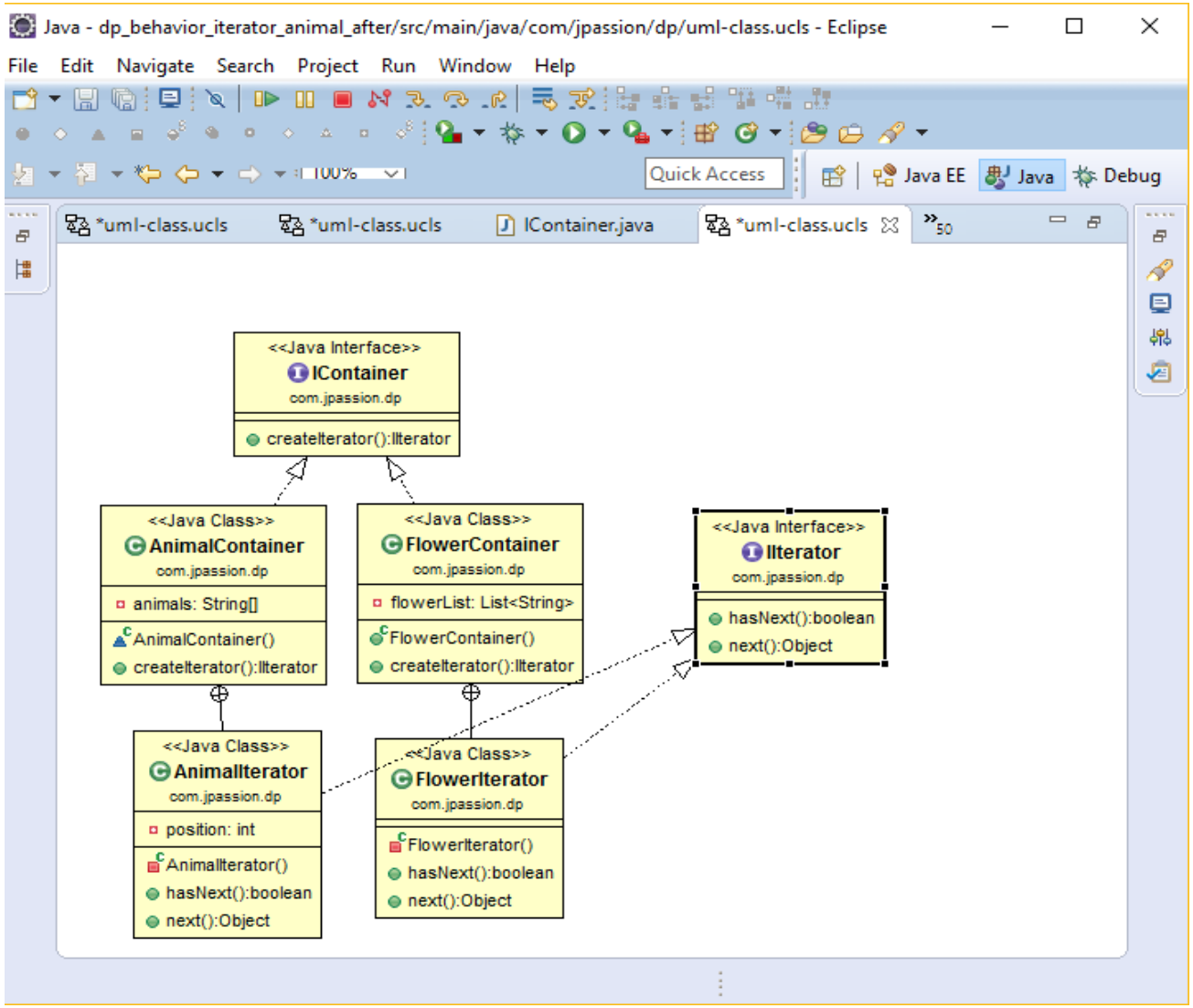
Iterator Pattern

- What is it?
  - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation
- Future changes (without violating Open Close principle)
  - Change the underlying representation of the aggregate object

Participants

- Iterator interface
  - Represent the AbstractIterator, defining the iterator
- ConcreteIterator
  - Implementation of Iterator
- Container interface
  - Represents aggregation
  - Creates Iterator object
- ConcreteContainer
  - Implements Container interface

Iterator Pattern



Lab:

Exercise 9: Iterator Pattern
9004_dp_behavior.zip

Code with Passion!
JPassion.com