University of South Australia

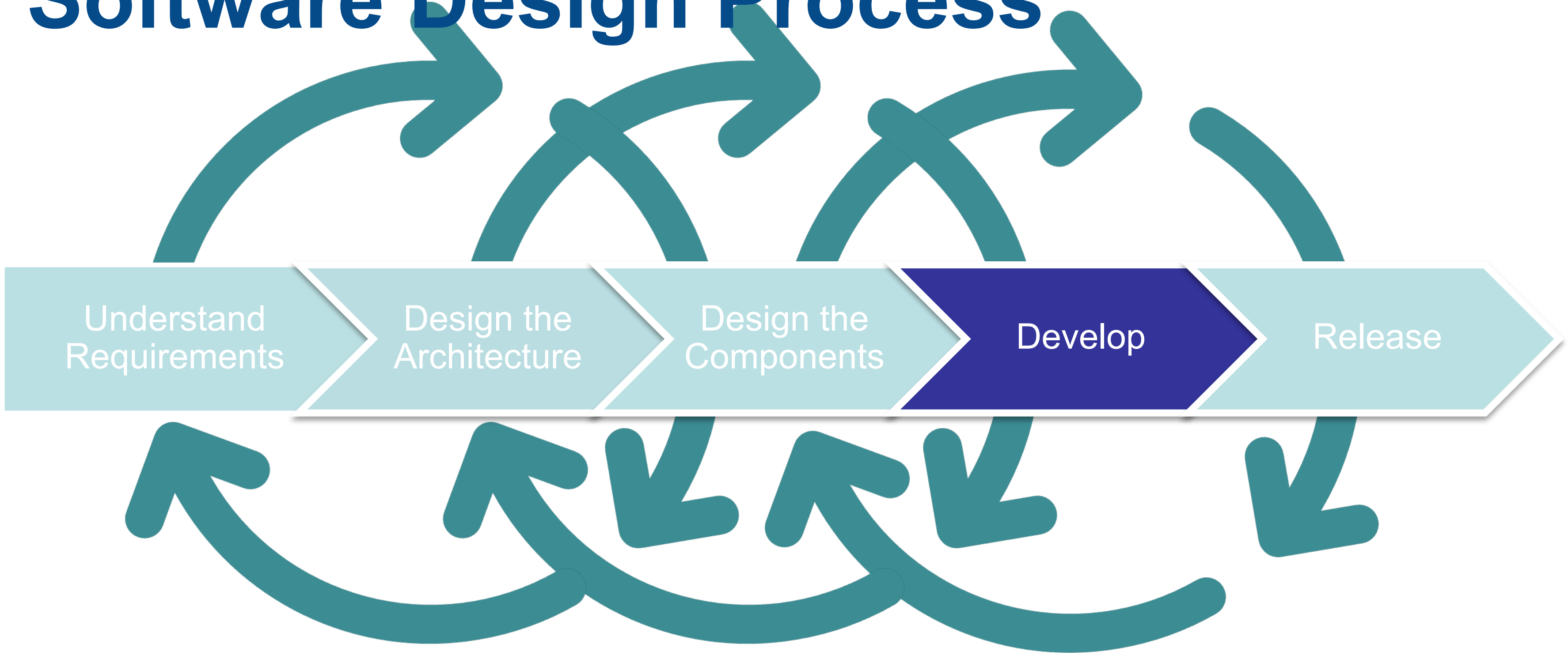
INFS 2044

Workshop 5a Answers

Preparation

- Read the required readings
- Watch the Week 5 Lecture
- Bring a copy of the workshop instructions (this document) to the workshop

Software Design Process



Where We Are At

- Designed components, their interfaces, and their interactions

- Documented implementation design using UML Sequence diagrams and UML Class diagrams

Learning Objectives

- Apply design principles to assess alternate implementation designs
- Apply design patterns in implementation design

Task 1. Apply the Strategy Pattern

- Revisit the price calculation aspect of the *UC01 Make Booking* use case defined in Workshop 3.

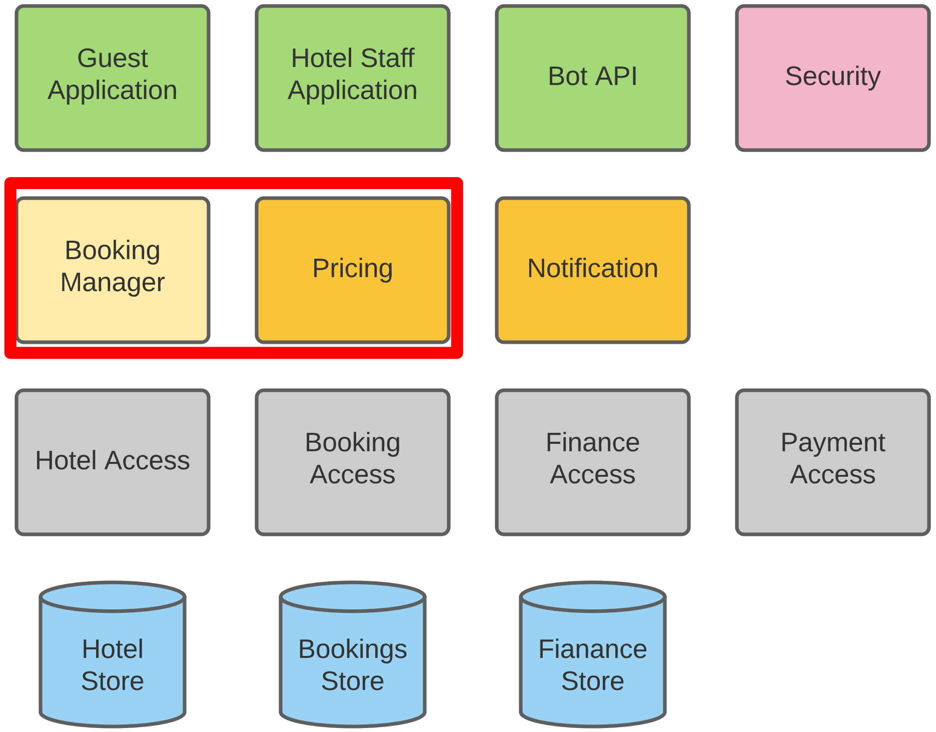
- Our requirements have changed:

  The system shall support *multiple different pricing policies* to support promotion campaigns run at different times of the year.

Pricing Policies

- Policy 1: Discount by x% (already done in Workshop 3)
- Policy 2: Discount by $x
- Policy 3: Discount increases with undiscounted $$ price
- Policy 4: Discount of x% for selected room types
- Policy 5: Discount of x% for VIP guests
- …

Booking System Decomposition



Booking Manager Pricing Design #1

- Assess the design presented on the subsequent slides with respect to design principles.
- Does it satisfy these principles?:
  - High cohesion
  - Low coupling
  - Single Responsibility Principle
  - Open-closed Principle
  - Liskov's Substitution Principle
  - Interface Segregation Principle

Booking Manager Pricing Design #1

```
class BookingManager:
    def createBooking(self, roomID, inDate, outDate, contactDetails):
        basePrice = self.getBasePrice(roomID, inDate, outDate)
        totalPrice = self.getTotalPrice(self, roomID, inDate, outDate)
        #...
    def getTotalPrice(self, roomID, inDate, outDate):
        totalPrice = ...
        if self.__ppolicy = 'PercentDiscount':
            return self.__percentDiscount * totalPrice
        elif self.__ppolicy = 'DollarDiscount':
            return max(0,totalPrice - self.__dollarDiscount)
        elif ...
        return totalPrice
```

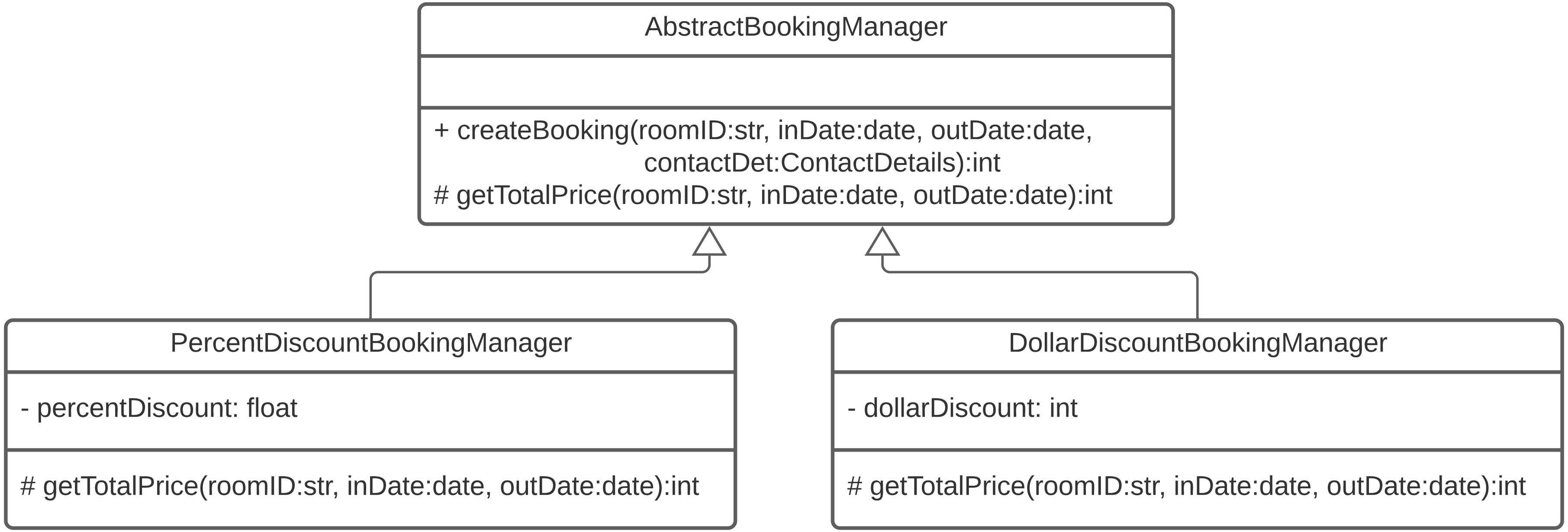
Issues with Booking Manager #1

- Booking Manager has diluted cohesion
  - Orchestration of use cases, and
  - Pricing policy calculations (multiple)
  - The many attributes __percentDiscount, __dollarDiscount, etc that are used in only one place are red flags
- The implementation of the pricing policies is tightly coupled to the implementation of Booking Manager
- The implementation is difficult to extend (violates Open-Closed Principle)
  - Must modify existing code to introduce other pricing policies
- The Booking Manager violates Single Responsibility Principle
  - It has two reasons to change (use case narrative, pricing policy update)

Booking Manager Pricing Design #2

- Assess the design presented on the subsequent slides with respect to design principles.
- Does it satisfy these principles?:
  - High cohesion
  - Low coupling
  - Single Responsibility Principle
  - Open-closed Principle
  - Liskov's Substitution Principle
  - Interface Segregation Principle

Booking Manager Pricing Design #2


AbstractBookingManager
+ createBooking(roomID:str, inDate:date, outDate:date, contactDet:ContactDetails):int
# getTotalPrice(roomID:str, inDate:date, outDate:date):int
PercentDiscountBookingManager
- percentDiscount: float
# getTotalPrice(roomID:str, inDate:date, outDate:date):int
DollarDiscountBookingManager
- dollarDiscount: int
# getTotalPrice(roomID:str, inDate:date, outDate:date):int

Booking Manager Design #2

```
class AbstractBookingManager:
    def createBooking(self, roomID, inDate, outDate, contactDetails):
        basePrice = self.getBasePrice(roomID, inDate, outDate)
        totalPrice = self.getTotalPrice(self, roomID, inDate, outDate)
        #...
        bookingID = self.__bookingAccess.createBooking(roomID,
                                      inDate, outDate, guestID, totalPrice)
        return bookingID
```

Booking Manager Design #2

```
class BookingManagerPercentageDiscount(AbstractBookingManager):
    def __init__(self, percentDiscount, ...):
        self.__percentDiscount = percentDiscount

    def getTotalPrice(self, roomID, inDate, outDate):
        basePrice = self.getBasePrice(roomID, inDate, outDate)
        return basePrice * self.__percentDiscount
```

Booking Manager Design #2

```python
class BookingManagerDollarDiscount(AbstractBookingManager):
    def __init__(self, dollarDiscount, ...):
        self.__dollarDiscount = dollarDiscount

    def getTotalPrice(self, roomID, inDate, outDate):
        basePrice = self.getBasePrice(roomID, inDate, outDate)
        return max(0,basePrice - self.__dollarDiscount)
```

Issues with Design #2

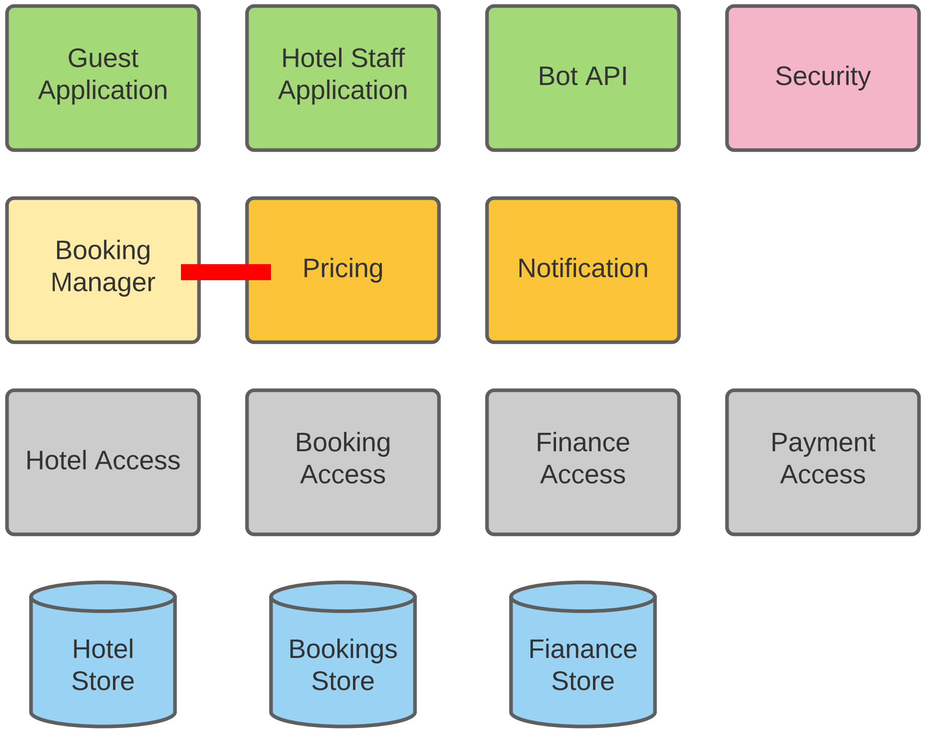
- Booking Manager again has poor Cohesion and violates the Single-Responsibility- and the Interface Segregation Principles
- The implementation of the pricing policies is tightly coupled to the implementation of Booking Manager
- These should be separated as per our component design
- Impossible to change the pricing policy without creating a new BookingManager instance

- The Open-Closed principle is satisfied this time

Booking Manager Pricing Design #3

- Assess the design presented on the subsequent slides with respect to design principles.
- Does it satisfy these principles?:
  - High cohesion
  - Low coupling
  - Single Responsibility Principle
  - Open-closed Principle
  - Liskov’s Substitution Principle
  - Interface Segregation Principle
  - Dependency Inversion Principle

Booking System Decomposition



University of South Australia

Booking Manager #3 (as of WS3)

```python
class BookingManager:
    def __init__(self):
        self.__pricingPolicy = PercentDiscountPricingPolicy(10)

    def createBooking(self, roomID, inDate, outDate, contactDetails):
        basePrice = self.getBasePrice(roomID, inDate, outDate)
        totalPrice = self.__pricingPolicy.getTotalPrice(basePrice)
        #...
```

Issues with Booking Manager #3

- Booking Manager tightly coupled to implementation of a pricing policy (Violates low coupling)

- Requires modifying existing code to support extra pricing policies (Violates Open-Closed Principle)

- Satisfies Single Responsibility Principle (almost)
- Satisfies the Interface Segregation Principle

- Violates Dependency Inversion Principle
  - Because Booking Manager depends on a volatile, concrete implementation, not a stable abstraction

Booking Manager #4 (DIP)

```
class BookingManager:
    def __init__(self, pricingPolicy):
        self.__pricingPolicy = pricingPolicy

    def createBooking(self, roomID, inDate, outDate, contactDetails):
        basePrice = self.getBasePrice(roomID, inDate, outDate)
        totalPrice = self.__pricingPolicy.getTotalPrice(basePrice)
        #...
```

Issues with Booking Manager #4

- Now all principles are satisfied by Booking Manager

Task 2. Design the Pricing Policies?

- Assess the following design with respect to the design principles
  - High Cohesion
  - Low Coupling
  - Encapsulation / Information Hiding
  - Single Responsibility Principle
  - Open-Closed Principle

Pricing Policy Design #1

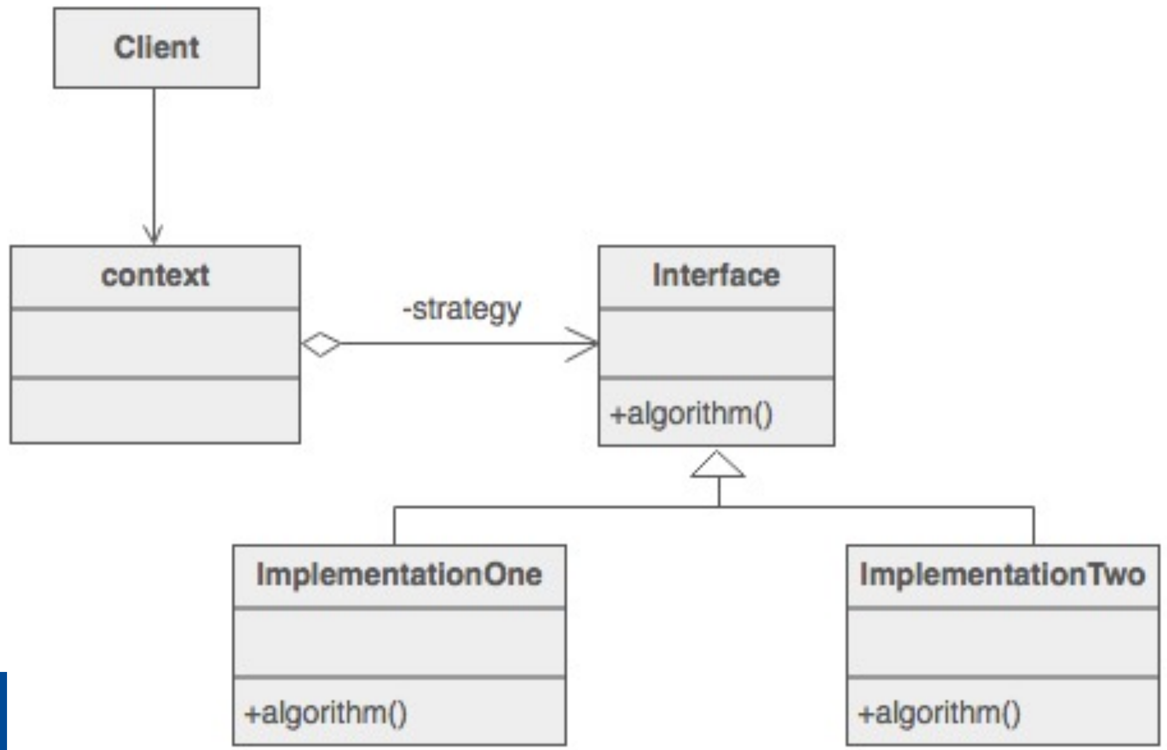
```
class EveryDiscountPricingPolicy:
  def __init__(self, pD, dD, ...):
    self.__percentDiscount = pD
    self.__dollarDiscount = dD

  def getTotalPrice(self, basePrice, discount):
    if discount = 'PercentDiscount':
       discountedPrice = basePrice * self.__percentDiscount
    elif discount = 'DollarDiscount':
       discountedPrice = max(0, basePrice - self.__dollarDiscount)
    elif ...
    return discountedPrice
```

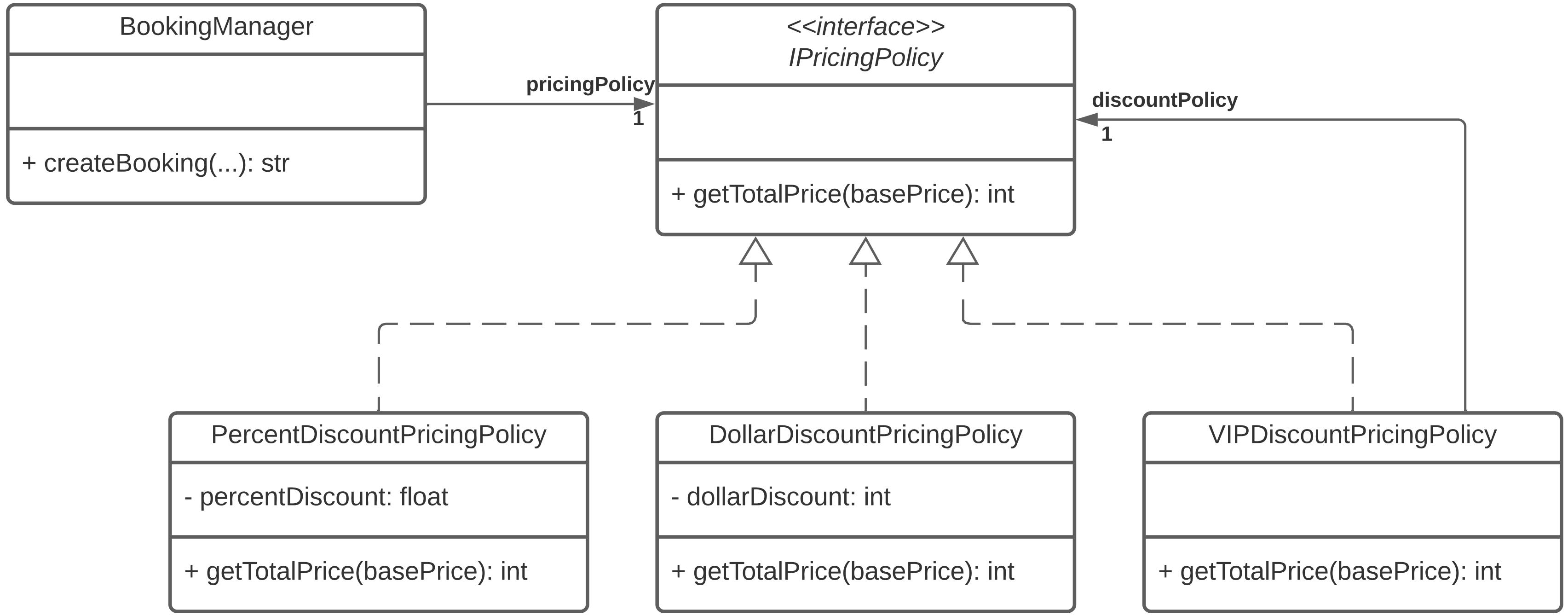
Issues with Pricing Policy Design #1

- Violates the Open-Closed Principle
  - Difficult to extend
- Class has poor Cohesion
  - Many variables used in only one method is a red flag
- Exposes some implementation detail to the client
  - “discount” parameter in getTotalPrice(…)

Strategy Pattern



Strategy Pattern for *PricingPolicy

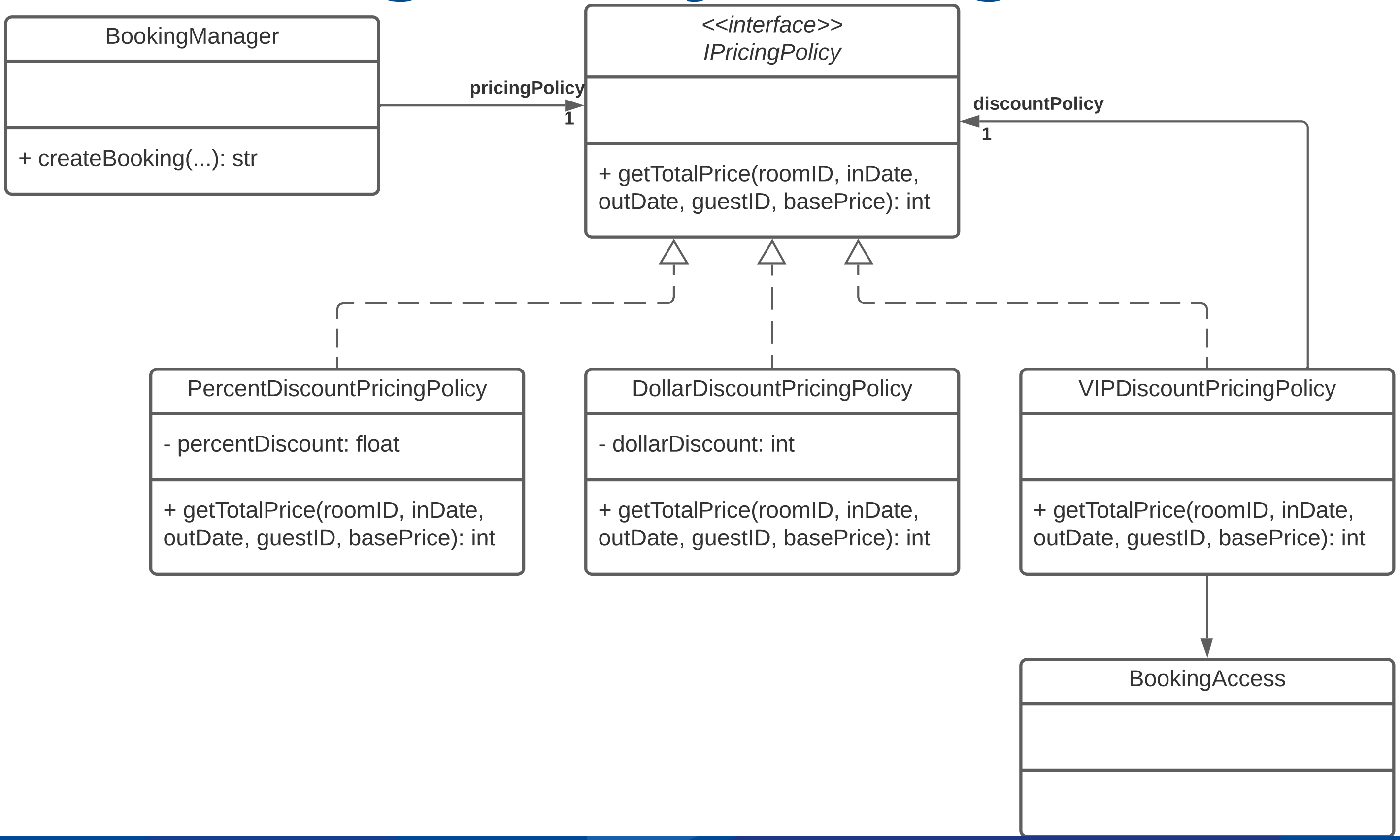


BookingManager
+ createBooking(...): str
pricingPolicy
1
<<interface>>
IPricingPolicy
+ getTotalPrice(basePrice): int
discountPolicy
1
PercentDiscountPricingPolicy
- percentDiscount: float
+ getTotalPrice(basePrice): int
DollarDiscountPricingPolicy
- dollarDiscount: int
+ getTotalPrice(basePrice): int
VIPDiscountPricingPolicy
+ getTotalPrice(basePrice): int

Issues with this Design

- This design does not support some of the desired pricing policies
- Policy 4: Discount of x% for selected room types
  - Requires room type information
  - Current PricingPolicy interface does not receive this information
- Policy 5: Discount of x% for VIP guests
  - Requires guest information
  - Current PricingPolicy interface does not receive this information

Final Pricing Policy Design


BookingManager
+ createBooking(...): str
pricingPolicy
1
<<interface>>
IPricingPolicy
+ getTotalPrice(roomID, inDate, outDate, guestID, basePrice): int
discountPolicy
1
PercentDiscountPricingPolicy
- percentDiscount: float
+ getTotalPrice(roomID, inDate, outDate, guestID, basePrice): int
DollarDiscountPricingPolicy
- dollarDiscount: int
+ getTotalPrice(roomID, inDate, outDate, guestID, basePrice): int
VIPDiscountPricingPolicy
+ getTotalPrice(roomID, inDate, outDate, guestID, basePrice): int
BookingAccess

Final Pricing Policy Design in Python

```
class IPricingPolicy:
    def getTotalPrice(self, roomID, inDate, outDate, guestID, basePrice
        pass

class PercentDiscountPricingPolicy(IPricingPolicy):
  def __init__(self, percentDiscount):
    self.percentDiscount = percentDiscount

  def getTotalPrice(self, roomID, inDate, outDate, guestID, basePrice):
    discountedPrice = basePrice * self.percentDiscount
    return discountedPrice
```

University of South Australia

Final Pricing Policy Design

```python
class DollarDiscountPricingPolicy(IPricingPolicy):
  def __init__(self, dollarDiscount):
    self.dollarDiscount = dollarDiscount

  def getTotalPrice(self, roomID, inDate, outDate, guestID, basePrice):
    discountedPrice = max(0, basePrice - self.dollarDiscount)
    return discountedPrice
```

Final Pricing Policy Design

```
class VIPDiscountPricingPolicy(IPricingPolicy):
  def __init__(self, discountPolicy):
    # we can plug in any discount policy we like to apply for VIPs
    self.discountPolicy = discountPolicy
  def isVIP(self, guestID):
    # fetch guest from bookingAccess and determine if they are a VIP
    return True or False
  def getTotalPrice(self, roomID, inDate, outDate, guestID, basePrice):
    if self.isVIP(guestID):
        discountedPrice = self.discountPolicy.getTotalPrice(roomID, inDate,
                                                  outDate, guestID, basePrice)
    else:
        discountedPrice = basePrice
    return discountedPrice
```

Task 3. Design the Policy Creation

- Suppose the active pricing policy and the discount is determined by a configuration file.
- Assess the design on the next slide with respect to the design principles used in Task 1
- Re-Design the policy creation mechanism using the Abstract Factory Pattern

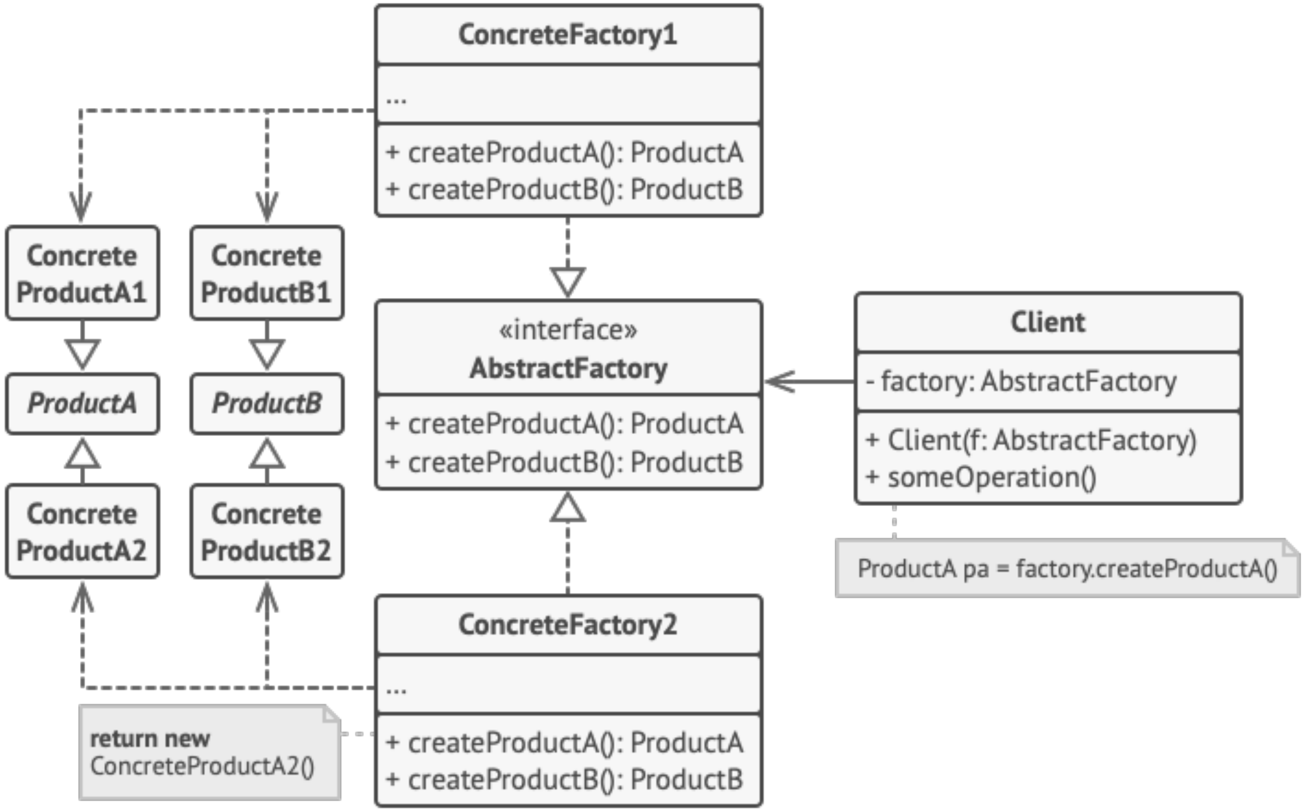
Booking Manager #5

```python
class BookingManager:
    def __init__(self, configFile):
        # read `configFile` and determine the pricing policy
        # ...
        # create an instance of the corresponding *PricingPolicy class
        # and store it in the private attribute
        self.__pricingPolicy = ...
```

Issues with Booking Manager #5

- Booking Manager has knowledge about how the pricing policy is determined, created, and configured
  - Poor cohesion
  - Coupled to configuration file implementation details
  - Violates the Single Responsibility Principle

Abstract Factory Pattern


ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
Concrete ProductA1
Concrete ProductB1
ProductA
ProductB
Concrete ProductA2
Concrete ProductB2
«interface» AbstractFactory
+ createProductA(): ProductA
+ createProductB(): ProductB
Client
- factory: AbstractFactory
+ Client(f: AbstractFactory)
+ someOperation()
ProductA pa = factory.createProductA()
ConcreteFactory2
...
+ createProductA(): ProductA
+ createProductB(): ProductB
return new ConcreteProductA2()

University of South Australia

https://refactoring.guru/design-patterns/abstract-factory

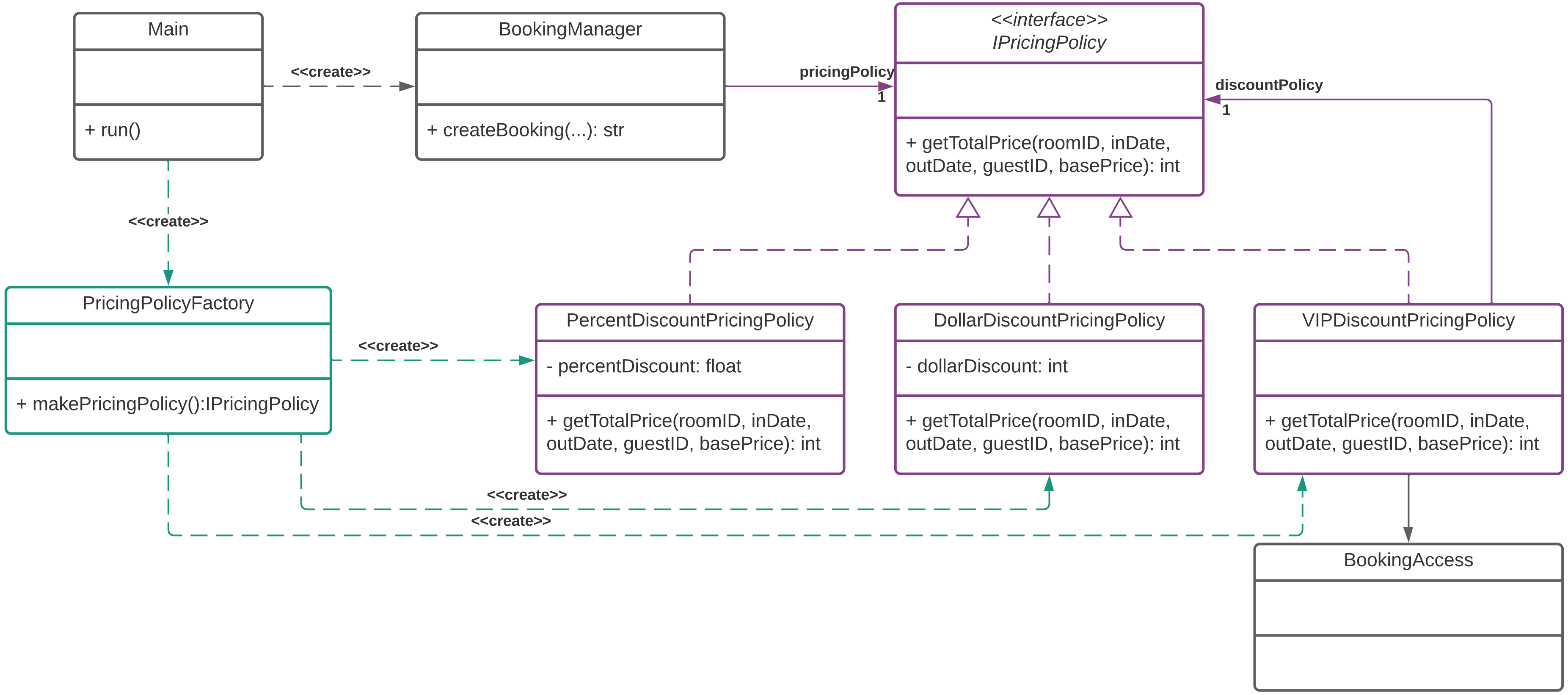
Booking Manager #6

```python
class PricingPolicyFactory:
    def makePricingPolicy(self):
        # read `configFile` and determine the pricing policy
        # ...
        # create and return an instance of the corresponding *PricingPolicy class
        return ...
```

Booking Manager #6

```python
class BookingManager:
    def __init__(self, pricingPolicy):
        self.__pricingPolicy = pricingPolicy

class Main:
    def run(self):
        factory = PricingPolicyFactory()
        policy = factory.makePricingPolicy()
        manager = BookingManager(policy)
        ...
```


Main
+ run()
<<create>>
BookingManager
+ createBooking(...): str
pricingPolicy
1
<<interface>>
IPricingPolicy
+ getTotalPrice(roomID, inDate, outDate, guestID, basePrice): int
discountPolicy
1
<<create>>
PricingPolicyFactory
+ makePricingPolicy():IPricingPolicy
<<create>>
PercentDiscountPricingPolicy
- percentDiscount: float
+ getTotalPrice(roomID, inDate, outDate, guestID, basePrice): int
DollarDiscountPricingPolicy
- dollarDiscount: int
+ getTotalPrice(roomID, inDate, outDate, guestID, basePrice): int
VIPDiscountPricingPolicy
+ getTotalPrice(roomID, inDate, outDate, guestID, basePrice): int
<<create>>
<<create>>
BookingAccess

You Should Know

- Recognise violations of Design Principles
- Assess alternative designs with respect to Design Principles
- Apply Design Patterns to solve common implementation design problems

Activities this Week

- Complete Quiz 5
- Continue working on Assignment 1

University of
South Australia