University of
South Australia

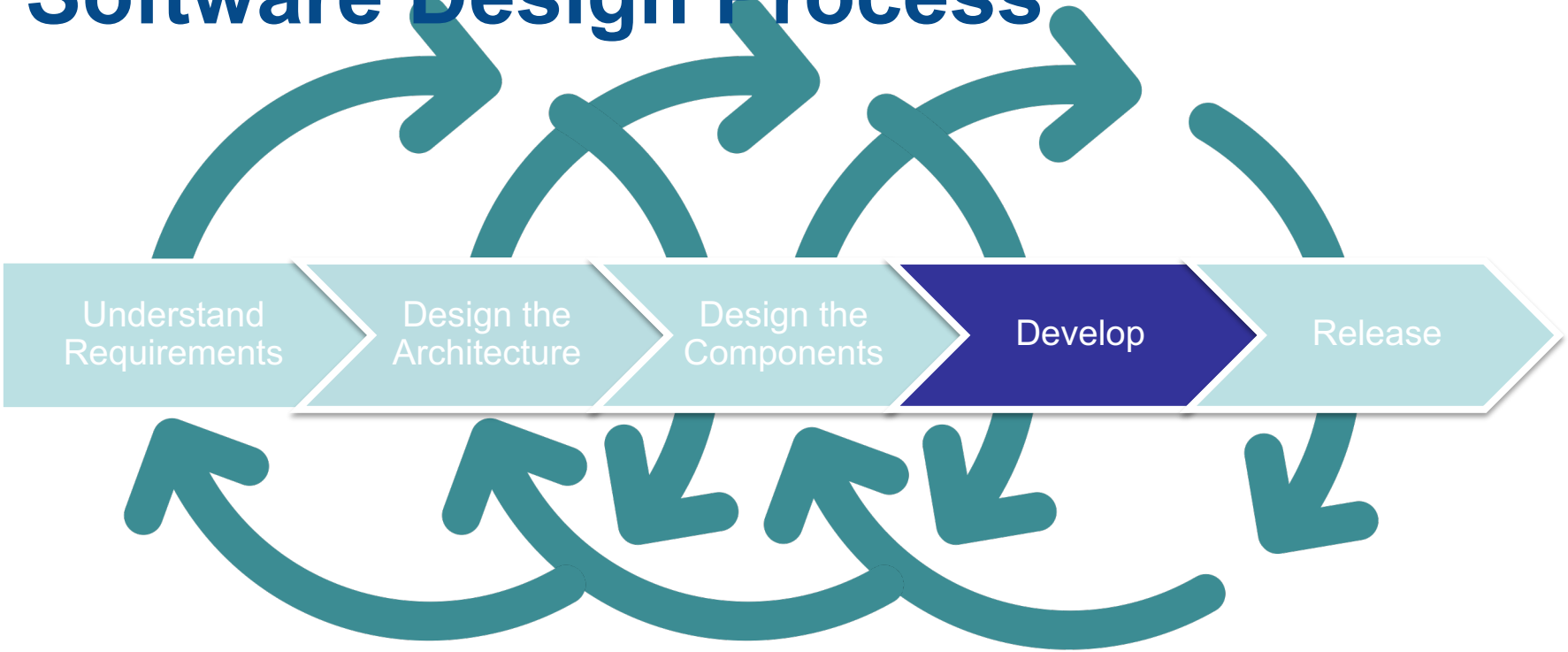
INFS 2044

Workshop 5a

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.

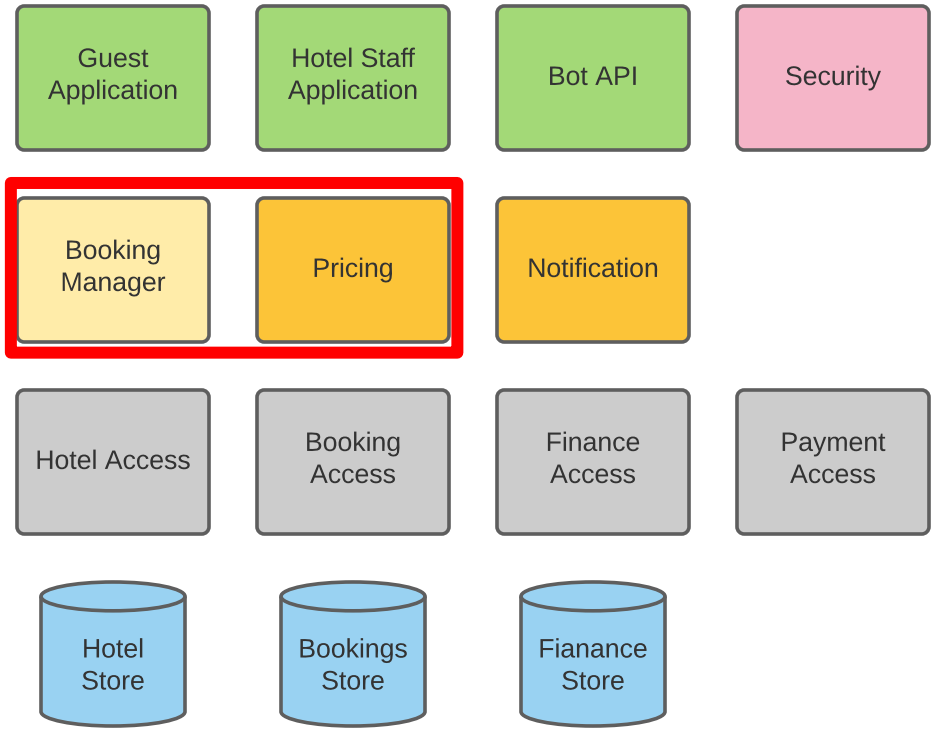
- Our requirements have changed:

  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
  - Open-closed
  - Liskov's Substitution Principle
  - Interface Segregation Principle

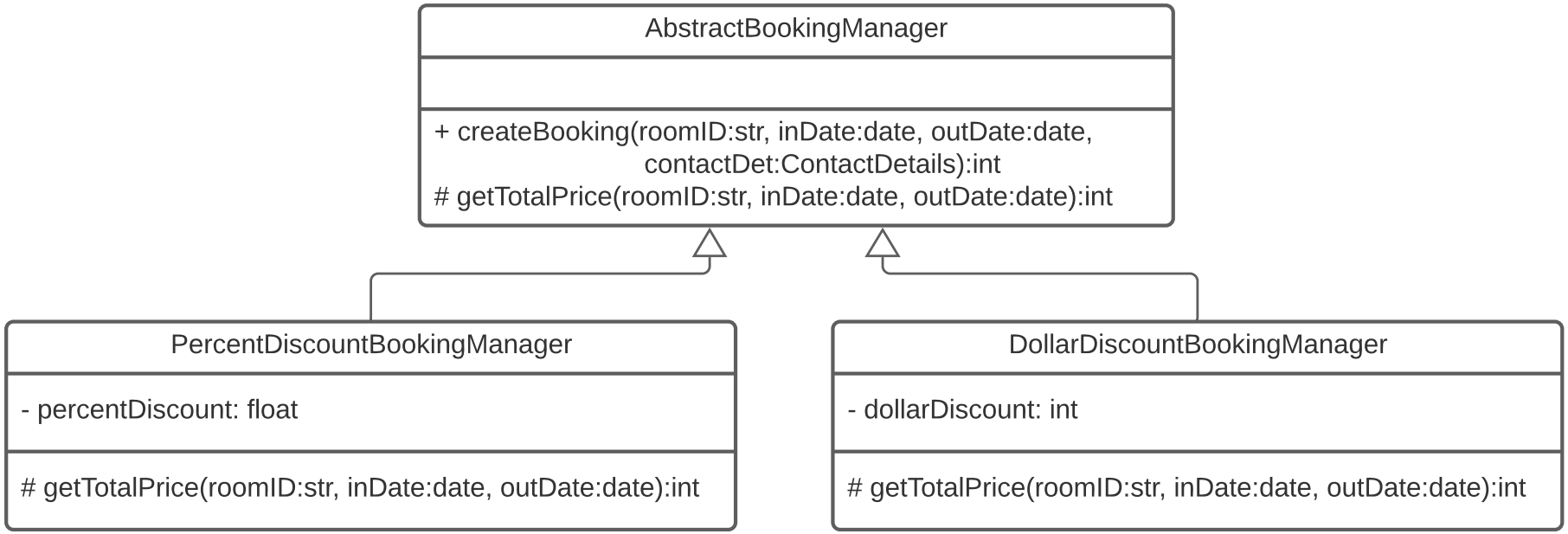
Booking Manager Pricing Design #1

```
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
```

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



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

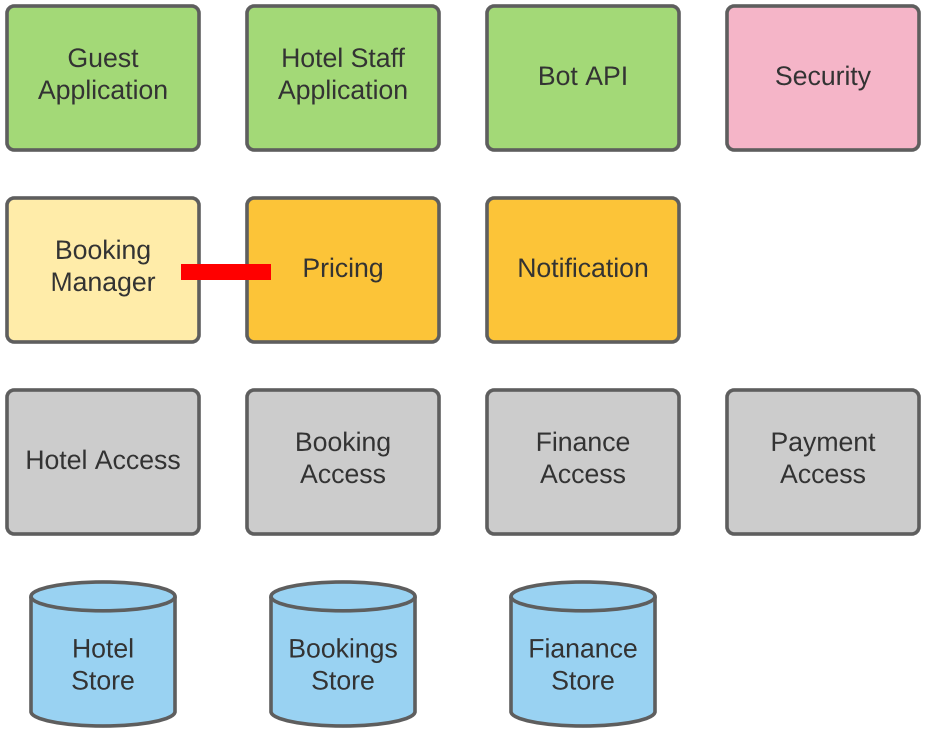
```
class BookingManagerDollarDiscount(AbstractBookingManager):
    def __init__(self, dollarDiscount, ...):
        self.__dollarDiscount = dollarDiscount

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

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



Booking Manager #3 (as of WS3)

```
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(self, basePrice)
        #...
```

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(self, basePrice)
        #...
```

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
```

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

```
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 = ...
```

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