Object-Oriented Analysis and Design Using UML

Activity Guide

OO-226 Rev E

D61808GC21

Edition 2.1

June 2010

D67901

ORACLE®

Disclaimer

This document contains proprietary information, is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except as expressly permitted in your license agreement or allowed by law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Sun Microsystems, Inc. Disclaimer

This training manual may include references to materials, offerings, or products that were previously offered by Sun Microsystems, Inc. Certain materials, offerings, services, or products may no longer be offered or provided. Oracle and its affiliates cannot be held responsible for any such references should they appear in the text provided.

Restricted Rights Notice

If this documentation is delivered to the U.S. Government or anyone using the documentation on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS
The U.S. Government’s rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This page intentionally left blank.

This page intentionally left blank.

Table of Contents

About This Workbook

Conducting the labs

Except for self-check activities or where directed otherwise, it is suggested that the lab activities in this workbook are completed in small groups. This promotes the UML modeling ideals of communicating and sharing of ideas to create better solutions, which can enhance the learning experience. The instructor will assist in assigning groups.

Self-check activities are meant to be completed individually by each student. However, these activities may also be completed in pairs or small groups.

Conventions

The following conventions are used in this course to represent various training elements and alternative learning resources.

Typographical Conventions

`Courier` is used for the names of commands, files, directories, programming code, and on-screen computer output; for example:

> Use `ls -al` to list all files.
> `system% You have mail.`

`Courier` is also used to indicate programming constructs, such as class names, methods, and keywords; for example:

> The `getServletInfo` method is used to get author information.
> The `java.awt.Dialog` class contains `Dialog` constructor.

`Courier bold` is used for characters and numbers that you type; for example:

> To list the files in this directory, type:
> `#` **`ls`**

`Courier bold` is also used for each line of programming code that is referenced in a textual description; for example:

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
```

> Notice the `javax.servlet` interface is imported to allow access to its life cycle methods (Line 2).

`Courier italics` is used for variables and command-line placeholders that are replaced with a real name or value; for example:

> To delete a file, use the `rm` *`filename`* command.

`Courier italic bold` is used to represent variables whose values are to be entered by the student as part of an activity; for example:

> Type **`chmod a+rwx`** ***`filename`*** to grant read, write, and execute rights for *`filename`* to world, group, and users.

Palatino italics is used for book titles, new words or terms, or words that you want to emphasize; for example:

> Read Chapter 6 in the *User's Guide*.
> These are called *class* options.

Additional Conventions

Java™ programming language examples use the following additional conventions:

- Method names are not followed with parentheses unless a formal or actual parameter list is shown; for example:

  "The `doIt` method..." refers to any method called `doIt`.

  "The `doIt()` method..." refers to a method called `doIt` that takes no arguments.

- Line breaks occur only where there are separations (commas), conjunctions (operators), or white space in the code. Broken code is indented four spaces under the starting code.

- If a command used in the Solaris™ Operating Environment is different from a command used in the Microsoft Windows platform, both commands are shown; for example:

  If working in the Solaris Operating Environment

  ```
  $cd $SERVER_ROOT/bin
  ```

  If working in Microsoft Windows

  ```
  C:>cd %SERVER_ROOT%\bin
  ```

Lab 1

Examining Object-Oriented Concepts and Terminology

Objectives

Upon completion of these activities, you should be able to:

- Define fundamental object-oriented (OO) concepts
- Create simple class diagrams using classes, associations, inheritance, and interfaces
- Apply the concepts of cohesion and delegation to class diagrams

Activity 1: Using Abstraction

In this exercise, you will use abstraction on a real-world car object. You must ensure that you keep only those responsibilities of the car object that are relevant for each specified domain.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following step:

1. Draw a UML class description for each of the following business domain contexts:
   a. A Car in a "Used Car Lot" business context
   b. A Car in a "Motor Racing Circuit Game" business context

Note – The "Motor Circuit Racing Game" can be a computer simulation or a physical game (for example, Digital Scaletrix) with methods to control the car.

   c. A Car in a "Vehicle Licensing Registration" business context
   d. A Car in any other business context agreeable to your group (optional)

Note – You can refer to the example UML diagrams in Module 1 of the Student Guide to review the graphical notation of a UML Class diagram.

Activity 2: Using Inheritance

In this exercise, you will use inheritance to extend the responsibilities (attributes and methods) of a class.

Preparation

No special preparation is required for this activity.

Task 1 - Create an Inheritance Hierarchy

Complete the following step:

1. Draw UML classes that are related by inheritance. Each class should include its name, attributes, and methods. These classes, which are all from the banking domain, are as follows:
   - An `Account` class that contains the following attributes and methods:
     - Attributes: `balance` and `accountNumber`
     - Methods: `credit` and `debit`
   - A `SavingsAccount` class, whose business rules include that your balance cannot go below zero.
   - A `LoanAccount` class, whose business rules include that your balance cannot go above zero and you can debit only once (that is, when the account is opened).
   - A `CheckingAccount(CurrentAccount)` class, whose business rules include the following:
     - You may have an overdraft limit.
     - If you have an overdraft limit, your balance can drop below zero, but it cannot exceed that overdraft limit.
   - A `TaxFreeSavingsAccount` class, whose business rules include that you have an annual credit limit and you cannot credit more than that amount each year.

Note – You can refer to the example UML diagrams in Module 1 of the Student Guide to review the graphical notation for inheritance.

Note – The problem has a reduced scope to eliminate distractions that add no value at this time. For example, we are not adding any aspects on interest charged or interest payable.

Task 2 - Map the Business Rules to the Methods

Complete the following step:

1. For each method identified in the previous task, create a list containing the following information:
   a. Class name and method name
   b. The business rules that apply to that method

Activity 3: Using Delegation and Cohesion

In this exercise, you will delegate coherent responsibilities to classes by using class associations and inheritance.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Add the following details to the Class diagram that you created in the previous activity:
   - Customer information (such as name, address, and telephone number)
     - A business customer must also have a company registration number.
     - A personal customer might have a tax reference number.
   - A branch code identifier (each branch in this bank has a unique identifier)
   - Bank name and address details
2. Ensure that your Class diagram satisfies the following condition:

   A customer can have many accounts in this bank. However, an account can be owned by only one customer.

Activity 4: Understanding the Benefits of Using Interfaces

In this exercise, you will understand how interfaces enable you to build flexible solutions.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the following scenario:

   You were asked to write a software system that can be used to buy and sell property. You decided to delegate the buy and sell behavior that is specific to properties to a `PropertyTradingService` class that includes the `buy` and `sell` methods.

   Now, you have been asked to ensure that this software system is able to support the trading of different types of items, such as vehicles and stock market shares.

2. Draw a class diagram that describes the classes and interfaces that will be required to provide a flexible solution to the problem specified in step 1. You should ensure that new types of items can be added to the software system with minimal changes to the existing software system.

Activity 5: Defining Object-Oriented Terminology

Self-Check – Match the object-oriented programming terms with their definitions.

Term	Definition
polymorphism	Identifying only those attributes and methods that are relevant to the class within the system.
object link	The ability to derive new classes that acquire attributes and methods from a base class.
inheritance	The blueprint for an object.
abstraction	Hiding internal methods and data from direct access from outside the class.
object	A relationship between two objects (instances).
delegation	The measure of how much an entity (component or class) supports a singular purpose within a system.
encapsulation	The ability to determine the actual method called based on the object's subtype.
association	An instance of a class.
class	Offloading some coherent responsibilities to another component (one or more classes) or method.
coupling	A relationship between two classes.
cohesion	The degree to which classes within our system are dependent on each other.

Lab 2

Introducing Modeling and the Software Development Process

Objectives

Upon completion of these activities, you should be able to:

- Define the workflows in the Object-Oriented Software Development (OOSD) process
- Identify characteristics of the Requirements Gathering workflow
- Identify characteristics of the Requirements Analysis workflow
- Identify characteristics of the Architecture workflow
- Identify characteristics of the Design workflow
- Identify benefits of modeling software
- Define the various Unified Modeling Language (UML) diagram types

Activity 1: Defining Workflows

Self-Check – Match each workflow with its description.

Term	Definition
Requirements Gathering	Model the high-level structure of the system, paying particular attention to the NFRs and mitigation of risk.
Requirements Analysis	Install the implementation into the production environment.
Architecture	Build/Code the software components defined in the Solution model.
Design	Determine the requirements of the system by meeting the business owner and users of the proposed system.
Implementation	Ensure the implementation meets the expectations defined in the requirements.
Testing	Create a Solution model of the system that satisfies the functional requirements.
Deployment	Analyze, refine, and model the requirements of the system.

Activity 2: Identifying Characteristics of the Requirements Gathering Workflow

Self-Check – Select the characteristics of the Requirements Gathering workflow mentioned during the lecture.

a. ___ This workflow includes meeting the business owner and users of the proposed system to understand their requirements.

b. ___ This workflow requires you to model the high-level system structure to satisfy the non-functional requirements (NFRs).

c. ___ The purpose of this workflow is to determine what the system must do.

d. ___ You will create a business domain class diagram showing the required business classes during this workflow.

e. ___ You will create initial Use Case diagrams during this workflow.

Activity 3: Identifying Characteristics of the Requirements Analysis Workflow

Self-Check – Select the characteristics of the Requirements Analysis workflow mentioned during the lecture.

a. ___ This workflow includes recording Use Case scenarios.

b. ___ This workflow starts with analyzing and Use Case scenarios.

c. ___ The purpose of this workflow is to model how the system will support the use cases.

d. ___ You will create a business domain class diagram showing the required business classes during this workflow.

e. ___ You will create a detailed Deployment diagram showing the system architecture during this workflow.

Activity 4: Identifying Characteristics of the Architecture Workflow

Self-Check – Select the characteristics of the Architecture workflow mentioned during the lecture.

a. ___ You will create detailed Deployment diagram during this workflow.

b. ___ The purpose of this workflow is to model the high-level structure of the system to satisfy the NFRs.

c. ___ You will create a tiers and layers diagram during this workflow.

d. ___ The purpose of this workflow is to model the high-level structure of the system to satisfy the FRs.

e. ___ You will refine the Design model during this workflow.

Activity 5: Identifying Characteristics of the Design Workflow

Self-Check – Select the characteristics of the Design workflow mentioned during the lecture.

a. ___ You will use an Activity diagram to verify Use Case diagrams during this workflow.

b. ___ You will analyze the Use Case scenarios to determine additional detail during this workflow.

c. ___ You will create a Solution model during this workflow.

d. ___ You might create a Statechart diagram during this workflow.

e. ___ The purpose of this workflow is to model how the system will support the use cases.

Activity 6: Exploring the Benefits of Modeling

In this exercise, you will explore one of the benefits of modeling.

Preparation

No special preparation is required for this activity.

Tasks

As a group, complete the following steps:

1. Review the following front and plan views of an object that a company wants you to build:

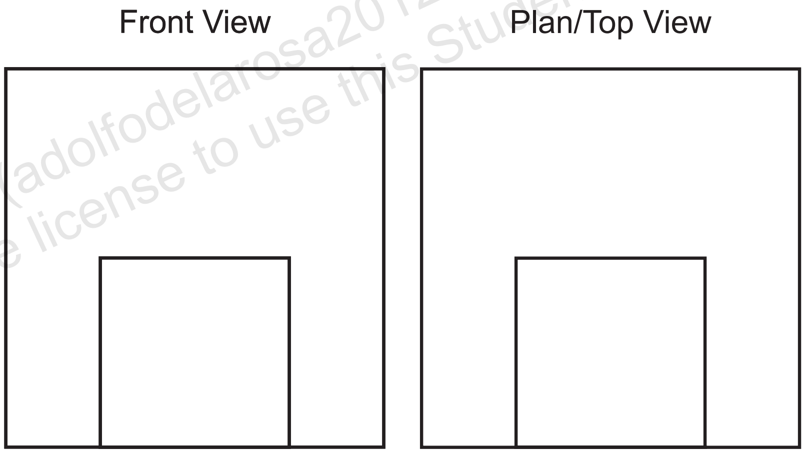


Figure 0-1 Front View and Plan View of an Object

2. Answer the following questions:
   - Has the company given you sufficient information to build the object? If not, what additional information do you need?
   - Assuming that someone did actually build the object for the company, what do you think would the object look like?

Activity 7: Identifying the Benefits of Modeling Software

Self-Check – Select the benefits of modeling software mentioned during the lecture.

a. ___ Models give you only a starting point for a new system.

b. ___ Models help you only to understand what you have developed.

c. ___ Models help you visualize new or existing systems.

d. ___ Models are a concrete realization of a system.

e. ___ Models help you communicate decisions to project stakeholders.

Activity 8: Identifying Diagram Types

Self-Check – Match the UML diagram with its description of the diagram type.

Diagram Name	Definition
Use Case	Represents changes in state or value along with state duration constraints
Class	Represents a flow of tasks that might be performed by either a system or an actor
Object	Represents a collection of components, and shows how these are distributed across one or more hardware nodes
Communication	Represents a flow of task with fragments of detailed object interactions
Sequence	Represents a collection of objects that work together to support some system behavior
Activity	Represents the internal structure of a class in terms of parts
State Machine	Represents a conceptual view of a collection of other modeling elements and diagrams
Component	Represents a collection of physical software components and their interrelationships
Deployment	Represents the set of states that an object might experience along with triggers that transition the object from one state to another
Package	Represents a time-oriented perspective of an object communication
Interaction Overview	Represents a runtime snapshot of software objects and their interrelationships
Timing	Represents extensions to standard diagrams
Composite Structure	Represents the set of high-level behaviors that the system must perform for a given actor

Diagram Name	Definition
Profile	Represents a collection of software classes and their interrelationships

Lab 3

Creating Use Case Diagrams

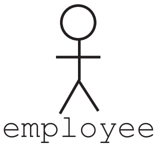
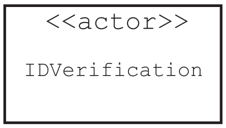
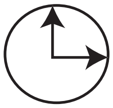
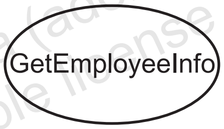
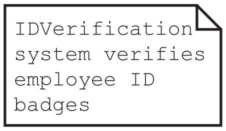
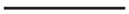
Objectives

Upon completion of these activities, you should be able to:

- Identify the essential elements in a Use Case diagram
- Extend a Use Case diagram based on a list of additional high-level FRs provided by the business owner
- Divide a Use Case diagram into views
- Extend the Use Case diagram views based on a list of additional high-level FRs provided by the additional stakeholders
- Refine the Use Case diagram views by modeling dependency relationships

Activity 1: Identifying Use Case Symbols

Self-Check – Write the name of each Use Case diagram symbol in the space allotted next to each symbol.

Symbol	Symbol Name
	
	
	
	
	
	

Hotel System – Abstract of Additional Requirements (1)

This workbook contains several exercises you can use to apply what you have learnt. Most of the exercises involve a fictional case study requiring you to extend the example Hotel system.

You will use the following abstract of additional high-level requirements discovered during a meeting with the business owner for "Activity 2: Creating a Use Case Diagram" on page L3-4 of this workbook.

Abstract

The receptionist must be able to check out customers before they leave the hotel. During the check-out procedure, a customer's bill is calculated and presented to the customer. In most cases, immediate payment is required to settle the bill for any outstanding room, meal, or other chargeable items added to the bill during the stay. The most common payment method is in the form of a credit card or a debit card.

Some customers will have all or part of their bill guaranteed by a purchase order. Invoices for these charges will be generated daily and either printed or sent electronically to the company or travel agent.

Customers who are currently checked-in, but have not yet checked-out, can charge food, beverages, or other services to their room account.

The room telephone is enabled upon check-in and disabled upon check-out.

Activity 2: Creating a Use Case Diagram

In this exercise, you extend the initial Use Case diagram of the Hotel System based on additional high-level requirements discovered during a meeting with the business owner.

Figure 3-1 illustrates the initial Use Case diagram of the Hotel System.

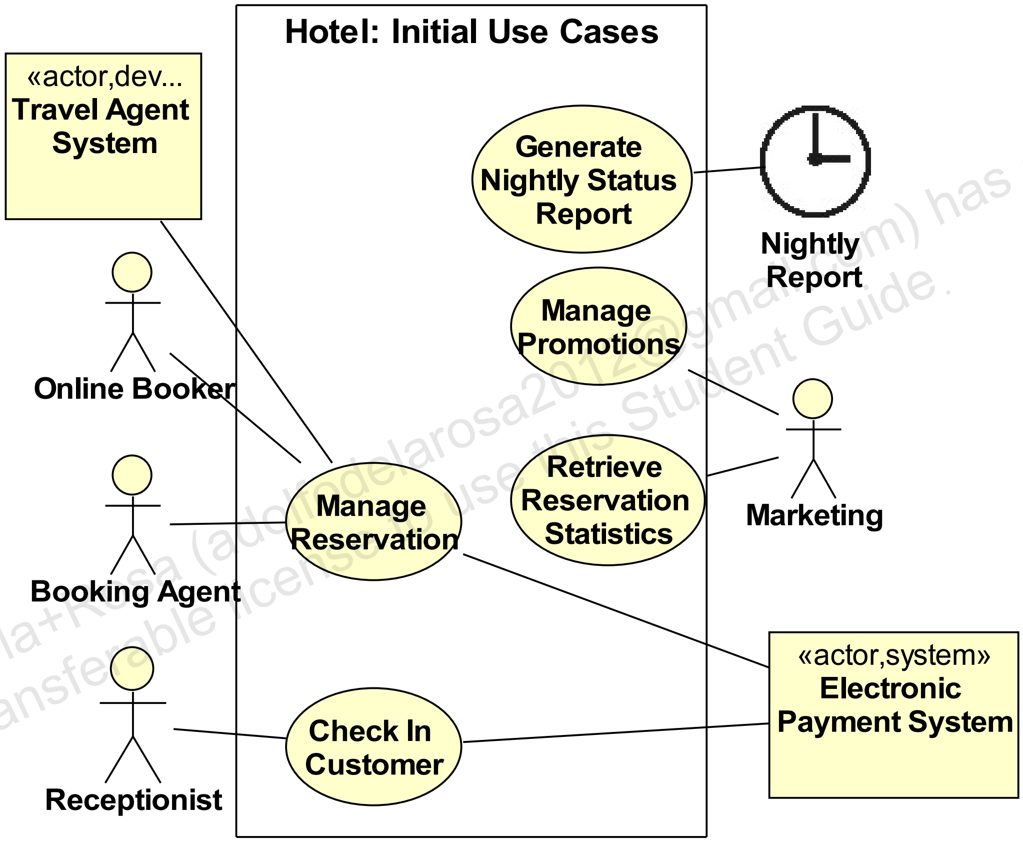


Figure 3-1 Initial Use Case Diagram of the Hotel System

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the abstract of additional requirements for the Hotel System that is provided on page L3-3.
2. Create a system boundary for the Hotel System.

   Hint: Use Case diagrams can grow to be very large. Therefore, it is good practice to only draw the upper-left corner of the system boundary when starting your diagram.
3. From Figure 3-1 on page L3-4, copy any of the actors, use cases, and associations that are relevant to the additional requirements.
4. Add actors that are required by the additional requirements.
5. Add use cases that are required by the additional requirements.
6. Make associations between actors and use cases that are required by the additional requirements.

Hotel System – Abstract of Additional Requirements (2)

You will use the following abstract of additional high-level requirements discovered during a meeting with the additional stakeholders for "Activity 3: Refining the Use Case Diagram" on page L3-7 of this workbook.

Abstract

The marketing staff can publish current offers to their customer base via e-mail.

The hotel group does not currently have a loyalty scheme. However, the group is affiliated with airlines, car rental companies, and other reward points schemes. For example, customers traveling on affiliated airlines will earn airmiles. Customers that have registered a loyalty card will have points awarded to that external scheme when their bill for a reservation is settled.

Customers who provided a phone number must be sent a text message during the morning of their due arrival date. This is a courtesy text message reminding them of their booking.

Cleaners can request a list of rooms that have been vacated. Once a vacated room has been prepared for the next customer, the cleaning staff must mark the room as ready for use.

Occasionally, there are discrepancies in a customer's bill. Therefore, the receptionist may need to make adjustments to the bill.

Night auditors will request that the system generates a discrepancy report during the night. Based on this report, the night auditors might need to make adjustments to a customer's bill. The duty of night auditors is complete for that night only when there are no more discrepancies.

Activity 3: Refining the Use Case Diagram

In this exercise, you complete the following tasks:

- Add new use cases, actors, and associations
- Expand high-level use cases
- Refine the Use Case diagram with dependencies and inheritance where applicable

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the abstract of additional requirements for the Hotel System that is provided on the preceding page.
2. Create a package structure to hold different use case views.
3. Copy the use cases, actors, and associations created in "Activity 2: Creating a Use Case Diagram" to the relevant package views.

Note – A use case or actor may be shown in several Use Case diagram views.

4. Refine the Use Case diagram views by adding actors, use cases, and associations for the newly discovered requirements.
5. Refine the Use Case diagram views by expanding any high-level use cases.
6. Refine the Use Case diagram views by showing dependencies between use cases.

Note – These dependencies can be `<<include>>`, `<<extend>>`, or `generalization` dependencies.

7. You may, if time allows, add other use cases or use case dependencies not mentioned in the additional requirements, that would enhance your understanding.

Lab 4

Creating Use Case Scenarios and Forms

Objectives

Upon completion of these activities, you should be able to:

- Create brief descriptions of the main scenarios for a use case
- Fill in portions of a Use Case form to document a use case and its scenarios
- Add some business terms found in the scenarios to the project's Glossary of Terms

Activity 1: Examining Use Case Scenarios

In this exercise, you will write summaries of the Use Case scenarios for a use case.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the following key facts of the Check In use case that were determined during a meeting with some of the stakeholders:
   - A reservation can be identified by customer name—in which case, the reservation may match more than one customer—or by reservation number. In both situations, it is possible that the required reservation is not found. For example, the reservation will not be found if customer name is not spelled correctly or if the reservation number provided is not correct.
   - The customer may be given the option of upgrading any of the reserved rooms, in which case the additional cost will need to be calculated.
   - The customer may ask to change any of the booked rooms for an alternate room of the same type. For example, the customer asks the Receptionist if the room has a view and is told that the room does not have a view. In this case, the customer might ask the Receptionist to change the room.
   - Payment pre-authorization may be obtained unless already guaranteed for the additional services—such as, meals, beverages, and movies—that may be purchased. For example, the customer's credit card may be pre-authorized with the Electronic Payment Services.
   - If there is no payment guarantee for the additional services, customers are not allowed to charge any items to their bills.
   - If check-in is successful, an electronic room key is generated for each reserved room.

- If check-in is successful, the phone in each reserved room is enabled for making outgoing call, provided that the customer has a payment guarantee for the additional services.
- A Bill—also known as a Folio—is created for the reservation.
- Check-in should take no longer than 8 minutes to complete.
- Upon completion of the Check In use case, the rooms allocated are marked as being occupied.

Note – This list of key facts is not an exhaustive list. Your group may wish to add to this list during the lab.

2. Create a brief description of a primary (successful) scenario.
3. Create a brief description of any other primary (successful) scenarios and any secondary (unsuccessful) scenarios that could occur.
4. Revisit the Use Case diagram that you refined in “Activity 3: Refining the Use Case Diagram” on page L3-7 of this workbook, and determine if the diagram is still correct. If not, update the diagram.
5. Optionally, if time allows, repeat steps 2 through 4 for another use case (for example, Charge Customers Bill or Check Out Customer).

Activity 2: Creating Use Case Forms

In this exercise, you will document a use case by using a Use Case form.

Preparation

Your instructor should hand you a Use Case form template to document the use case. Alternatively, you can use a flip chart or a whiteboard to write the details of the Use Case form.

Tasks

Complete the following steps:

1. Complete all the elements of the Use Case form from the information included in the Use Case scenario for the Check In use case, which you completed in the previous activity, and the key facts provided in the previous activity.
2. Optionally, if time allows, repeat step 1 for another use case (for example, Charge Customers Bill or Check Out Customer).

Activity 3: Writing Glossary of Terms

In this exercise, you will add some new business terms to the project's Glossary of Terms.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Identify a few business terms used in the scenarios of "Activity 1: Examining Use Case Scenarios" or in the Use Case form of "Activity 2: Creating Use Case Forms" that would need to be defined in the project's Glossary of Terms.
2. Write a definition for each of the terms identified in step 1.

Lab 5

Creating Activity Diagrams

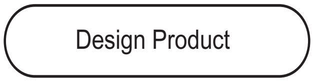
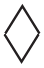
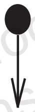
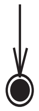
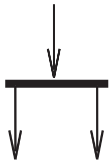
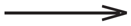
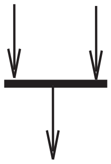
Objectives

Upon completion of these activities, you should be able to:

- Identify the essential elements in an Activity diagram
- Create an Activity diagram to model a Use Case diagram

Actrivity1: Identifying Activity Diagram Symbols

Self-Check – Write the name of each Activity diagram symbol in the space allotted next to each symbol, in the following table.

Symbol	Symbol Name
	
	
	
	
	
	
	

Activity 2: Creating an Activity Diagram

In this exercise, you construct an Activity diagram for a use case in the Hotel System case study.

Preparation

No preparation is required for this activity.

Task

Complete the following steps:

1. Examine the flow of events in the solution Use Case form from the lab exercise in module 4.
2. Create an Activity diagram to represent the flow of events.
3. Start your diagram with a start node.
4. Beginning with the first activity, represent each activity from the Use Case form as an activity node. You may represent a complex set of activities as one activity node in the main activity diagram, then show the internal activities in another activity diagram.

Note – Start with a simple diagram using branching, iteration, and concurrency. Then add swimlanes, interruptible activity regions, and object flow where necessary

5. Identify whether you have discovered any new understanding, or problems with the previous artifacts.
6. Optionally, model another use case with an activity diagram.

Lab 6

Determining the Key Abstractions

Objectives

Upon completion of these activities, you should be able to:

- Identify candidate key abstractions from the project artifacts
- Use Class-Responsibility Collaboration (CRC) analysis to identify key abstractions from candidate key abstractions

Activity 1: Finding Candidate Key Abstractions

In this exercise, you find candidate key abstractions from the Hotel system's artifacts.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Using the use case forms, scenarios, and glossary descriptions from the previous lab, identify the candidate key abstractions and list them in a candidate key abstraction template form.
2. Using the use case forms, scenarios, and glossary descriptions from the example in Module 6 of the Student Guide, identify any additional candidate key abstractions and then add them in a candidate key abstraction template form.

Activity 2: Finding Key Abstractions Using CRC

In this exercise, you identify key abstractions using CRC.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Select a candidate key abstraction from the candidate key abstraction solution of the first activity in this module.
2. Identify whether there is a use case in which the candidate key abstraction is prominent. If the candidate key abstraction is not prominent in a use case, it probably is not a key abstraction.
3. Scan the relevant project artifacts to identify responsibilities and collaborators for the candidate key abstraction. If the candidate key abstraction does not have any responsibilities or collaborators, it probably is not a key abstraction.
4. Document any responsibilities and collaborators using a CRC card.
5. Modify your candidate key abstractions form according to the information found during the CRC process. Write down the reasons for any candidate key abstraction not qualifying as a key abstraction.

Lab 7

Constructing the Problem Domain Model

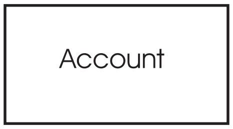
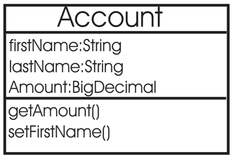
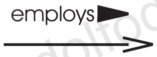
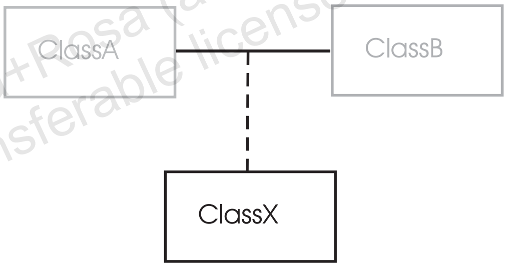
Objectives

Upon completion of this module, you should be able to:

- Identify the essential elements in a Class diagram
- Extend the Domain model class diagram of the Hotel System by adding key abstractions
- Identify the essential elements in an Object diagram
- Validate the Domain model using one or more Object diagrams

Activity 1: Identifying Class Diagram Elements

Self-Check – Write the name of each Class diagram symbol in the space allotted next to each symbol.

Symbol	Symbol Name
	
 Account firstName:String lastName:String Amount:BigDecimal getAmount() setFirstName()	
1..*	
	
	

Activity 2: Extending a Class Diagram

In this exercise, you extend the Domain model class diagram of the Hotel System by adding key abstractions identified during CRC analysis.

Preparation

Copy the classes from the Domain model class diagram of the Hotel system that is shown in Figure 7-1.

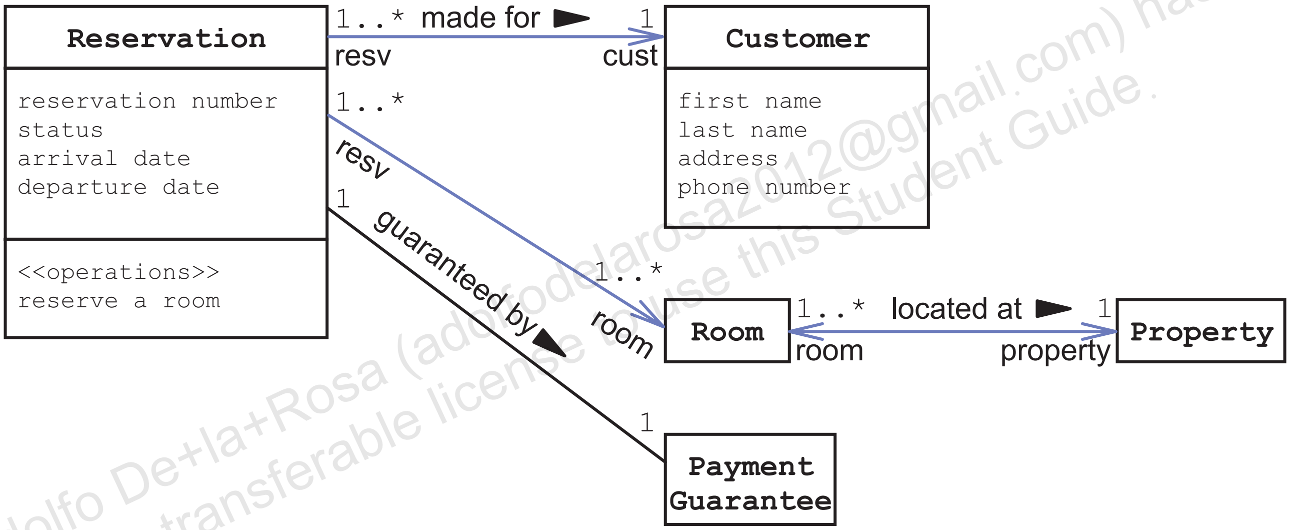


Figure 7-1 Domain Model Class Diagram of the Hotel System

Ensure that you leave space for new classes, attributes, and operations.

Note – This activity is best performed in groups by using a whiteboard or a flip chart.

Tasks

Complete the following steps:

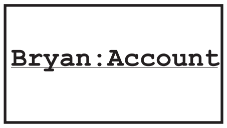
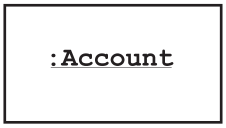
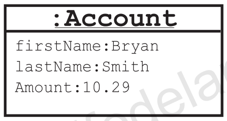
1. Add new class nodes for each key abstraction that you found in"Activity 2: Finding Key Abstractions Using CRC" on page L6-3 of this workbook. Ensure that you also include a list of known attributes and responsibilities.

   Hint: Start simple by creating one diagram containing class names and association links between collaborators, leaving space for adding attributes and operations. When the shape (topology) of the Class diagram solidifies, create the final version of the diagram with attributes and operations.

2. Add new class nodes that are documented in the `exercises/DomainModel/AdditionalKeyAbstractions.pdf` file.
3. Add associations between collaborating classes.
4. Add relationship names along with an arrow indicating the direction to read the association.
5. Add association multiplicity.
6. Add role names, but only if they improve the clarity of the diagram.

Activity 3: Identifying Object Diagram Symbols

Self-Check – Write the name of each Object diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	
	
 :Account firstName:Bryan lastName:Smith Amount:10.29	

Self-Check – Select the statement or statements about UML Object diagrams that are TRUE.

a. ___ Object diagrams can be validated using Class diagrams.

b. ___ Object diagrams show runtime links.

c. ___ Object symbols can have three compartments: name, attribute, and an operations compartment.

d. ___ Object diagrams often show object state.

e. ___ An Object diagram is an instance of a class diagram.

Activity 4: Validating a Class Diagram

In this exercise, you validate a Class diagram using one or more Object diagrams.

Preparation

Choose a subset of classes that you want to model as objects. This subset of classes should be based on the classes used in the primary (successful) scenario that you created for the Check In use case in "Activity 1: Examining Use Case Scenarios" on page L4-2 of this workbook.

Tasks

Complete the following steps:

1. Draw an object node for each key abstraction in the scenario.
2. Draw links between collaborating objects.
3. Follow a partial flow of the chosen scenario, changing your object model as the flow progresses.
4. Compare the Object diagram to the Domain model.

Lab 8

Transitioning from Analysis to Design Using Interaction Diagrams

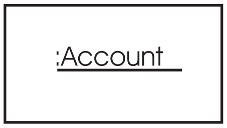
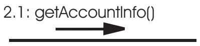
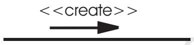
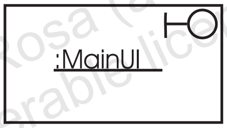
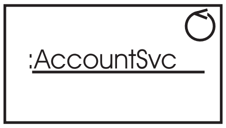
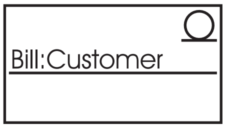
Objectives

Upon completion of these activities, you should be able to:

- Identify the essential elements of a Communication diagram
- Create a Communication diagram
- Identify the essential elements of a Sequence diagram
- Create a Sequence diagram

Activity 1: Identifying Communication Diagram Elements

Self-Check – Write the name of each Collaboration diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	
	
	
	
	
	

Activity 2: Creating a Communication Diagram

In this exercise, you create a Communication diagram for the Check-In use case.

Preparation

You will need the Check-In Use Case Form from Lab 4.

Tasks

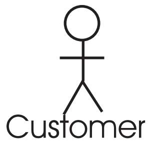
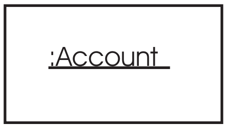
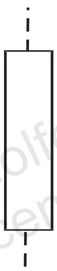
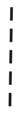
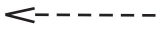
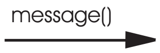
Complete the following steps:

1. Read the main flow of events in the Check-In Use Case Form.
2. Place the actor for the use case on the diagram.
3. Analyze the flow of events. For every action in the use case:
   a. Identify and add boundary components.
   b. Identify and add control components.
   c. Identify and add entity components.
   d. Draw the associations between these components.
   e. Label the actions performed by each component to satisfy the interactions in the use case.
4. Optionally create another diagram or fragment diagram for another scenario.

Note – If your group prefers, you may do activity 4 before doing activity 2. In which case you may find it easier to convert from the Sequence diagram created in activity 4.

Activity 3: Identifying Sequence Diagram Elements

Self-Check – Write the name of each Sequence diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	
	
	
	
	
	

Activity 4: Creating a Sequence Diagram

In this exercise, you create a Sequence diagram for the Check-In use case.

Preparation

You will need the Check-In Use Case Form from Lab 4.

Tasks

Complete the following steps:

1. Read the main flow of events in the Check-In Use Case Form.
2. Arrange the components at the top of the Sequence diagram.

   Hint: Put actors and boundary components on the left, followed by service components in the middle, and entities on the right.
3. Add message links and activation bars for each message in the first activity.
4. Repeat Step 3 for each activity in the use case until the task is complete.

Note – If you have done activity 2 then you may find it easier to convert from the Communication diagram.

Lab 9

Modeling Object State Using State Machine Diagrams

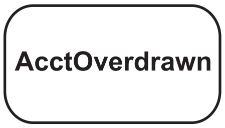
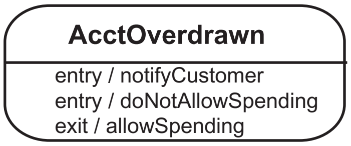
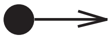
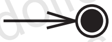
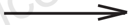
Objectives

Upon completion of these activities, you should be able to:

- Identify the essential elements in a State Machine diagram
- Create a State Machine diagram for a complex object

Activity 1: Identifying State Machine Diagram Elements

Self-Check – Write the name of each State Machine diagram symbol in the space allotted next to each symbol.

Symbol	Symbol Name
	
 AcctOverdrawn entry / notifyCustomer entry / doNotAllowSpending exit / allowSpending	
	
	
	

Activity 2: Creating a State Diagram

In this exercise, you create a State diagram for a banking example.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the following description for a complex `Account` object.
   - All accounts start with a balance of zero.
   - An account with a balance greater than or equal to zero has a state of "Active." When accounts are active, a customer may deposit any amount of money, withdraw any amount of money (as long as the overdraft limit is not reached), and transfer money from the Active account to another account.

Note – An overdraft limit is a set amount that a customer's account can have in short-term (less than 3 months) debt without being frozen. It is similar to a short-term loan from the bank. For example, if the bank gives their customers an overdraft limit of $100.00, it means that a customer's account balance can be –$99.99 without being frozen. But as soon as the account goes to –$100.00 or below, it is frozen and the customer can no longer withdraw money.

   - If the balance in an account becomes less than zero (but greater than the overdraft limit such as in the previous example), the account state is changed to "Overdrawn." When an account is overdrawn:
     - A customer can still withdraw money from an overdrawn account as long as the amount withdrawn does not cause the balance to go below the overdraft limit.
     - A customer is sent monthly notices about the status of the account.
     - A customer is notified of being overdrawn when they withdraw money from an automated teller machine (ATM).

    - A customer cannot transfer money from the overdrawn account to another account.
    - A customer cannot close the account.
- If the status remains "Overdrawn" for more than 3 months, the account state is changed to "Frozen." When an account is frozen:
    - The customer cannot withdraw money from the account.
    - The customer can deposit money into the account.
    - The customer is sent monthly notices about the status of the account.
    - The customer is notified by the ATM that they must meet with the bank's manager to unfreeze the account.
    - The customer cannot close the account.
    - The bank manager might decide to close all of the customer's accounts and file legal action against the customer. Should this happen, the customer cannot open any accounts with this bank again.
- Whenever the account balance falls below zero by an amount equal to the overdraft limit, the account state is changed to "Frozen." See previous bullet for information on the Frozen status.
- If an account is closed by the customer, the account state is changed to "Closed" and the customer can no longer use the account.

2. Choose the initial and final states for the object. Specify the preconditions of the initial state and the post conditions of the final state.
3. Choose the stable states of the object.
4. Specify the partial ordering of stable states over the lifetime of the object.
5. Specify the events that trigger the transitions between the states of the object. Specify transition actions (if any).
6. Specify the actions with a state (if any).

Lab 10

Applying Design Patterns to the Design Model

Objectives

Upon completion of these activities, you should be able to:

- Apply the Composite Reuse Principle (CRP)
- Apply the Strategy pattern
- Apply the Observer pattern
- Apply the State pattern

Activity 1: Applying CRP

In this exercise, you apply CRP to a case study.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the following information pertaining to a Vehicle Information System:

   You need a system to record the details of various types of vehicle that can move on land, water, air, or a combination of these terrains. The information that is required to be recorded for each vehicle includes the following:

   - weight
   - Maximum dimensions (h,l,w), where h stands for height, l stands for length, and w stands for width
   - Power sources with fuel type
   - Maximum speed on each terrain
   - Certification information (airworthiness, seaworthiness, and land vehicle inspection).

   Your job is to ensure that the system can accommodate these different types of vehicle and can be easily modified for new terrains and power sources in the future.
2. Draw a single Class diagram that shows how CRP can be applied to the problem to create a flexible solution.

Activity 2: Applying the Strategy Pattern

In this exercise, you apply the Strategy pattern to the Hotel System case study.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the following information pertaining to the Hotel System case study.

   The marketing department of the hotel will implement various discount offers that are used to calculate the quoted price. These offers may include the following:

   - Two nights stay for the price of one night, provided it is a weekend—that is, Friday, Saturday, or Sunday.
   - 5 nights stay for the price of 4 nights at any time during the offer period.
   - 10% discount during the offer period.

   These discount offers will vary between the hotel properties.

   Your job is to ensure that the Hotel System can accommodate algorithms for these different offers. You must also ensure that new offers with different algorithms can be easily added in the future.
2. Draw a single Class diagram that shows the Strategy pattern solving the problem in the case study.

Activity 3: Applying the Observer Pattern

In this exercise, you apply the Observer pattern to the Hotel System case study.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the following information pertaining to the Hotel System case study:

   When a customer vacates a room after checking out, various individual members of the staff should be notified immediately and departmental display screens should be updated. The individual staff members should be notified by a message sent to a portable hand-held device. These members of the staff should include a chambermaid, the housekeeping supervisor, the mini-bar stockist, and room maintenance. The departmental display screens should show the status of rooms in various departments including housekeeping and maintenance.

   Your job is to ensure that the Hotel System includes the functionality to notify the concerned members of the staff and update the departmental display screens as soon as possible after the room becomes vacant. You must also ensure that when a room becomes vacant, other notifications that may be required in the future can be added easily.

2. Draw a single Class diagram that shows the Observer pattern solving the problem in the case study.

Activity 4: Applying the State Pattern

In this exercise, you complete the following tasks:

- Simplify pseudo code by refactoring it based on the State pattern.
- Create a Class diagram that implements the State pattern for the previous example.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Examine the following pseudo code. This large block of code is currently used in the Account object to ensure that it acts appropriately when a customer attempts to withdraw money.

```
withdrawlMethod() {

   if (accountStatus = "Active") {
      allow customer to withdrawl up to (OverdraftLimit
      + currentBalance);
   }
   else if (accountStatus = "Overdrawn"{
      DisplayMessage("You are Overdrawn");
      allow customer to withdrawl up to (OverdraftLimit
      + currentBalance);
   }
   else if (accountStatus = "Frozen") {
      DisplayMessage("Account Frozen, you cannot
      withdrawl money, please visit with bank manager");
   }
   else {
      DisplayMessage("Account is closed, you cannot
      withdrawl money."
   }
```

2. Given what you know about the State pattern, how would you refactor this code?

3. Create a Class diagram that represents your refactoring (based on the state pattern). Be sure to annotate each class in the diagram to show where the code goes.

Lab 11

Introducing Architectural Concepts and Diagrams

Objectives

Upon completion of these activities, you should be able to:

- Identify the tiers, layers, and systemic qualities for the SunTone Architectural Methodology
- Identify the essential elements of a Component diagram
- Identify the essential elements of a Deployment diagram
- Create a high-level Deployment diagram

Activity 1: Identifying Tiers, Layers, and Systemic Qualities

Self-Check – Match the layers with their definitions.

Layer	**Definition**
Application	Provides the APIs that application components implement
Virtual Platform	Consists of the operating system
Upper Platform	Includes computing components such as servers, storage, and network devices
Lower Platform	Provides a concrete implementation of components to satisfy the functional requirements
Hardware Platform	Consists of products such as web and containers and middleware

Self-Check – Match the tiers with their definitions.

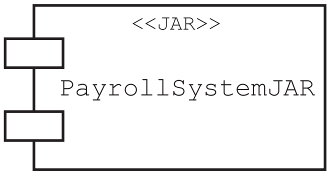
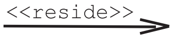
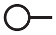
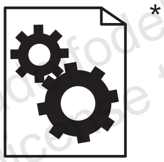
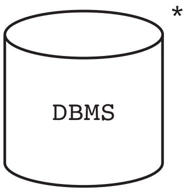
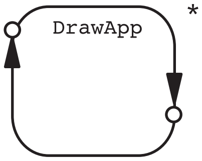
Tier	**Definition**
Client	Provides services and entities
Presentation	All back-end components, such as a Database Management System (DBMS) or Enterprise Information System (EIS)
Business	Usually described as “thin”; often is a web browser.
Integration	Provides the Hyper Text Markup Language (HTML) pages and forms sent to a web browser and process the user’s requests
Resource	Provides components that tie the business tier to the resource tier

Self-Check – Match the systemic qualities with their definitions.

Systemic Quality	Definition
Developmental	Addresses the requisite qualities as the system evolves
Manifest	Addresses the qualities reflected in the execution of the system
Evolutionary	Addresses the requisite qualities in production
Operational	Addresses the requisite qualities during system development

Activity 2: Exploring Component Diagrams

Self-Check – Write the name of each Component diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	
	
	
	
	
	

Self-Check – Select the statement or statements about UML Component diagrams that are TRUE.

a. ___ A component represents a software unit.

b. ___ There are descriptor and instance forms of Component diagrams.

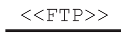
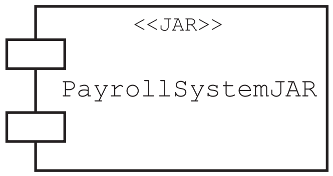
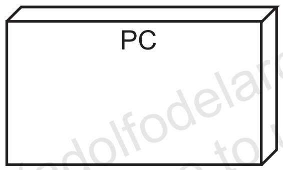
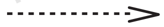
c. ___ Components cannot be abstract.

d. ___ Component diagrams show the organizations and dependencies among components.

e. ___ You can create your own component icons to extend the UML.

Activity 3: Understanding Deployment Diagrams

Self-Check – Write the name of each Deployment diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	
	
	
	

Self-Check – Select the statement or statements about UML Deployment diagrams that are TRUE.

a. ___ You can assign your own icons to represent hardware in Deployment diagrams.

b. ___ There are descriptor and instance forms of Deployment diagrams.

c. ___ Descriptor Deployment diagrams show a particular deployment of a system.

d. ___ The «deploy» stereotype can be used to document components within a node.

e. ___ There is only a descriptor form of Deployment diagrams.

Activity 4: Creating a High-Level Deployment Diagram

In this exercise, you create a high-level Deployment diagram for the Hotel System case study.

Preparation

Read the following requirements pertaining to the Hotel System case study:

Hotel System – Additional Abstract of Requirements

The database server and the Web server must be on separate hardware nodes.

Hotel System – Recap of System Requirements

Note – The recap of system requirements are an abstract of the requirements provided in Lab 3 and Lab 10.

The booking agent (internal staff) must be able to manage reservations on behalf of customers who telephone or e-mail with reservation requests. The majority of these requests will make a new reservation, but occasionally they will need to amend or cancel a reservation. A reservation holds one or more rooms of a room type for a single time period, and must be guaranteed by either an electronic card payment or the receipt of a purchase order for corporate customers and travel agents. These payment guarantees must be saved for future reference.

A reservation can also be made electronically from the Travel Agent system and also by customers directly via the internet.

The receptionist must be able to check in customers arriving at the hotel. This action will allocate one or more rooms of the requested type. In most cases, a further electronic card payment guarantee is required.

The receptionist must be able to check out customers before they leave the hotel. During the check-out procedure, a customer's bill is calculated and presented to the customer. In most cases, immediate payment is required to settle the bill for any outstanding room, meal, or other chargeable items added to the bill during the stay. The most common payment method is in the form of a credit card or a debit card.

Some customers will have all or part of their bill guaranteed by a purchase order. Invoices for these charges will be generated daily and either printed or sent electronically to the company or travel agent.

The room telephone is enabled upon check-in and disabled upon check-out.

The marketing staff can publish current offers to their customer base via e-mail.

The hotel group does not currently have a loyalty scheme. However, the group is affiliated with airlines, car rental companies, and other reward points schemes. For example, customers traveling on affiliated airlines will earn airmiles. Customers that have registered a loyalty card will have points awarded to that external scheme when their bill for a reservation is settled.

Customers who provided a phone number must be sent a text message during the morning of their due arrival date. This is a courtesy text message reminding them of their booking.

Cleaners can request a list of rooms that have been vacated. Once a vacated room has been prepared for the next customer, the cleaning staff must mark the room as ready for use.

When a customer vacates a room after checking out, various individual members of the staff should be notified immediately and departmental display screens should be updated. The individual staff members should be notified by a messages sent to a portable hand-held device. These members of the staff should include a chambermaid, the housekeeping supervisor, the mini-bar stockist, and room maintenance. The departmental display screens should show the status of rooms in various departments including housekeeping and maintenance.

Your job is to ensure that the Hotel System includes the functionality to notify the concerned members of the staff and update the departmental display screens as soon as possible after the room becomes vacant. You must also ensure that when a room becomes vacant, other notifications that may be required in the future can be added easily.

Tasks

Complete the following steps:

1. Create a high-level Deployment diagram from the selected architecture type and technologies.
2. List the applications mentioned in the case study.

   Remember, some applications might exist within the context of another application. For example, an applet is run within a Web browser.
3. List the actors and hardware nodes
4. Draw relationships between actors and the hardware nodes. Draw relationships between hardware nodes.
5. Label the hardware nodes.
6. Place the applications identified in Step 2 in the appropriate hardware node.

Lab 12

Introducing the Architectural Tiers

Objectives

Upon completion of these activities, you should be able to:

- Update and extend the tiers and layers package diagram used in the example case study for the Hotel System.
- Extend the DAOFactory used in the example case study to allow a change of database and the integration of external resources

Activity 1: Update and Extend a Tiers and Layers diagram

In this exercise, you update and extend the tiers and layers package diagram used in the example case study for the Hotel System.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Open the `TiersAndLayers.pdf` file located in the `exercises/ArchTiers/` directory.
2. Update and extend the Tiers and Layers diagram by using the information provided in the `TiersAndLayers.pdf` file.

Activity 2: Create DAOFactory and DAO Classes

In this exercise, you create a Class diagram for the DAOFactory and DAO classes required by the Hotel System.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Create a Class diagram containing the DAO classes that will allow you to access two different data sources. You only need to show a representative number of DAOs to support the Reservation and Customer entities.
2. Add the DAOFactory classes you will require to create the DAO objects.

Lab 13

Refining the Class Design Model

Objectives

Upon completion of these activities, you should be able to:

- Refine attributes
- Encapsulate attributes
- Refine associations
- Refine methods
- Declare constructors
- Review your class design to ensure that it maintains high cohesion and low coupling
- Create components with well-defined interfaces

Activity 1: Refining Attributes

In this exercise, you will:

- Refine the class attributes in the Design model.
- Encapsulate the class attributes in the Design model.

Preparation

Review the Class diagram of the Hotel System case study that you created in "Activity 2: Extending a Class Diagram" on page L7-3 of this workbook. This Class diagram is the starting point for this activity.

Task 1 – Refine Attributes

Complete the following steps:

1. Refine the class attributes in the Class diagram.
2. Refine the name of each attribute to follow convention.
3. Identify the data type for each attribute. Create new `<<utility>>` types if necessary. For example, the arrival date and departure date could be combined into a `DateRange` class with a method to check for overlapping dates and a method to return the duration.
4. Designate an initial value for each attribute (if applicable).
5. Identify the property for each attribute (`changeable`, `addOnly`, or `frozen`).

Task 2 – Encapsulate Attributes

Complete the following steps:

1. Encapsulate the class attributes in the Class diagram.
2. Make each attribute private.
3. Add public accessor methods for all attributes. Some attributes you might want to keep hidden, if so then do not create an accessor method for this attribute.

4. Add public mutator methods for all attributes. Do not add a mutator method for any attribute that is marked with the property of “frozen.”

5. If appropriate, identify any derived attributes and create accessor or mutator methods to interface with these attributes.

Activity 2: Refining Associations using Aggregation and Composition

In this exercise, you will refine the class associations using aggregation or composition where appropriate.

Preparation

You will use the Class diagram of the Hotel System case study that you updated in the previous activity as the starting point for this activity.

Task – Adding Aggregation and Composition

Complete the following steps:

1. Review each class association and determine whether aggregation or composition would be more appropriate.
2. Refine you class diagram to add any aggregation or composition found in step 1.

Activity 3: Refining the Direction of Traversal

In this exercise, you will:

- Determine the direction of traversal of associations.
- Elaborate class methods to maintain the associations.

Preparation

You will use the Class diagram of the Hotel System case study that you updated in the previous activity as the starting point for this activity.

Task – Refine the Direction of Traversal of Associations

Complete the following steps:

1. Identify the direction of traversal (navigation).
2. Add association methods.

Activity 4: Refining Business Methods and Constructors

In this exercise, you will:

- Elaborate business methods.
- Declare constructors.
- Annotate methods and constructors.

Preparation

You will use the Class diagram of the Hotel System case study that you updated in the previous activity as the starting point for this activity.

Task – Refine Business Methods and Constructors

Complete the following steps:

1. Add or define new business methods.
2. Add constructors.
3. Annotate the methods and constructors that you added in steps 1 and 2.

Activity 5: Checking your Class Diagram for High Cohesion and Low Coupling

In this exercise, you will review your Class diagram to ensure that it has high cohesion and low coupling.

Preparation

You will use the Class diagram of the Hotel System case study that you updated in the previous activity as the starting point for this activity.

Task – Review Cohesion and Coupling

Complete the following steps:

1. Review your Class diagram, and discuss in your group whether the diagram has high cohesion.
2. Review your Class diagram, and discuss in your group whether the diagram has low coupling.
3. Resolve any issues identified in steps 1 and 2 by modifying your Class diagram.

Activity 6: Creating Components with Interfaces

In this exercise, you will create cohesive components from groups of classes that should work together.

Preparation

You will use the Class diagram of the Hotel System case study that you updated in the previous activity as the starting point for this activity. In addition, you will use the service classes that you created in "Activity 2: Creating a Communication Diagram" on page L8-3 and "Activity 4: Creating a Sequence Diagram" on page L8-5 of this workbook to specify names for the interfaces.

Task – Define Components and Interfaces

Complete the following steps:

1. Create components for the different areas of common functionality.

Note – For example, in the lending library system, we created the `Membership` and `BookInventory` components.

2. For each component created in step 1, add any provided interfaces.

Note – Provided interfaces are often interfaces of the service classes.

3. For each component created in step 1, add any required interfaces that the component requires other components to provide.

Lab 14

Overview of Software Development Processes

Objectives

Upon completion of these activities, you should be able to:

- Describe two characteristics of common methodologies
- Define the five object-oriented methodologies
- Develop an iteration plan
- Select the appropriate methodology for case studies

Activity 1: Identifying Methodology Characteristics

Self-Check – Select the statement or statements about Use-Case-driven methodologies that are true.

a. ___ Focus on relationships between actors and the system

b. ___ Based on the notion that software performs activities for users

c. ___ Uses non-functional requirements to drive structure of the system

d. ___ Must be iterative

e. ___ Focus on the systemic qualities, such as reliability and scalability

Self-Check – Select the statement or statements about Architecture-centric methodologies that are true.

a. ___ Focus on relationships between actors and the system

b. ___ Based on the notion that software performs activities for users

c. ___ Uses non-functional requirements to drive the architecture of the system

d. ___ Must be iterative

e. ___ Focus on the systemic qualities, such as reliability and scalability

Activity 2: Defining the Five Object-Oriented Methodologies

Self-Check – Match the object-oriented software development methodology terms with their definitions.

Term	**Definition**
Waterfall	An iterative software development process, created by Booch, Jacobson, and Rumbaugh, that is freely available for use.
eXtreme Programming (XP)	Coding and testing are the key activities within this methodology.
Unified Software Development Process (UP)	Team-oriented framework, where each Sprint produces a working version of the software.
Rational Unified Process (RUP)	This methodology uses a single phase in which all workflows proceed in a linear fashion.
Scrum	A commercial implementation of the UP methodology.

Activity 3: Producing an Iteration Plan

In this exercise, you produce an iteration plan for the Hotel System case study.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Re-assess the priorities and risks for the Use Case form that you created in "Activity 2: Creating Use Case Forms" on page L4-4 of this workbook.
2. Assess the priorities and risks for the remaining use cases that you discovered in "Activity 3: Refining the Use Case Diagram" on page L3-7 of this workbook.
3. Assess the architectural significance of each use case.
4. Construct an iteration plan based on the results of steps 1 to 3.

Activity 4: Selecting Methodologies

In this exercise, you select the appropriate methodology for each case study.

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Read the following scenarios.
2. Select and justify the methodology that is appropriate for each case study. Note: there can be more than one methodology for a scenario.

ACME Insurance Case Study

ACME Insurance Corporation (AIC) wants to create a system for entering and storing insurance policies for its over 200,000 world-wide customers. AIC has a small development team of 4 people who will work on this project, consisting of one project manager (who also writes documentation) and three engineers. The project manager has been asked to keep documentation on all phases of the project using the newest International Standards Organization (ISO) documentation process.

Methodology Selection ________________

Justification ____

____

____

NoLycra.com Case Study

NoLycra.com is a small startup company that produces cycling jerseys. The company is located in a small warehouse that includes all business functions (sales, marketing, manufacturing, shipping, and so on). NoLycra.com has become known for its unique jersey designs featuring bright colors and non-traditional fabrics. NoLycra.com has recently hired two computer engineers from a rival company to create an inventory control system that tracks its customer's orders and its inventory levels. These engineers were recruited by NoLycra.com because they had created an inventory control system at their previous company to help get products quickly to market.

Methodology Selection ________________

Justification ____

____

____

Bravo Case Study

A contractor has been hired by Bravo software to do analysis and design for an eXtensible Markup Language (XML)-based system that can store solutions to common technical support issues. The contractor will be working with a small team of employees to design and implement the system. Most of these employees have technical experience (including technical writing, engineering, testing, and graphical abilities), but they were hired as Technical Support Analysts (TSAs). One of the TSAs suggests that the team use Rational Rose to help with the project because Bravo owns a license to this product. The same TSA also reminds the team that they must follow the Product Life Cycle process when developing and delivering the new system.

Methodology Selection ________________

Justification ____

____

____

Lab 15

Overview of Frameworks

Objectives

Upon completion of these activities, you should be able to:

- Create a high level conceptual class diagram showing the classes and interfaces that would be required for a generic framework subsystem
- Discover potential domain neutral frameworks that could be shared across business domains

Activity 1: Creating a Conceptual Framework

In this exercise, you will complete the following tasks:

- Identify the generic classes that would be required for the framework
- Identify the specific domain classes that could use the framework
- Create a conceptual class diagram for the classes and relationships that were identified

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Identify the classes that would be required to build a framework for a generic resource allocation system, where resources are allocated for a period of time.
2. Draw a high level class diagram showing the generic classes and interfaces that would be required for the framework identified in step 1.
3. Draw domain specific classes that could use the framework that was created in step 2.

Activity 2: Identifying Potential Frameworks

In this exercise, you will identify potential frameworks that could be shared by a selection of business domains

Preparation

No special preparation is required for this activity.

Tasks

Complete the following steps:

1. Identify potential frameworks that could be shared, by the following applications: Hotel System, Car Rental System, and Private Hospital System.
2. Identify other applications that could use the frameworks discovered in step 1.
3. Optionally, model the frameworks identified in step 1, and model the specific uses of those frameworks.

Lab 16

Course Review

There are no activities for this module.

Object-Oriented Analysis and Design Using UML

Solutions to Labs

D61808BP21

Edition 2.1

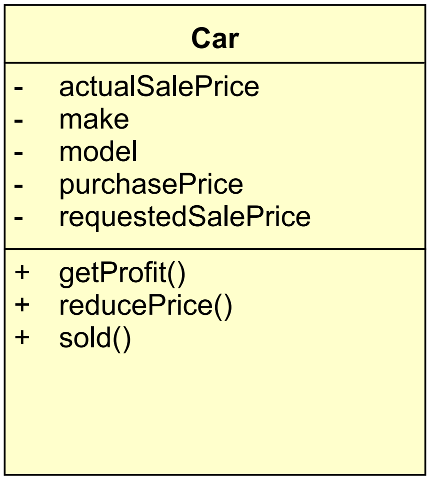
June 2011

ORACLE®

Table of Contents

Activity 1: Using Abstraction

a) Used Car Lot Context

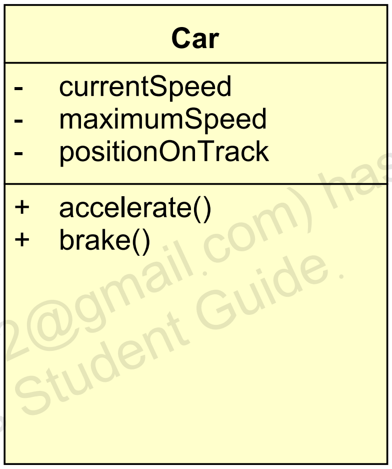


Car

- - actualSalePrice
- - make
- - model
- - purchasePrice
- - requestedSalePrice

- + getProfit()
- + reducePrice()
- + sold()

b) Motor Racing Circuit
Game Context

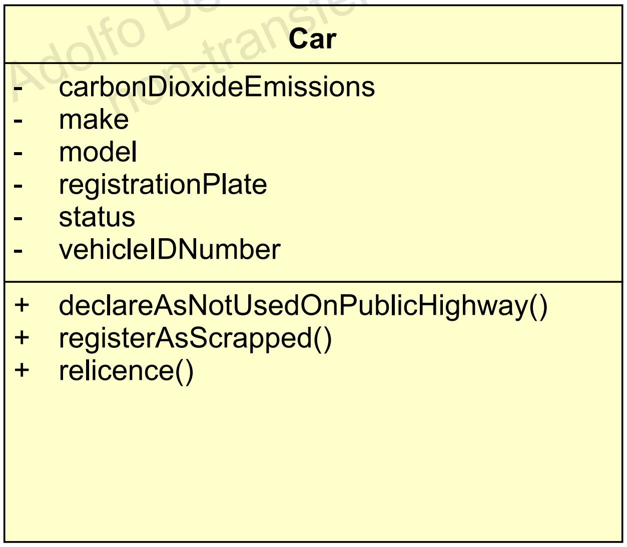


Car

- - currentSpeed
- - maximumSpeed
- - positionOnTrack

- + accelerate()
- + brake()

c) Vehicle Licensing
Registration Context



Car

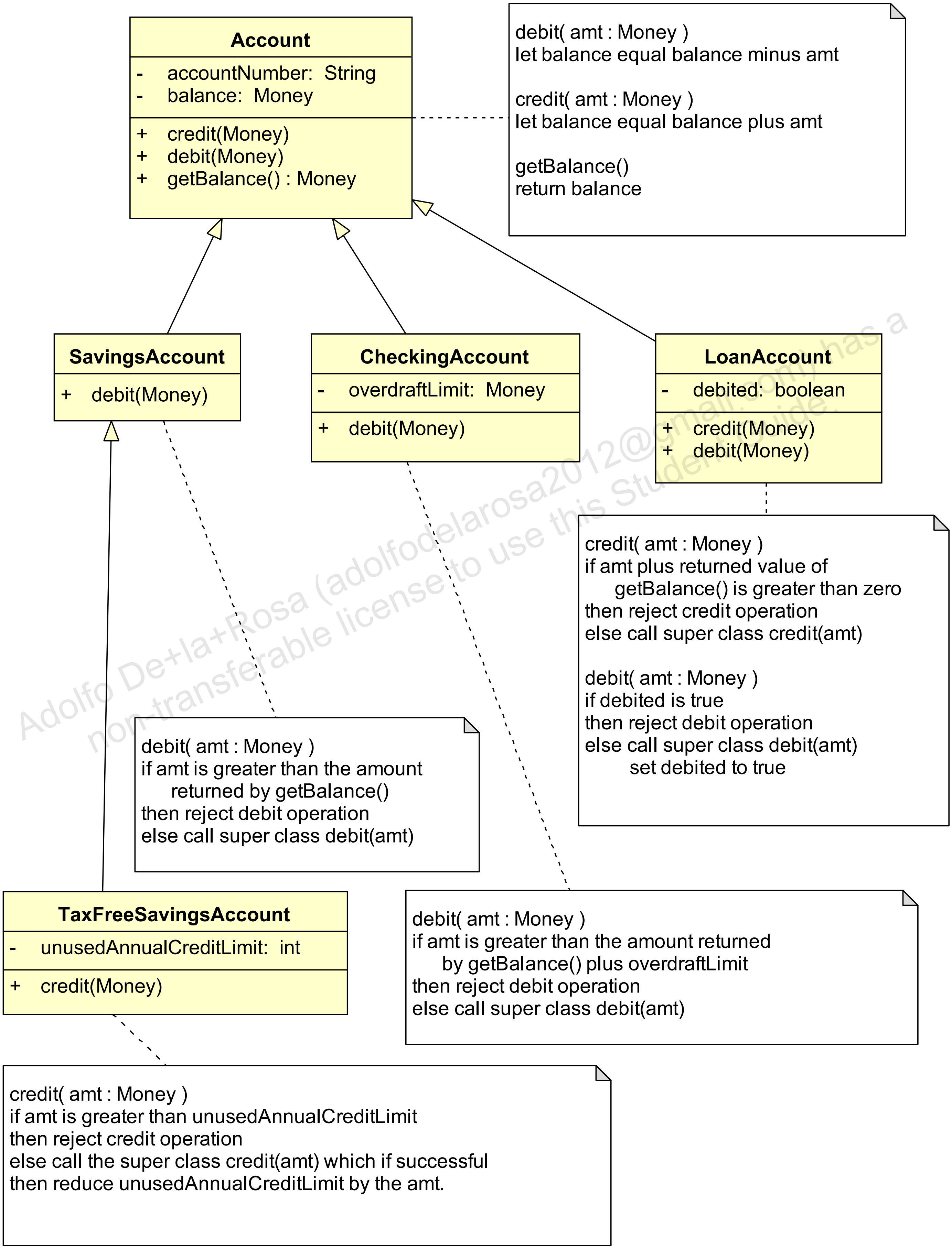
- - carbonDioxideEmissions
- - make
- - model
- - registrationPlate
- - status
- - vehicleIDNumber

- + declareAsNotUsedOnPublicHighway()
- + registerAsScrapped()
- + relicence()

Note 1: These are just some possible attributes and methods. You might have chosen other attributes and methods.

Note 2: Some of these attributes can be moved to another class. For example, carbonDioxideEmissions, make, and model can be moved to a CarType class.

Activity 2: Using Inheritance



Account

- - accountNumber: String
- - balance: Money
- + credit(Money)
- + debit(Money)
- + getBalance() : Money

```
debit( amt : Money )
let balance equal balance minus amt

credit( amt : Money )
let balance equal balance plus amt

getBalance()
return balance
```

SavingsAccount

- + debit(Money)

```
debit( amt : Money )
if amt is greater than the amount
    returned by getBalance()
then reject debit operation
else call super class debit(amt)
```

CheckingAccount

- - overdraftLimit: Money
- + debit(Money)

```
debit( amt : Money )
if amt is greater than the amount returned
    by getBalance() plus overdraftLimit
then reject debit operation
else call super class debit(amt)
```

LoanAccount

- - debited: boolean
- + credit(Money)
- + debit(Money)

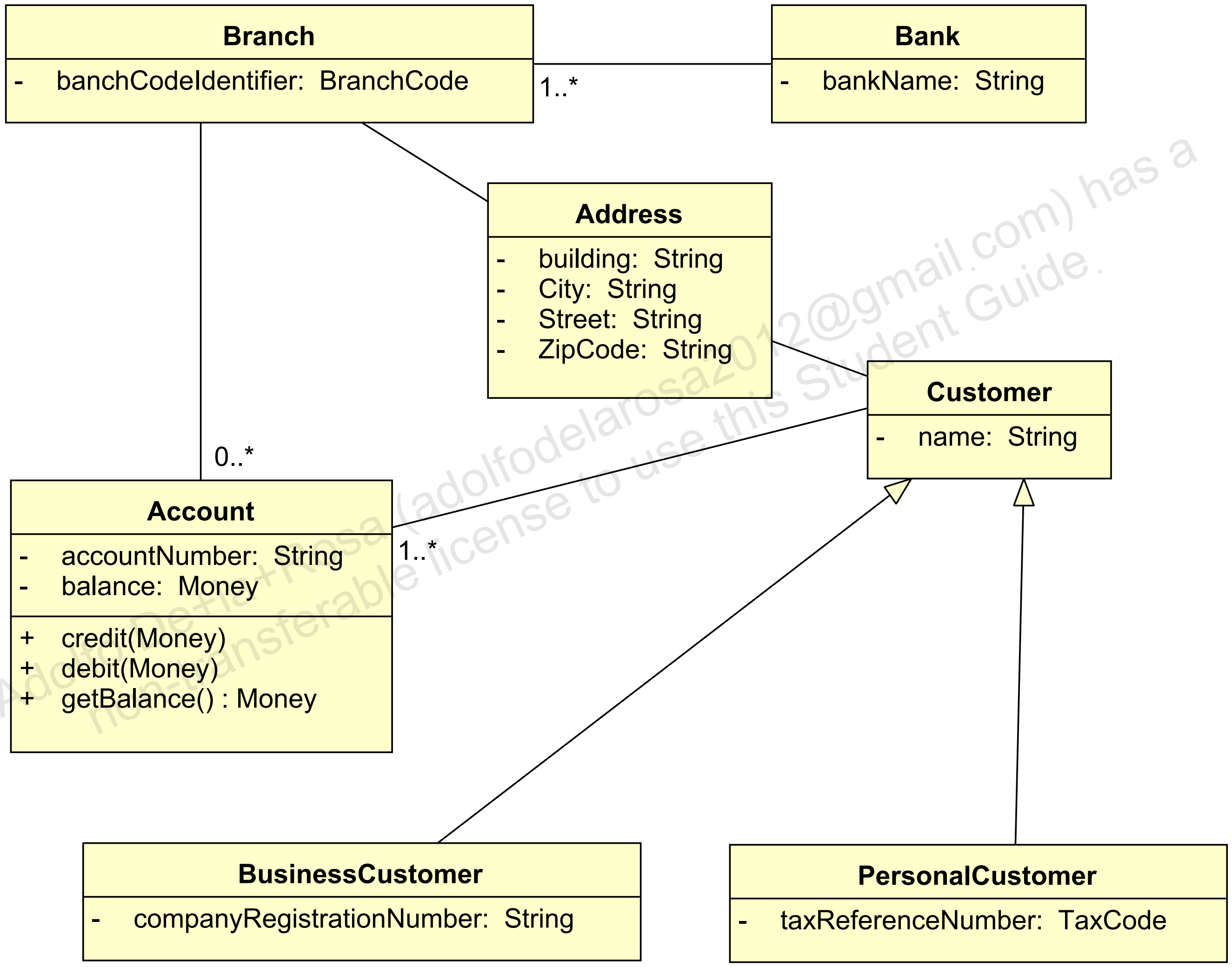
```
credit( amt : Money )
if amt plus returned value of
    getBalance() is greater than zero
then reject credit operation
else call super class credit(amt)

debit( amt : Money )
if debited is true
then reject debit operation
else call super class debit(amt)
        set debited to true
```

TaxFreeSavingsAccount

- - unusedAnnualCreditLimit: int
- + credit(Money)

```
credit( amt : Money )
if amt is greater than unusedAnnualCreditLimit
then reject credit operation
else call the super class credit(amt) which if successful
then reduce unusedAnnualCreditLimit by the amt.
```

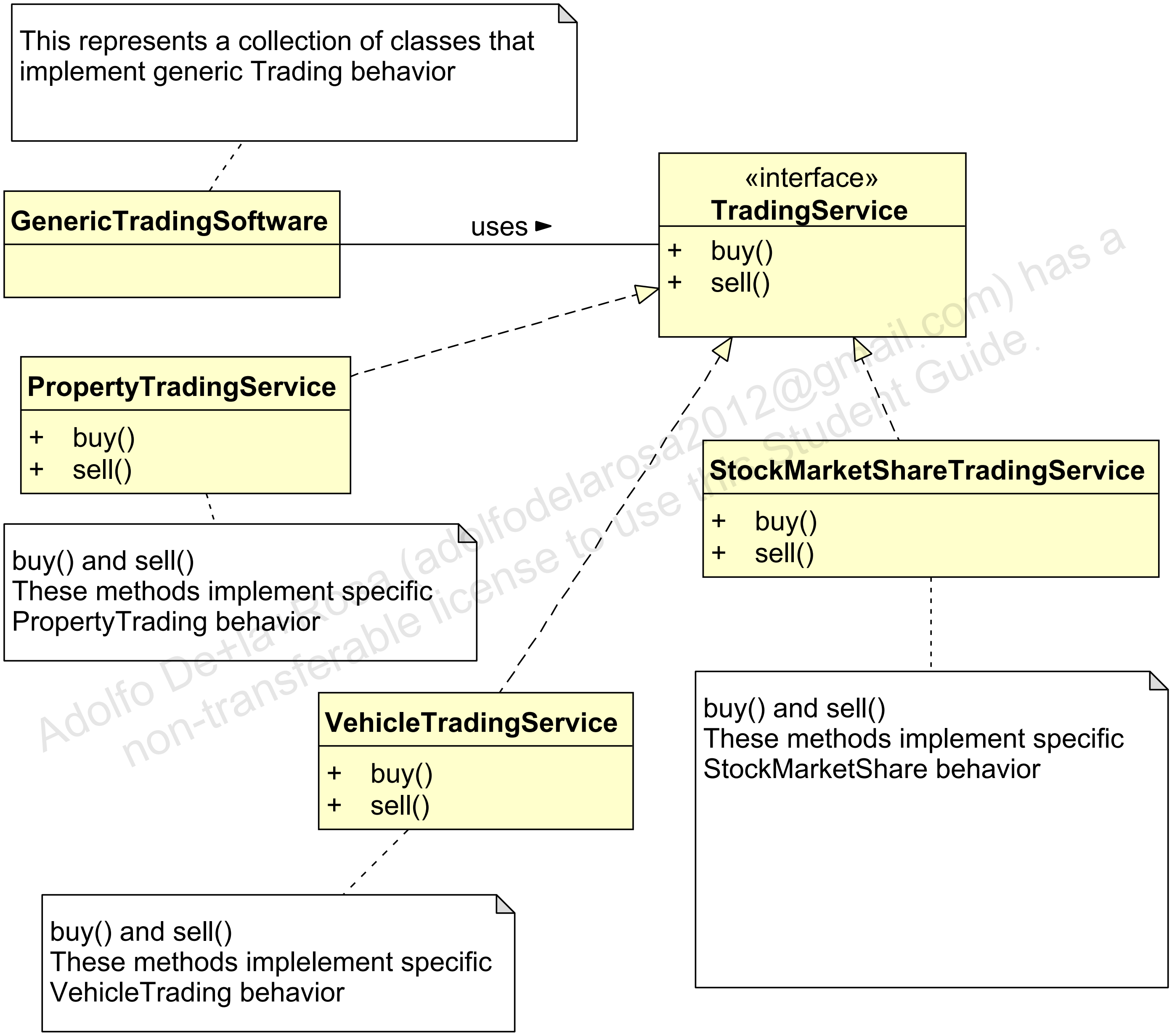
Activity 3: Using Delegation and Cohesion



Note 1:
This is just one possible solution.

Note 2:
The Address class would normally be a utility class shown as an attribute of type Address in the Customer and Branch classes.

Activity 4: Understanding the Benefits of Using Interfaces



This represents a collection of classes that implement generic Trading behavior

buy() and sell()
These methods implement specific PropertyTrading behavior

buy() and sell()
These methods implement specific StockMarketShare behavior

buy() and sell()
These methods implelement specific VehicleTrading behavior

Activity 5: Defining Object-Oriented Terminology

Self-Check – Match the object-oriented programming terms with their definitions.

Term	**Definition**
polymorphism	Identifying only those attributes and methods that are relevant to the class within the system.
object link	The ability to derive new classes that acquire attributes and methods from a base class.
inheritance	The blueprint for an object.
abstraction	Hiding internal methods and data from direct access from outside the class.
object	A relationship between two objects (instances).
delegation	The measure of how much an entity (component or class) supports a singular purpose within a system.
encapsulation	The ability to determine the actual method called based on the object's subtype.
association	An instance of a class.
class	Offloading some coherent responsibilities to another component (one or more classes) or method.
coupling	A relationship between two classes.
cohesion	The degree to which classes within our system are dependent on each other.

Answers: abstraction, inheritance, class, encapsulation, object link, cohesion, polymorphism, object, delegation, association, coupling

Activity 1: Defining Workflows

Self-Check – Match each workflow with its description.

Term	Definition
Requirements Gathering	Model the high-level structure of the system, paying particular attention to the NFRs and mitigation of risk.
Requirements Analysis	Install the implementation into the production environment.
Architecture	Build/Code the software components defined in the Solution model.
Design	Determine the requirements of the system by meeting the business owner and users of the proposed system.
Implementation	Ensure the implementation meets the expectations defined in the requirements.
Testing	Create a Solution model of the system that satisfies the functional requirements.
Deployment	Analyze, refine, and model the requirements of the system.

Answers: Architecture, Deployment, Implementation, Requirements Gathering, Testing, Design, Requirements Analysis

Activity 2: Identifying Characteristics of the Requirements Gathering Workflow

Self-Check – Select the characteristics of the Requirements Gathering workflow mentioned during the lecture.

a. ___ This workflow includes meeting the business owner and users of the proposed system to understand their requirements.

b. ___ This workflow requires you to model the high-level system structure to satisfy the non-functional requirements (NFRs).

c. ___ The purpose of this workflow is to determine what the system must do.

d. ___ You will create a business domain class diagram showing the required business classes during this workflow.

e. ___ You will create initial Use Case diagrams during this workflow.

Answers: a, c, e.

- Answer b is wrong because it is a characteristic of the Architecture workflow.
- Answer d is wrong because it is a characteristic of the Requirements Analysis workflow.

Activity 3: Identifying Characteristics of the Requirements Analysis Workflow

Self-Check – Select the characteristics of the Requirements Analysis workflow mentioned during the lecture.

a. ___ This workflow includes recording Use Case scenarios.

b. ___ This workflow starts with analyzing and Use Case scenarios.

c. ___ The purpose of this workflow is to model how the system will support the use cases.

d. ___ You will create a business domain class diagram showing the required business classes during this workflow.

e. ___ You will create a detailed Deployment diagram showing the system architecture during this workflow.

Answers: b, d.

- Answer a is wrong because this activity is part of the Requirements Gathering workflow.
- Answer c is wrong because this is the purpose of the Design workflow.
- Answer e is wrong because Deployment diagrams are created later in the process.

Activity 4: Identifying Characteristics of the Architecture Workflow

Self-Check – Select the characteristics of the Architecture workflow mentioned during the lecture.

a. ___ You will create detailed Deployment diagram during this workflow.

b. ___ The purpose of this workflow is to model the high-level structure of the system to satisfy the NFRs.

c. ___ You will create a tiers and layers diagram during this workflow.

d. ___ The purpose of this workflow is to model the high-level structure of the system to satisfy the FRs.

e. ___ You will refine the Design model during this workflow.

Answers: a, b, c.

- Answer d is wrong because the focus during the Architecture workflow is on NFRs, not FRs.
- Answer e is wrong because the Design model is refined later in the process.

Activity 5: Identifying Characteristics of the Design Workflow

Self-Check – Select the characteristics of the Design workflow mentioned during the lecture.

a. ___ You will use an Activity diagram to verify Use Case diagrams during this workflow.

b. ___ You will analyze the Use Case scenarios to determine additional detail during this workflow.

c. ___ You will create a Solution model during this workflow.

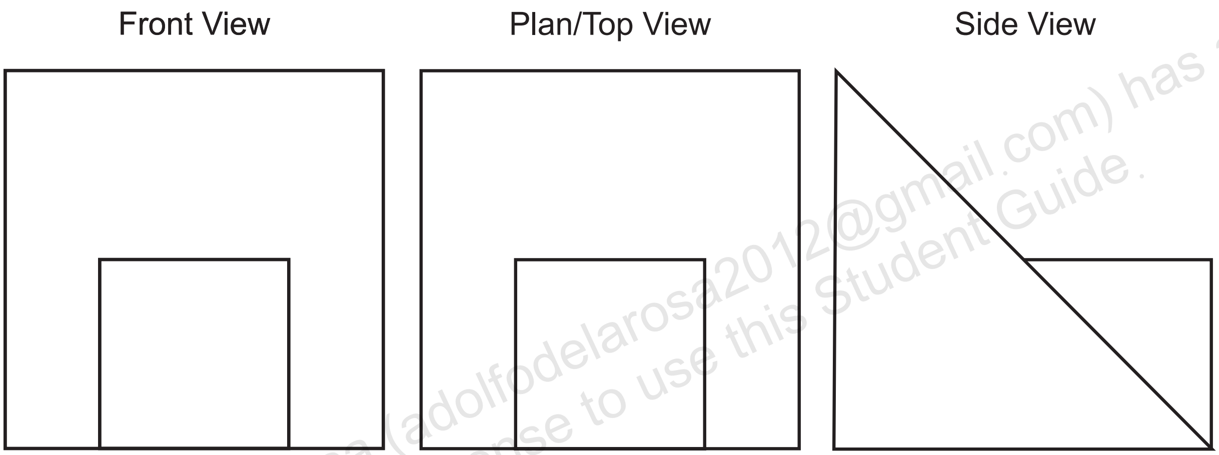
d. ___ You might create a Statechart diagram during this workflow.

e. ___ The purpose of this workflow is to model how the system will support the use cases.

Answers: c, d, e.

- Answer a is wrong because it is done during the Requirements Analysis workflow.
- Answer b is wrong because this is done during the Requirements Analysis workflow.

Activity 6: Exploring the Benefits of Modeling



Front View

Plan/Top View

Side View

Note: There may be alternative solutions.

Activity 7: Identifying the Benefits of Modeling Software

✓ **Self-Check –** Select the benefits of modeling software mentioned during the lecture.

a. ___ Models give you only a starting point for a new system.

b. ___ Models help you only to understand what you have developed.

c. ___ Models help you visualize new or existing systems.

d. ___ Models are a concrete realization of a system.

e. ___ Models help you communicate decisions to project stakeholders.

Answers: c, e.

- Answer a is wrong because models are not intended to become part of the system (such as a prototype might).
- Answer b is wrong because models are meant to help you understand what you will develop (not what has already been developed).
- Answer d is wrong because models are abstract, not concrete.

Activity 8: Identifying Diagram Types

Self-Check – Match the UML diagram with its description of the diagram type.

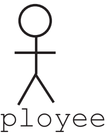
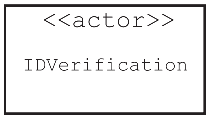
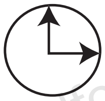
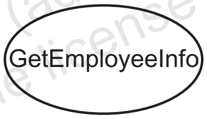
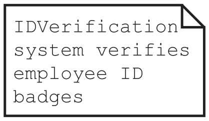
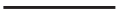
Diagram Name	Definition
Use Case	Represents changes in state or value along with state duration constraints
Class	Represents a flow of tasks that might be performed by either a system or an actor
Object	Represents a collection of components, and shows how these are distributed across one or more hardware nodes
Communication	Represents a flow of task with fragments of detailed object interactions
Sequence	Represents a collection of objects that work together to support some system behavior
Activity	Represents the internal structure of a class in terms of parts
State Machine	Represents a conceptual view of a collection of other modeling elements and diagrams
Component	Represents a collection of physical software components and their interrelationships
Deployment	Represents the set of states that an object might experience along with triggers that transition the object from one state to another
Package	Represents a time-oriented perspective of an object communication
Interaction Overview	Represents a runtime snapshot of software objects and their interrelationships
Timing	Represents extensions to standard diagrams
Composite Structure	Represents the set of high-level behaviors that the system must perform for a given actor

Diagram Name	Definition
Profile	Represents a collection of software classes and their interrelationships

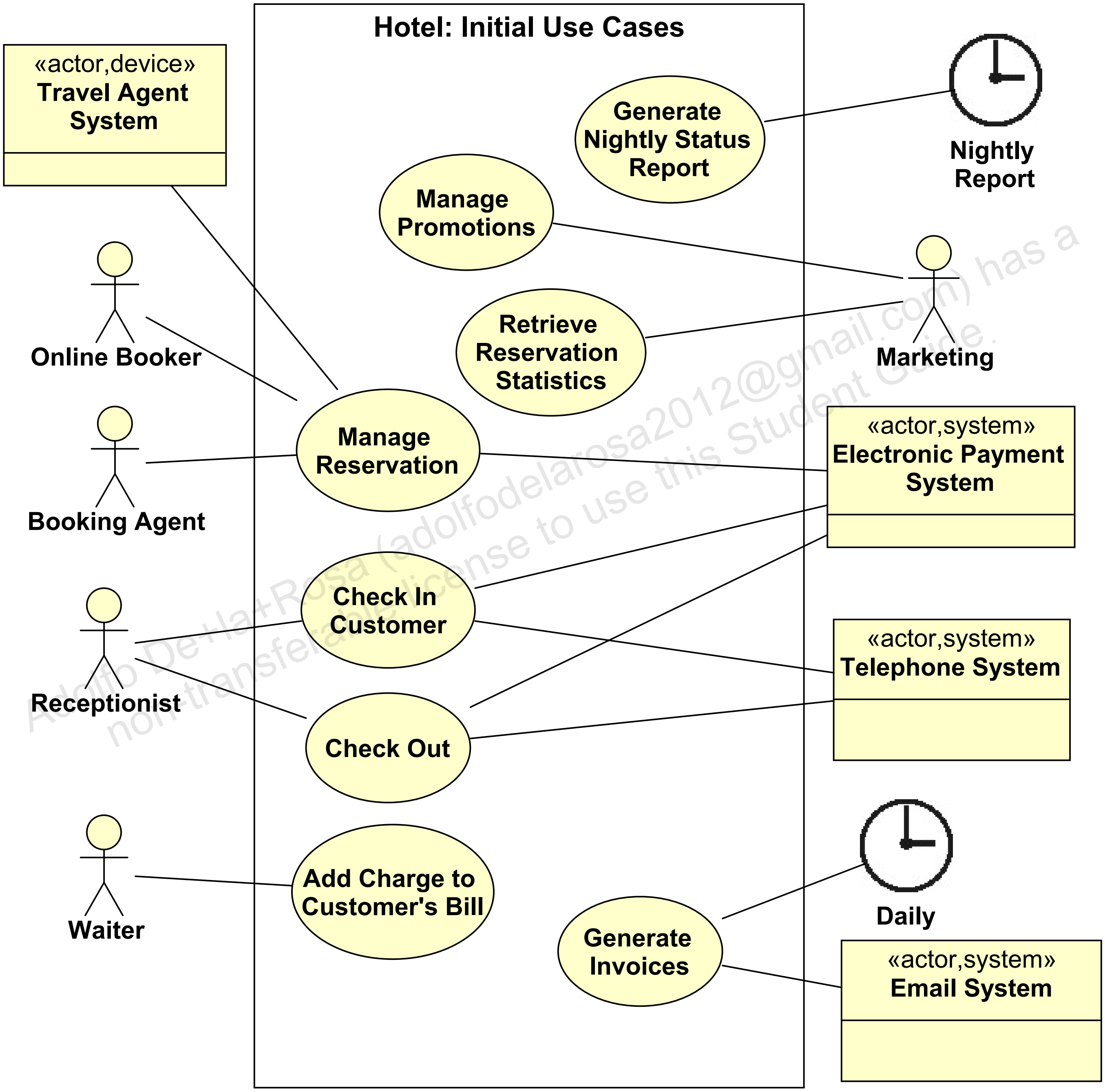
Answers: Timing, Activity, Deployment, Interaction Overview, Communication, Composite Structure, Package, Component, State Machine, Sequence, Object, Profile, Use Case, Class

Activity 1: Identifying Use Case Symbols

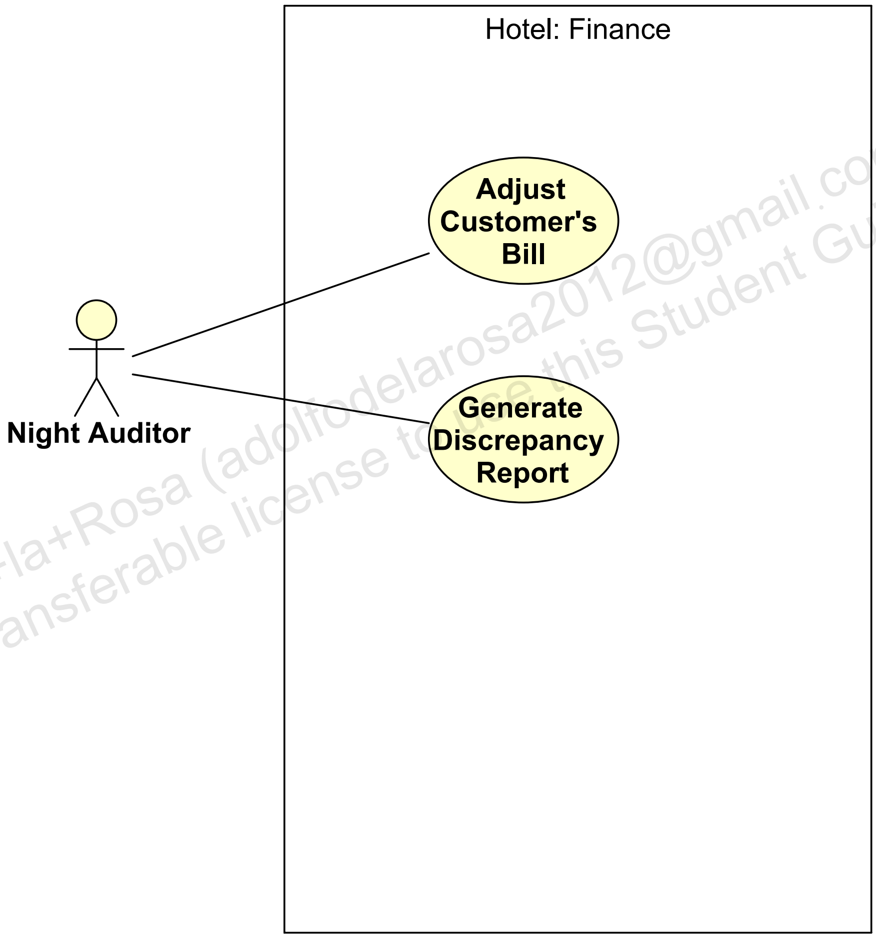
Self-Check – Write the name of each Use Case diagram symbol in the space allotted next to each symbol.

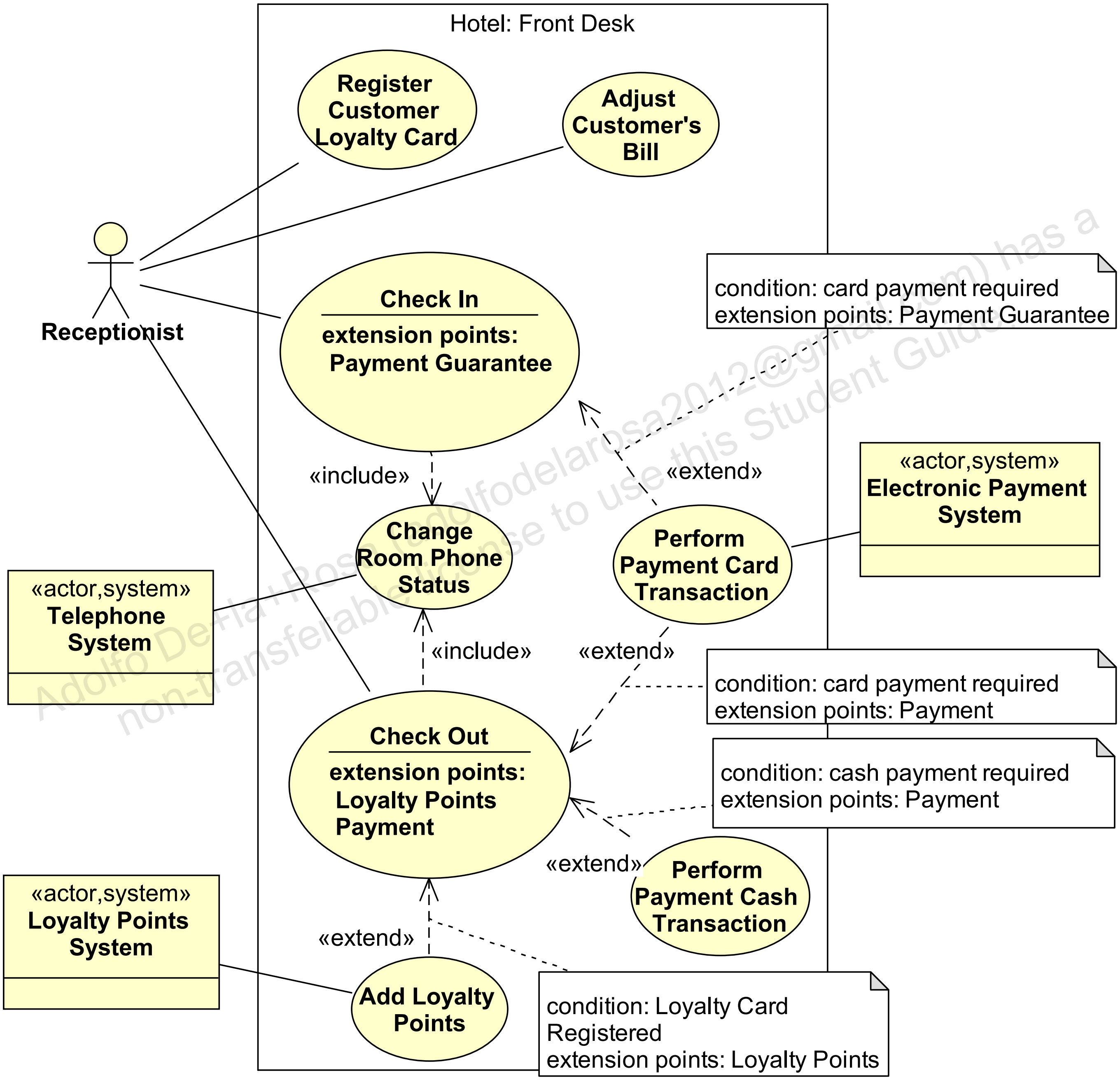
Symbol	Symbol Name
	Human actor node
	System actor node
	Time actor node
	Use case node
	Annotation
	Association

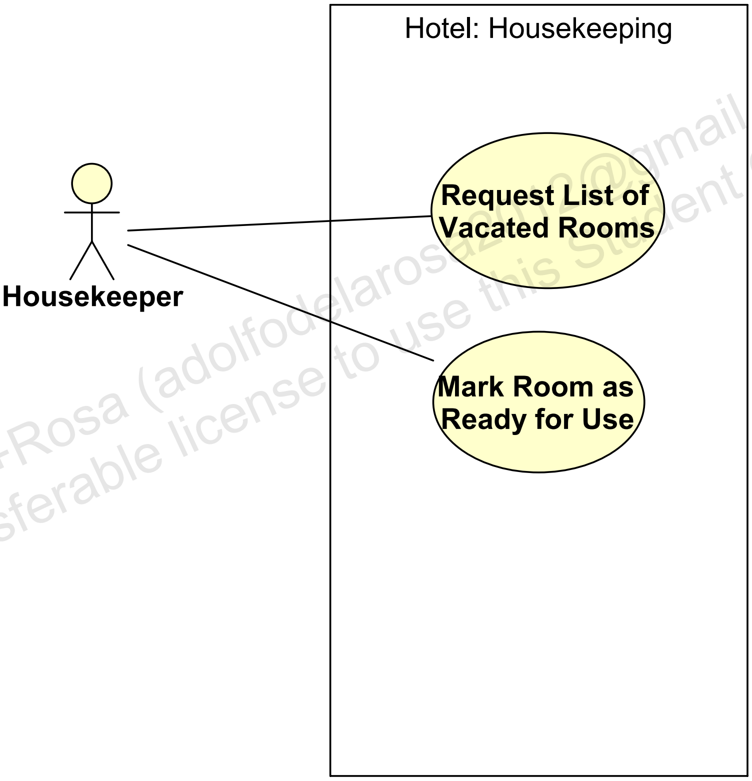
Activity 2: Creating a Use Case Diagram

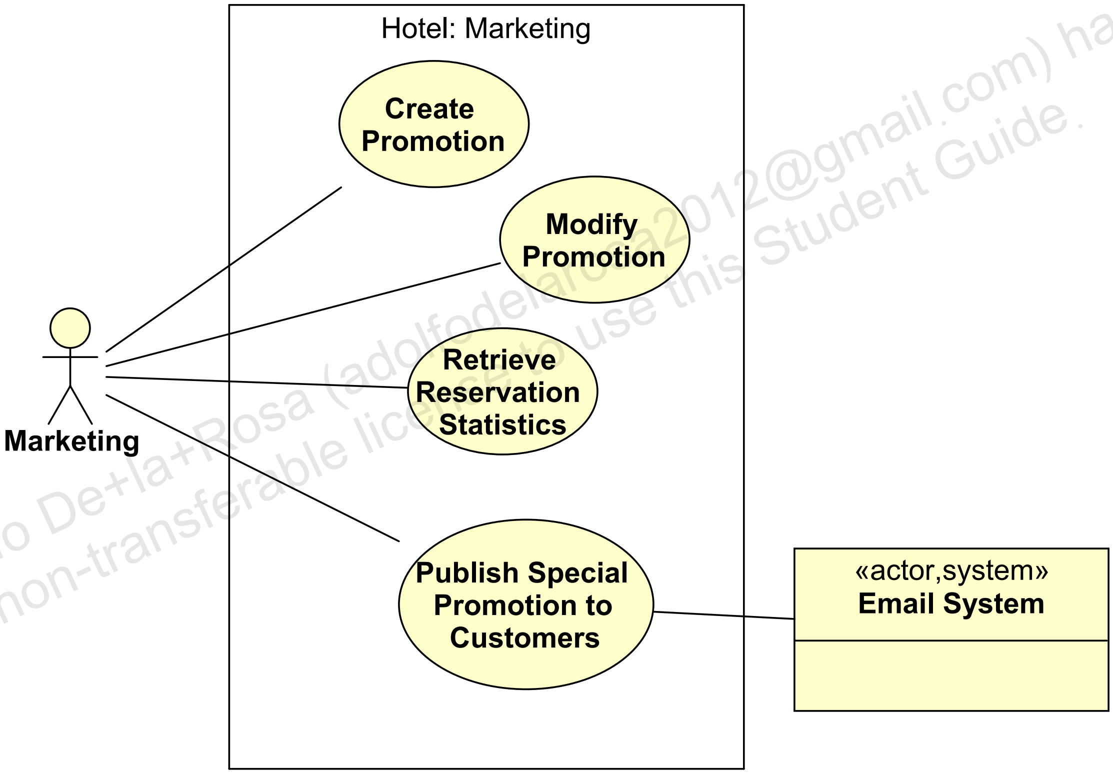


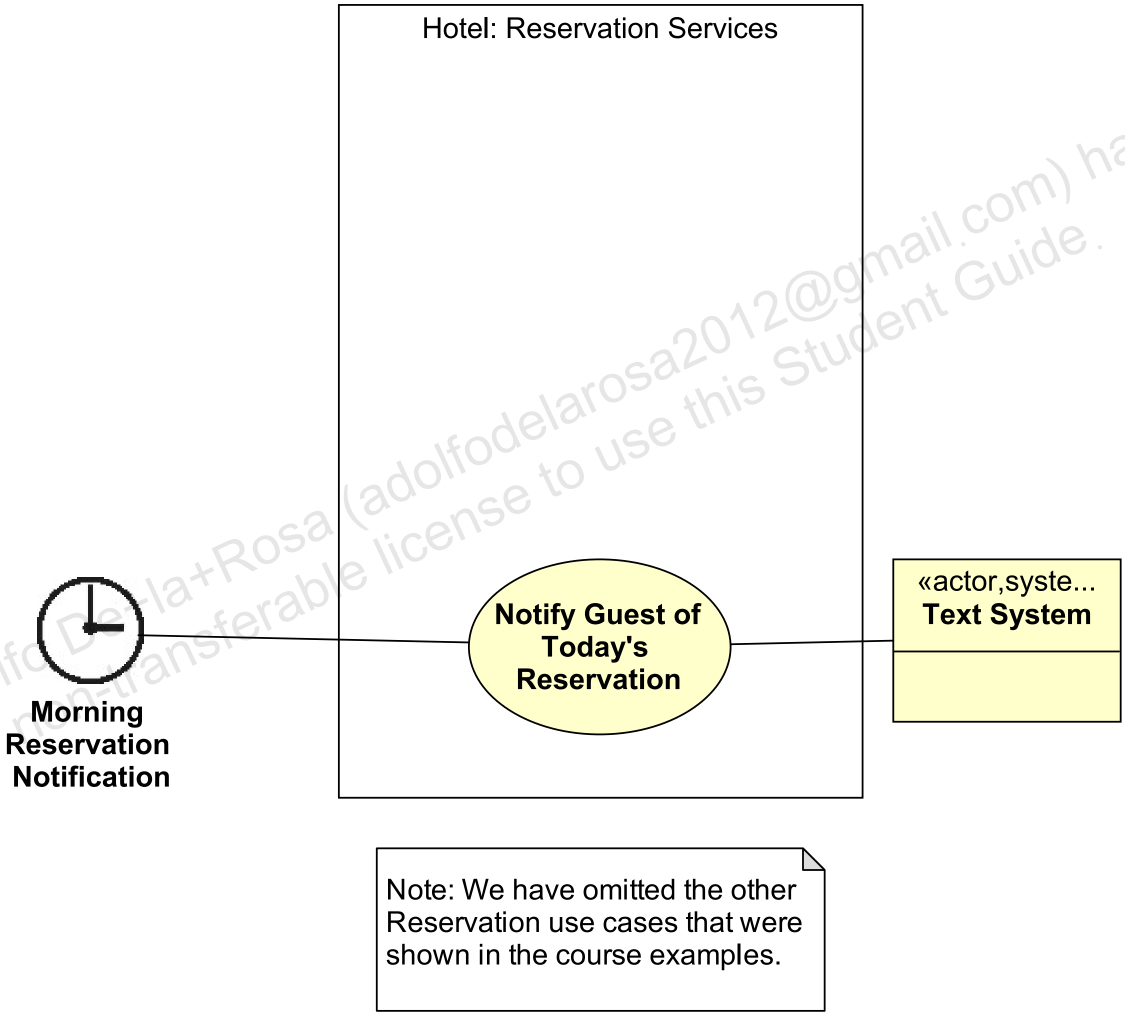
Activity 3: Refining the Use Case Diagram

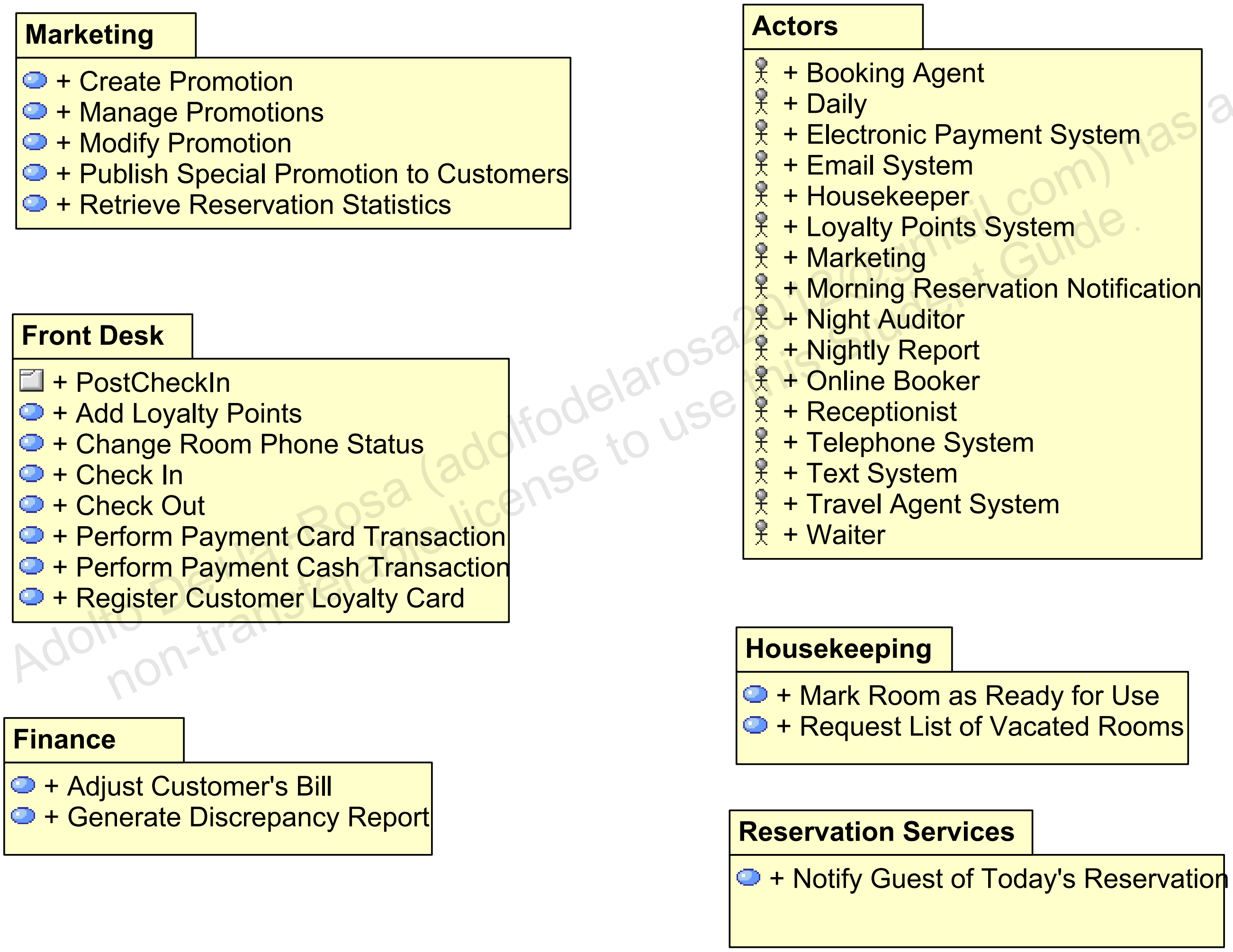



Hotel: Front Desk
Register Customer Loyalty Card
Adjust Customer's Bill
Receptionist
Check In
extension points: Payment Guarantee
condition: card payment required
extension points: Payment Guarantee
«include»
«extend»
«actor,system»
Electronic Payment System
Change Room Phone Status
Perform Payment Card Transaction
«actor,system»
Telephone System
«include»
«extend»
condition: card payment required
extension points: Payment
Check Out
extension points: Loyalty Points Payment
condition: cash payment required
extension points: Payment
«extend»
Perform Payment Cash Transaction
«actor,system»
Loyalty Points System
«extend»
Add Loyalty Points
condition: Loyalty Card Registered
extension points: Loyalty Points


Hotel: Housekeeping
Housekeeper
Request List of Vacated Rooms
Mark Room as Ready for Use


Hotel: Marketing
Create Promotion
Modify Promotion
Retrieve Reservation Statistics
Publish Special Promotion to Customers
Marketing
«actor,system»
Email System


Hotel: Reservation Services
Notify Guest of Today's Reservation
Morning Reservation Notification
«actor,syste...
Text System
Note: We have omitted the other Reservation use cases that were shown in the course examples.



Marketing

+ Create Promotion
+ Manage Promotions
+ Modify Promotion
+ Publish Special Promotion to Customers
+ Retrieve Reservation Statistics

Front Desk

+ PostCheckIn
+ Add Loyalty Points
+ Change Room Phone Status
+ Check In
+ Check Out
+ Perform Payment Card Transaction
+ Perform Payment Cash Transaction
+ Register Customer Loyalty Card

Finance

+ Adjust Customer's Bill
+ Generate Discrepancy Report

Actors

+ Booking Agent
+ Daily
+ Electronic Payment System
+ Email System
+ Housekeeper
+ Loyalty Points System
+ Marketing
+ Morning Reservation Notification
+ Night Auditor
+ Nightly Report
+ Online Booker
+ Receptionist
+ Telephone System
+ Text System
+ Travel Agent System
+ Waiter

Housekeeping

+ Mark Room as Ready for Use
+ Request List of Vacated Rooms

Reservation Services

+ Notify Guest of Today's Reservation

Abstract of Scenarios for the Check In Use Case

The following are some sample scenarios. However, the lab activity required you to only provide summaries.

Primary Scenario 1
The customer arrives at the hotel and is greeted by the receptionist. The customer informs the receptionist that they have a reservation. The receptionist asks for either the reservation number or customer name. The customer provides the reservation number. The receptionist selects the check-in form from the menu. The receptionist enters the reservation number. The system finds the requested reservation. The system displays the customer name, room IDs and room types that were reserved, dates of stay, and the price. The receptionist confirms the details with the customer. The receptionist accepts that the rooms are correct. Payment guarantee is required for further room charges. Customer provides payment guarantee. The system marks the reservation as Checked-In. The system marks all rooms reserved as being occupied. The system creates the room keys. The system enables the telephone in all the reserved rooms. The system creates the bill. The system adds the quoted charge for the rooms to the bill as a chargeable item of quoted room charge type. The system marks the bill as allowing additional charges.

Primary Scenario 2
The customer arrives at the hotel and is greeted by the receptionist. The customer informs the receptionist that they have a reservation. The receptionist asks for either the reservation number or customer name. The customer provides the reservation number. The receptionist selects the check-in form from the menu. The receptionist enters the reservation number. The system finds the requested reservation. The system displays the customer name, room IDs and room types that were reserved, dates of stay, and the price. The receptionist confirms the details with the customer. The receptionist accepts that the rooms are correct. Payment guarantee is required for further room charges. Customer does not provide payment guarantee. The system marks the reservation as Checked-In. The system marks all rooms reserved as being occupied. The system creates the room keys. The system enables the telephone in all the reserved rooms. The system creates the bill. The system adds the quoted charge for the rooms to the bill as a chargeable item of quoted room charge type. The system marks the bill as not allowing additional charges.

Note: In the above scenario and in all of the remaining scenarios, the text in blue indicates the difference in the particular scenario with respect to Primary Scenario 1.

Primary Scenario 3
The customer arrives at the hotel and is greeted by the receptionist. The customer informs the receptionist that they have a reservation. The receptionist asks for either the reservation number or customer name. The customer provides the reservation number. The receptionist selects the check-in form from the menu. The receptionist enters the reservation number. The system finds the requested reservation. The system displays the customer name, room IDs and room types that were reserved, dates of stay, and the price. The receptionist confirms the details with the customer. The receptionist accepts that the rooms are correct. No further payment guarantee is required for further room charges. The system marks the reservation as

Checked-In. The system marks all rooms reserved as being occupied. The system creates the room keys. The system enables the telephone in all the reserved rooms. The system creates the bill. The system adds the quoted charge for the rooms to the bill as a chargeable item of quoted room charge type. The system marks the bill as allowing additional charges.

Primary Scenario 4

The customer arrives at the hotel and is greeted by the receptionist. The customer informs the receptionist that they have a reservation. The receptionist asks for either the reservation number or customer name. The customer provides the name. The receptionist selects the check-in form from the menu. The receptionist enters the customer's name. The system finds only one matching reservation. The system displays the customer name, room IDs and room types that were reserved, dates of stay, and the price. The receptionist confirms the details with the customer. The receptionist accepts that the rooms are correct. Payment guarantee is required for further room charges. Customer provides payment guarantee. The system marks the reservation as Checked-In. The system marks all rooms reserved as being occupied. The system creates the room keys. The system enables the telephone in all the reserved rooms. The system creates the bill. The system adds the quoted charge for the rooms to the bill as a chargeable item of quoted room charge type. The system marks the bill as allowing additional charges.

Primary Scenario 5

The customer arrives at the hotel and is greeted by the receptionist. The customer informs the receptionist that they have a reservation. The receptionist asks for either the reservation number or customer name. The customer provides the name. The receptionist selects the check-in form from the menu. The receptionist enters the customer's name. The system finds three matching reservations. The system displays a summary of each reservation. The customer confirms which reservation is theirs. The receptionist selects the valid reservation. The system displays the customer name, room IDs and room types that were reserved, dates of stay, and the price. The receptionist confirms the details with the customer. The receptionist accepts that the rooms are correct. Payment guarantee is required for further room charges. Customer provides payment guarantee. The system marks the reservation as Checked-In. The system marks all rooms reserved as being occupied. The system creates the room keys. The system enables the telephone in all the reserved rooms. The system creates the bill. The system adds the quoted charge for the rooms to the bill as a chargeable item of quoted room charge type. The system marks the bill as allowing additional charges.

Secondary Scenario 1

The customer arrives at the hotel and is greeted by the receptionist. The customer informs the receptionist that they have a reservation. The receptionist asks for either the reservation number or customer name. The customer provides the reservation number. The receptionist selects the check-in form from the menu. The receptionist enters the reservation number. The system finds the requested reservation. The system displays the customer name, room IDs and room types that were reserved, dates of stay, and the price. The receptionist confirms the details with the customer. The receptionist rejects that the rooms are correct. The receptionist restarts the check-in process.

Secondary Scenario 2
The customer arrives at the hotel and is greeted by the receptionist. The customer informs the receptionist that they have a reservation. The receptionist asks for either the reservation number or customer name. The customer provides the name. The receptionist selects the check-in form from the menu. The receptionist enters the customer’s name. The system finds three matching reservations. The system displays a summary of each reservation. The customer confirms which reservation is theirs. The receptionist cannot find a valid reservation. The receptionist restarts the check-in process.

Use Case Form

Use Case Name	Check In
Description	The Customer arrives at the hotel and provides information to find an existing reservation. The customer is given the option to upgrade or change rooms. A further payment guarantee may be required. Upon successful check-in, an electronic room key is cut, the room phone is enabled, the new bill is created, and the quoted price is charged to the customer's bill.
Actors	Primary: Receptionist Secondary: None Note: Primary actors are proxies for the Customer
Priority	Must have Note: Essential to this system
Risk	Medium
Pre-conditions & Assumptions	At least one room exists in the hotel Primary Actor can be identified
Extension Points	Rooms Upgrade Change Assigned Rooms Payment guarantee
Extends	None
Trigger	A Customer wishes to check into the hotel using an existing reservation.
Main Flow of Events	1: Use Case starts when Customer requests to check into the hotel 2: Receptionist enters reservation number [A1] 2.1: System searches for matching reservation [A2] [A3] 2.2: System notifies Receptionist of the reservation details 2.3: Extension Point(Change Assigned Rooms) [A7] 2.4: Receptionist accepts that rooms are correct [A5] 3: Extension Point(Payment Guarantee) 3.1: System marks reservation status as being checked-in 3.2: System applies BR21 and marks assigned rooms as being occupied 3.3: Include (Create Electronic Keys) 3.4: Include (Change Room Telephone Status) [A8] 3.5: System creates bill for reservation 3.6: System applies BR22 to add quoted charge for the rooms to the bill as a chargeable item of quoted room charge type 3.7: System marks bill as allowing additional chargeable items [A6] 4: System indicates that check-in is complete
Alternate Flow of Events (A)	A1: Receptionist enters customer name, go to step 2.1 A2: No matching reservation found, notify Receptionist, go to step 2 A3: System finds more than one matching reservation, systems displays summary of each reservation found. Receptionist selects

	the valid reservation, go to step 2.2 [A4] A4: Receptionist enters that none of the reservations were valid, go to step 2 A5: Receptionist doesn’t accept that rooms are correct, go to step 2.3 A6: No payment guarantee for additional charges, marks bill as not allowing additional chargeable items, go to step 4. A7: Rooms changed, go to 2.2 A8: No Payment Guarantee for additional charges, do not change phone status to enabled, go to step 3.5 Steps 1 through 2.4: Actor may cancel the use case, use case ends
Post-conditions	Reserved rooms are marked as being occupied Reservation is marked as being check-in Bill is marked with payment status The initial room charge is added to the bill
Business Rules (BR)	BR21: Reserved rooms must be marked as being occupied at check-in BR22: Quoted price for the stay must be added to the bill on check-in
Non-Functional Requirements	NFR32 (Simultaneous Users) NFR33 (Duration of Use Case) NFR6 (System Availability)
Notes	Upgrade rooms can be accomplished using change rooms

Abstract of Supplementary Specification Document

NFR32: The System must support 6 simultaneous check-in operations per property.
NFR33: The Check In use case must take no longer than 5 minutes to complete.
NFR6: The System must be available “7 by 24 by 356”. However, the applications can be shut down for maintenance once a week for one hour. This maintenance activity should be scheduled between 3 a.m. and 6 a.m.

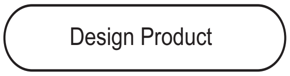
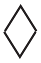
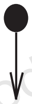
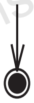
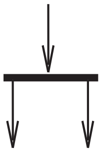
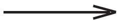
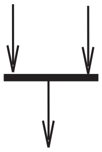
Activity 3: Writing Glossary of Terms

Abstract of Glossary of Terms

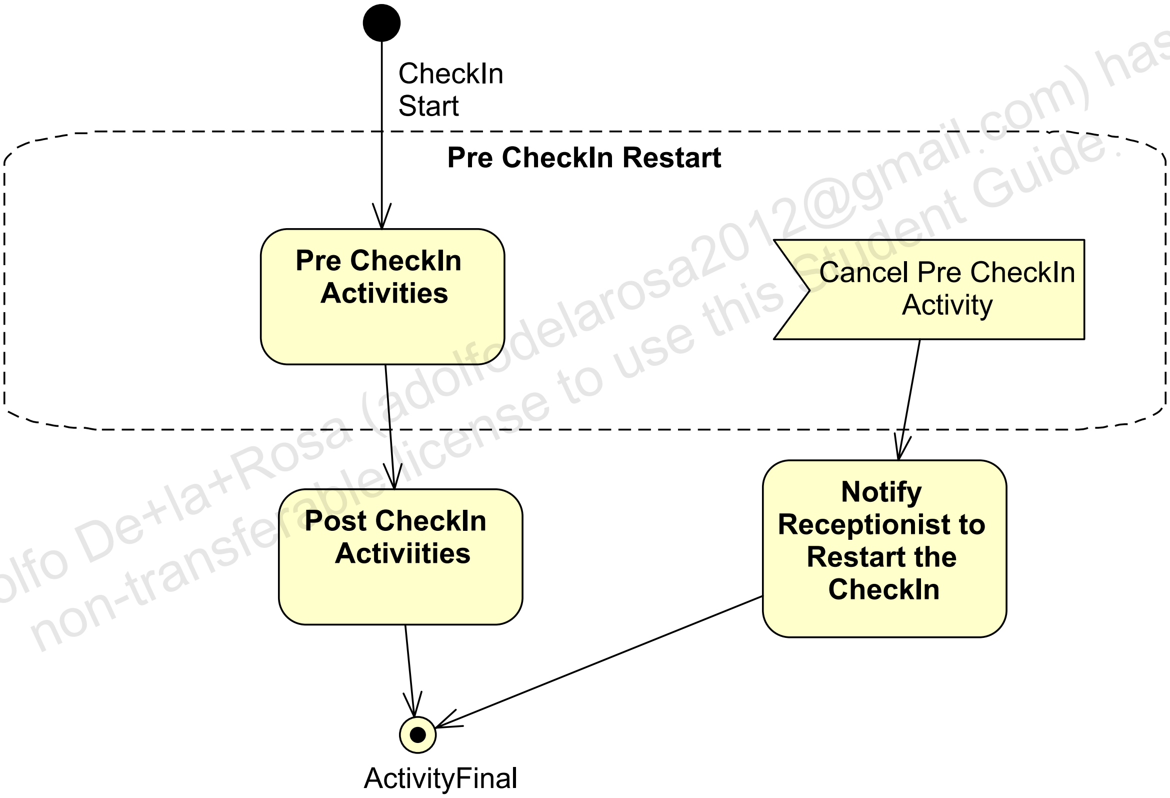
Term	Definition
Reservation Number	A unique numeric identifier assigned to a reservation.
Reservation Status	Marks the status of a reservation. The status includes New, Held, Confirmed, Checked-In, and Checked-Out.
Bill	Contains a list of chargeable items added to the reservation's bill.
Chargeable Item	The amount, date, and type of chargeable item.
Chargeable Item Type	The type of charge made to the bill—for example, beverage, food, room, or newspaper.
Quote Room Charge	The amount quoted for the chosen room types for the duration of the stay.
Room Status	Records the status of a room—for example, occupied, available, or unserviceable.

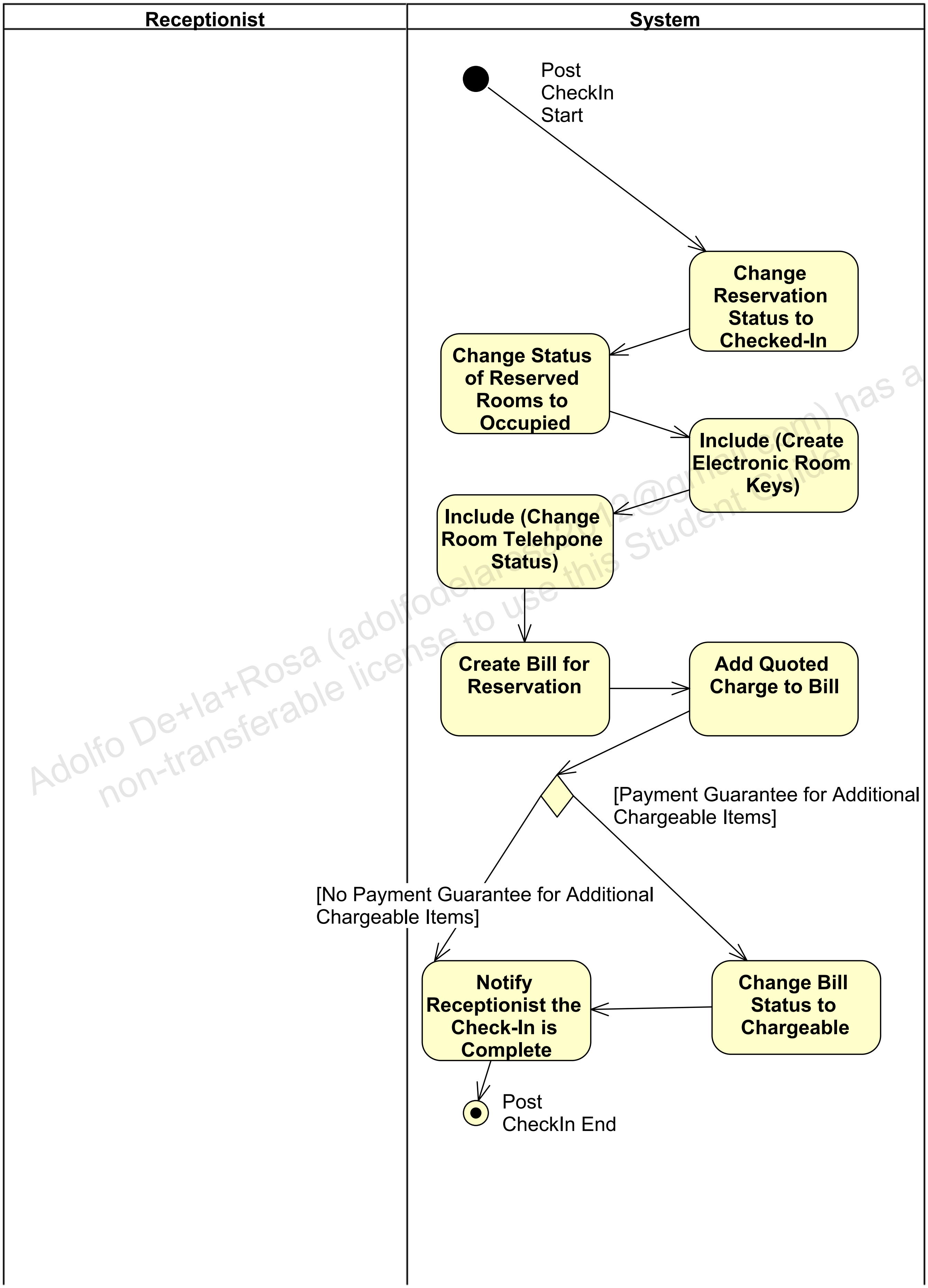
Actrivity1: Identifying Activity Diagram Symbols

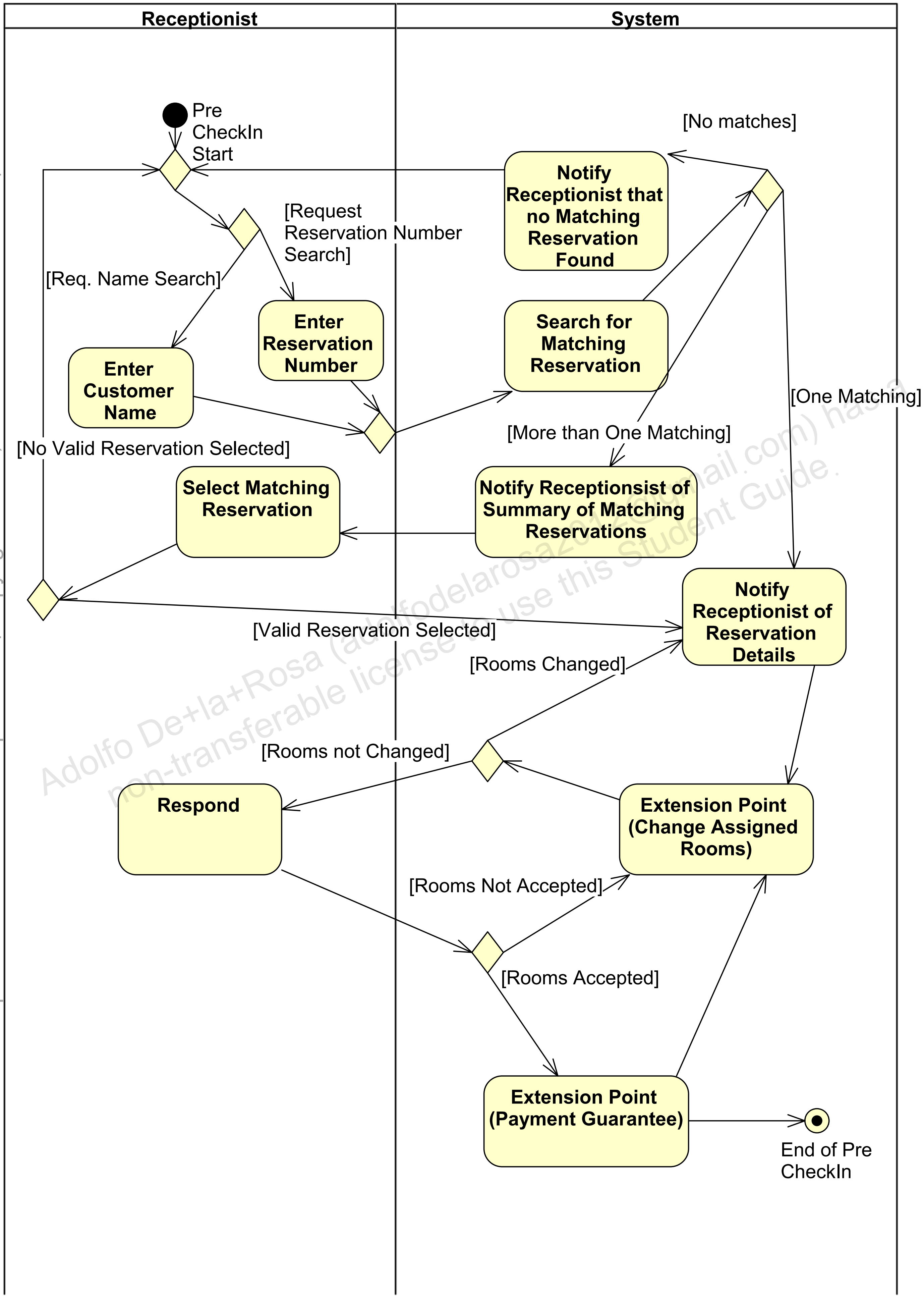
Self-Check – Write the name of each Activity diagram symbol in the space allotted next to each symbol, in the following table.

Symbol	Symbol Name
	Activity or Action
	Decision or Merge
	Activity Inititial
	Activity Final
	Fork
	Activity Flow
	Join

Activity 2: Creating an Activity Diagram




Receptionist
System
Post CheckIn Start
Change Reservation Status to Checked-In
Change Status of Reserved Rooms to Occupied
Include (Create Electronic Room Keys)
Include (Change Room Telehpone Status)
Create Bill for Reservation
Add Quoted Charge to Bill
[Payment Guarantee for Additional Chargeable Items]
[No Payment Guarantee for Additional Chargeable Items]
Notify Receptionist the Check-In is Complete
Change Bill Status to Chargeable
Post CheckIn End


Receptionist
System
Pre CheckIn Start
[No matches]
Notify Receptionist that no Matching Reservation Found
[Request Reservation Number Search]
[Req. Name Search]
Enter Reservation Number
Search for Matching Reservation
Enter Customer Name
[One Matching]
[More than One Matching]
[No Valid Reservation Selected]
Select Matching Reservation
Notify Receptionist of Summary of Matching Reservations
Notify Receptionist of Reservation Details
[Valid Reservation Selected]
[Rooms Changed]
[Rooms not Changed]
Respond
Extension Point (Change Assigned Rooms)
[Rooms Not Accepted]
[Rooms Accepted]
Extension Point (Payment Guarantee)
End of Pre CheckIn

Activity 1: Finding Candidate Key Abstractions

Candidate Classes Form

Candidate Class/Object	Candidate Class/Object Eliminated for the Following Reason	Selected Class Name
Customer (actor)		
Hotel		
Reservation		
Room		
Payment Guarantee		
Room Key		
Room Phone		
Bill		
Chargeable Item		
Receptionist		
System		
Rooms Upgrade		
Assigned Rooms		
Reservation Number		
Customer Name		
Room Id		
Room Type		
Dates of Stay		
Price		
Checked-In Status		
Room Occupied Status		
Quoted Charge		

Additional Chargeable Items Allowed Status		
Initial Room Charge		
Chargeable Item Type		
Customer		
Basic Rate		
Promotion		

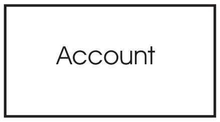
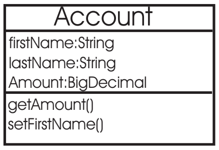
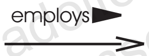
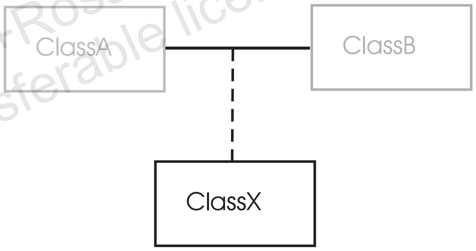
Candidate Classes Form

Candidate Class/Object	Candidate Class/Object Eliminated for the Following Reason	Selected Class Name
Customer (actor)	Actor	
Hotel		Property
Reservation		Reservation
Room		Room
Payment Guarantee		PaymentGuarantee
Room Key	External	
Room Phone	External	
Bill		Bill
Chargeable Item		ChargeableItem
Receptionist	Actor	
System	The system	
Rooms Upgrade	Reassignment of rooms	
Assigned Rooms	Assignment of Room	
Reservation Number	Attribute of Reservation	
Customer Name	Attribute of Customer	
Room Id	Attribute of Room	
Room Type		RoomType
Dates of Stay	Synonym for Arrival Date and Departure Date	
Price	Synonym for Quoted Price	
Checked-In Status	Attribute of Reservation	
Room Occupied Status	Attribute of Room	
Quoted Charge	Attribute of Reservation	

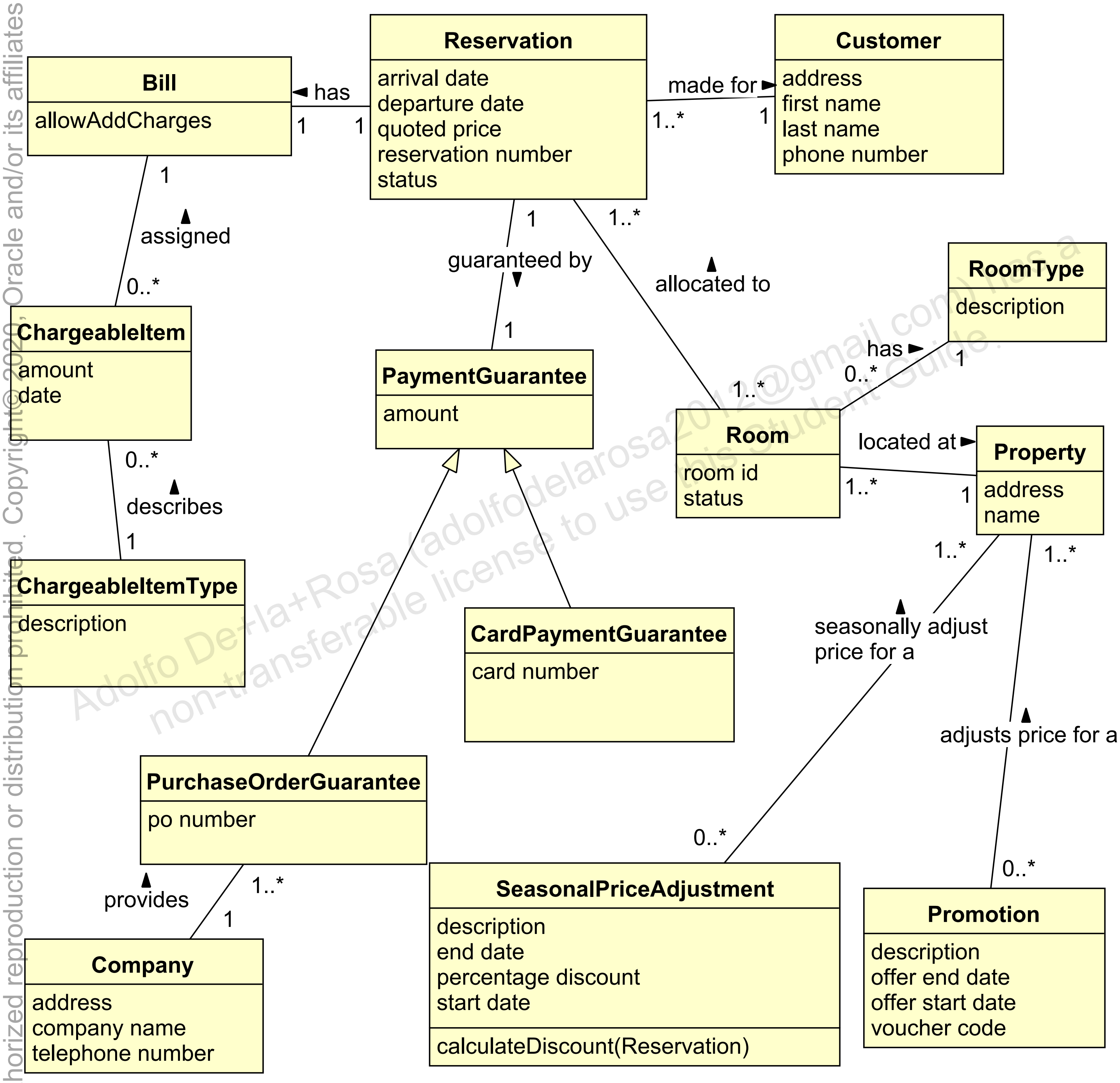
Additional Chargeable Items Allowed Status	Attribute of Bill	
Initial Room Charge	Synonym for Chargeable Item	
Chargeable Item Type		ChargeableItemType
Customer		Customer
Basic Rate	Attribute of Room Type	
Promotion		Promotion

Activity 1: Identifying Class Diagram Elements

Self-Check – Write the name of each Class diagram symbol in the space allotted next to each symbol.

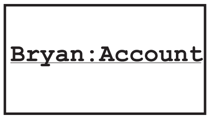
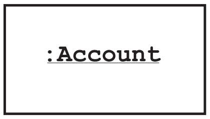
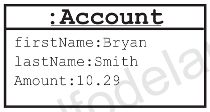
Symbol	Symbol Name
	Class node
 Account firstName:String lastName:String Amount:BigDecimal getAmount() setFirstName()	Class node with members
1..*	Mulitiplicity label
	Unidirectional association
	Association class

Activity 2: Domain Model Class Diagram



Activity 3: Identifying Object Diagram Symbols

Self-Check – Write the name of each Object diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	Named object node
	Unnamed object node
 :Account firstName:Bryan lastName:Smith Amount:10.29	Object node with attribute values

Self-Check – Select the statement or statements about UML Object diagrams that are TRUE.

a. ___ Object diagrams can be validated using Class diagrams.

b. ___ Object diagrams show runtime links.

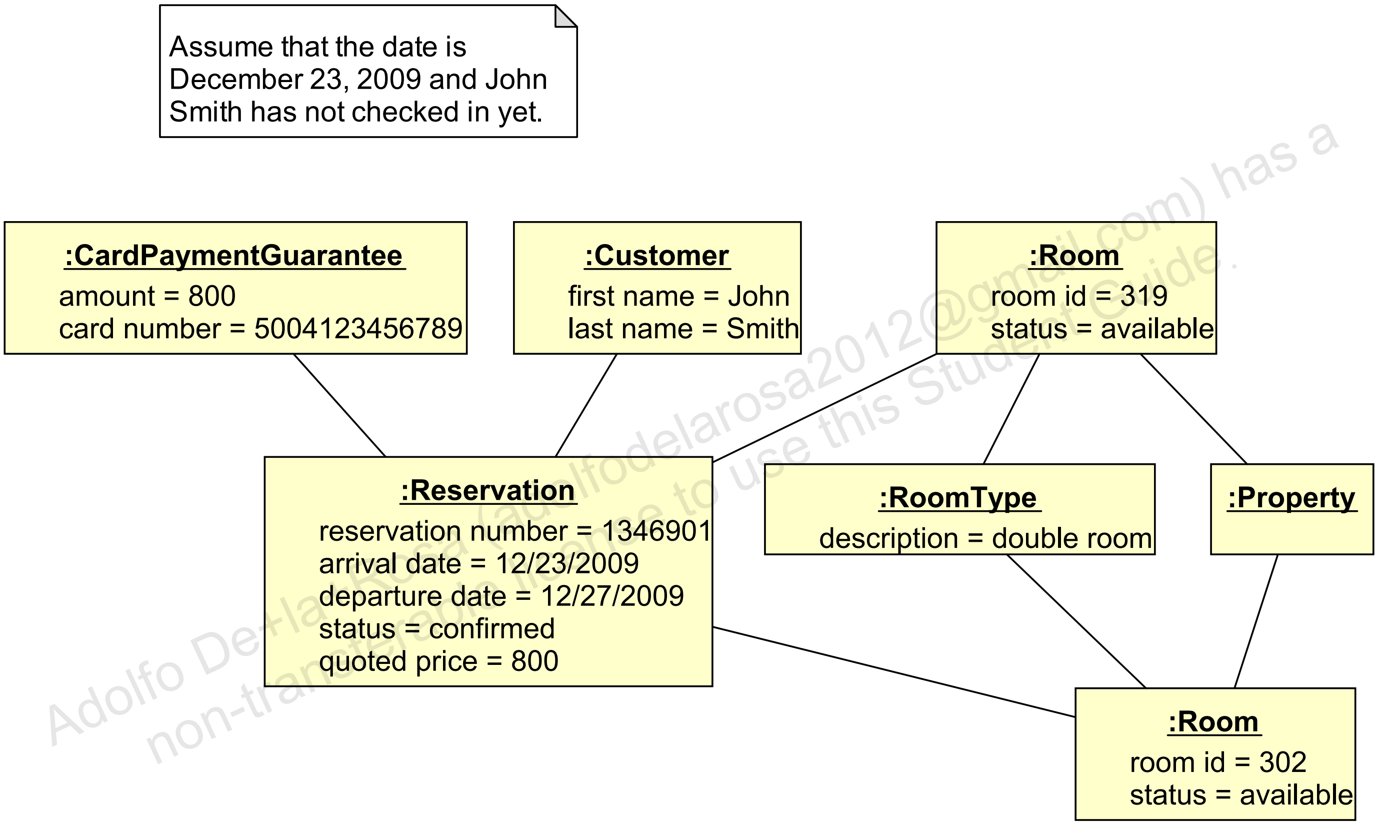
c. ___ Object symbols can have three compartments: name, attribute, and an operations compartment.

d. ___ Object diagrams often show object state.

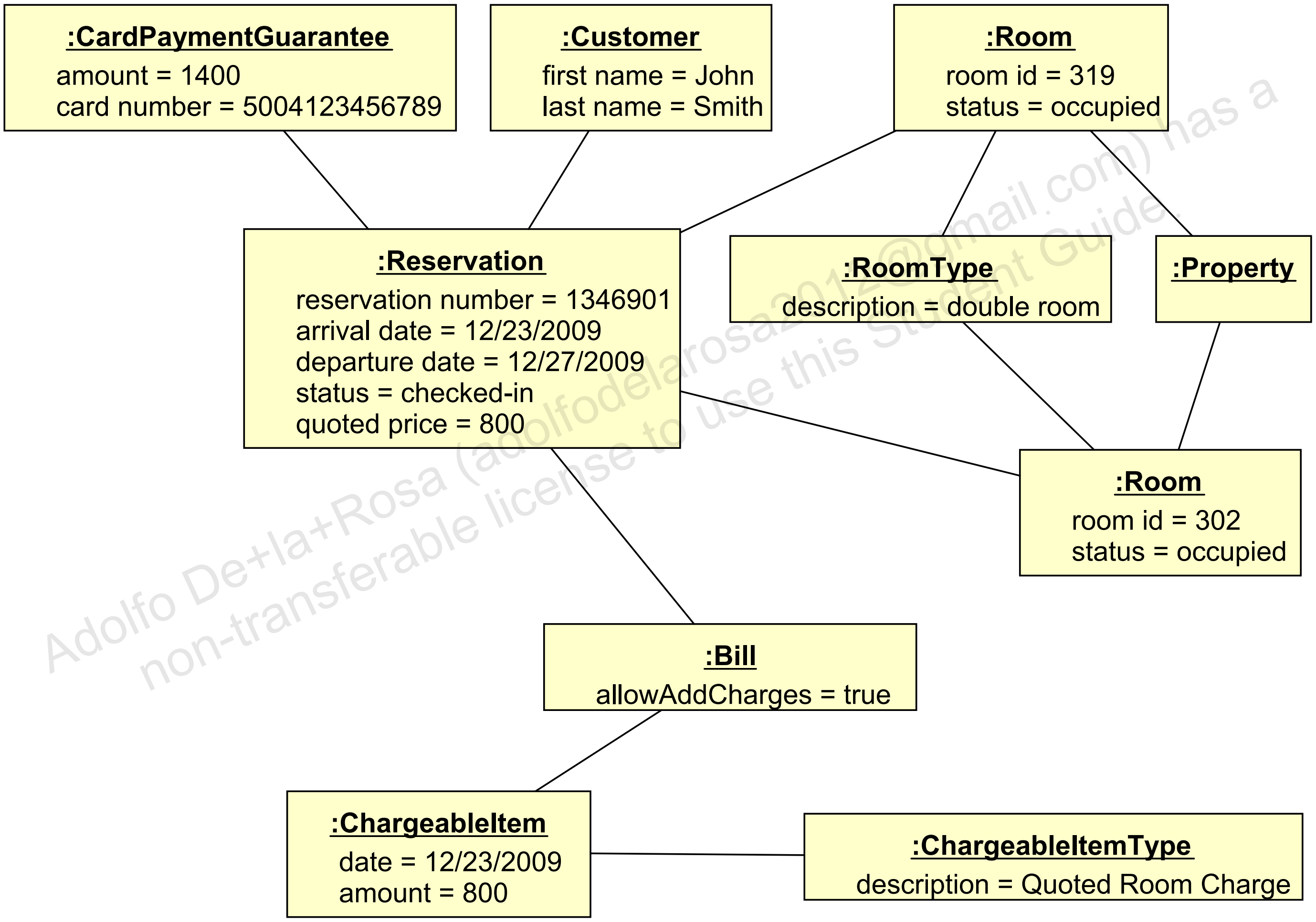
e. ___ An Object diagram is an instance of a class diagram.

Answers: b, d, e.

Activity 4: Validating a Class Diagram



Assume that the date is December 23, 2009 and John Smith has checked in.
He has extended his payment guarantee amount to cover additional charges.



:CardPaymentGuarantee
amount = 1400
card number = 5004123456789

:Customer
first name = John
last name = Smith

:Room
room id = 319
status = occupied

:Reservation
reservation number = 1346901
arrival date = 12/23/2009
departure date = 12/27/2009
status = checked-in
quoted price = 800

:RoomType
description = double room

:Property

:Room
room id = 302
status = occupied

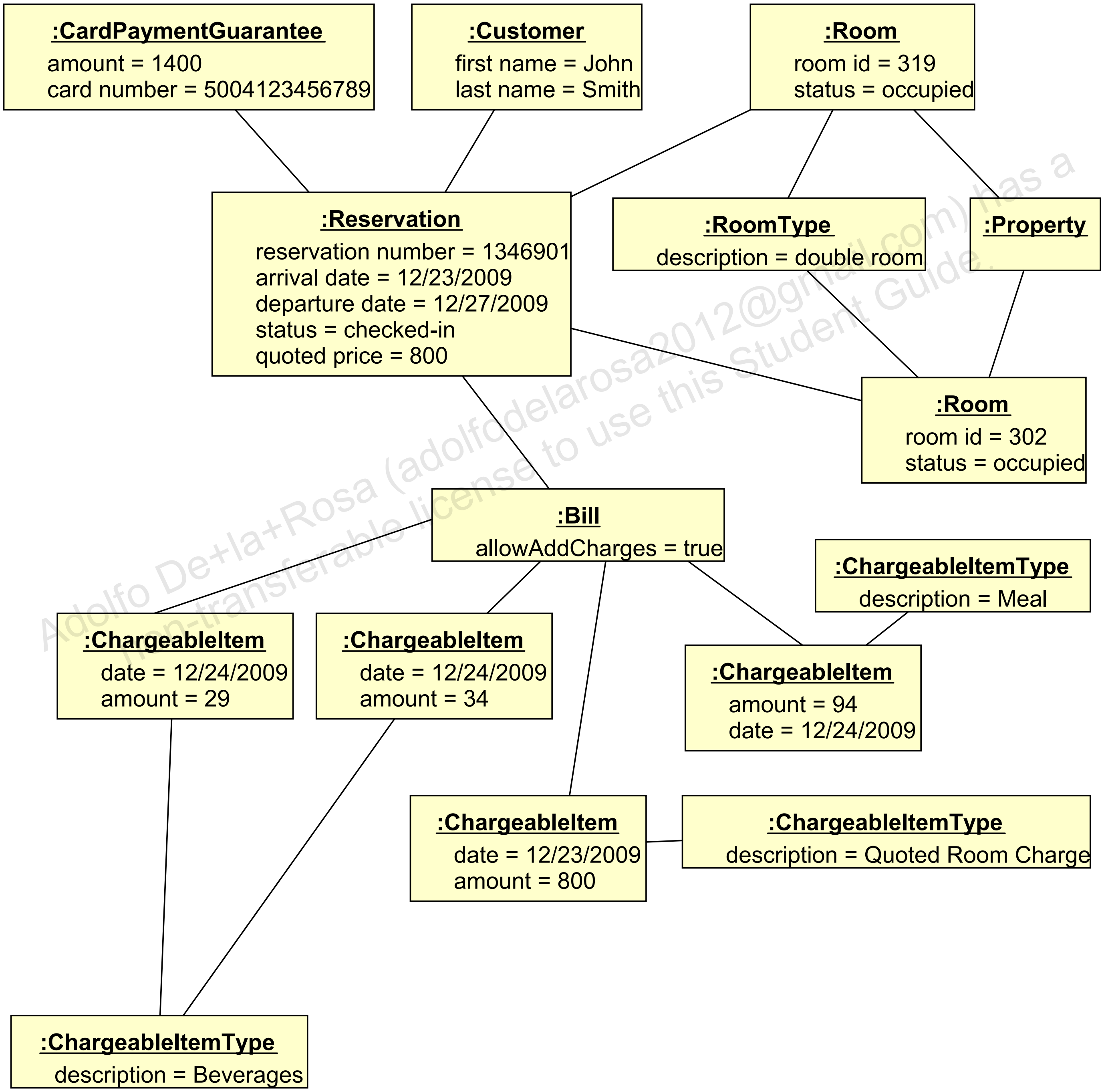
:Bill
allowAddCharges = true

:ChargeableItem
date = 12/23/2009
amount = 800

:ChargeableItemType
description = Quoted Room Charge

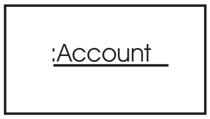
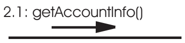
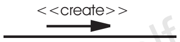
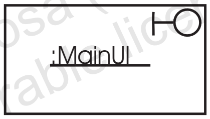
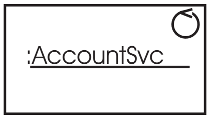
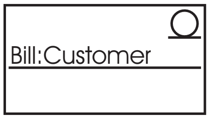
Note: During this modeling review with the client, you might discover that a reservation can have a second payment guarantee. In that case, the Domain class diagram as well as the Use Case forms will need to be modified.

This is not part of the lab, but may be of interest. Assume date is 24 Dec 2009, and John Smith and his party have a meal in the restaurant and purchase beverages twice.

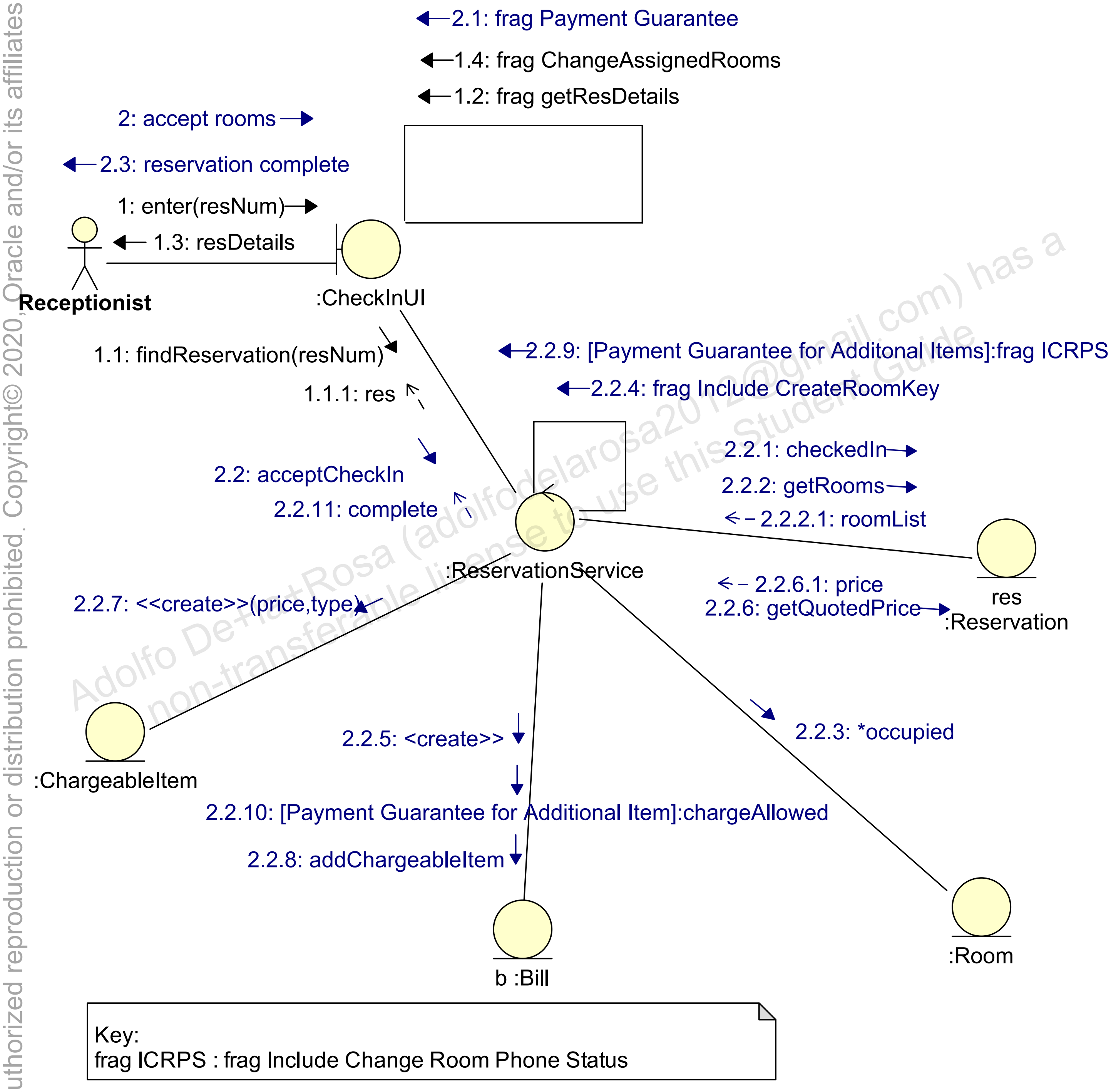


Activity 1: Identifying Collaboration Diagram Elements

Self-Check – Write the name of each Collaboration diagram symbol in the space next to each symbol.

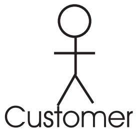
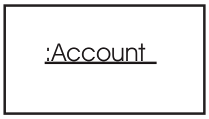
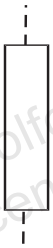
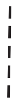
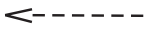
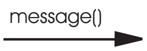
Symbol	Symbol Name
	Object node
	Message link
	Create object message
	Boundary component
	Service component
	Entity component

Activity 2: Creating a Communication Diagram

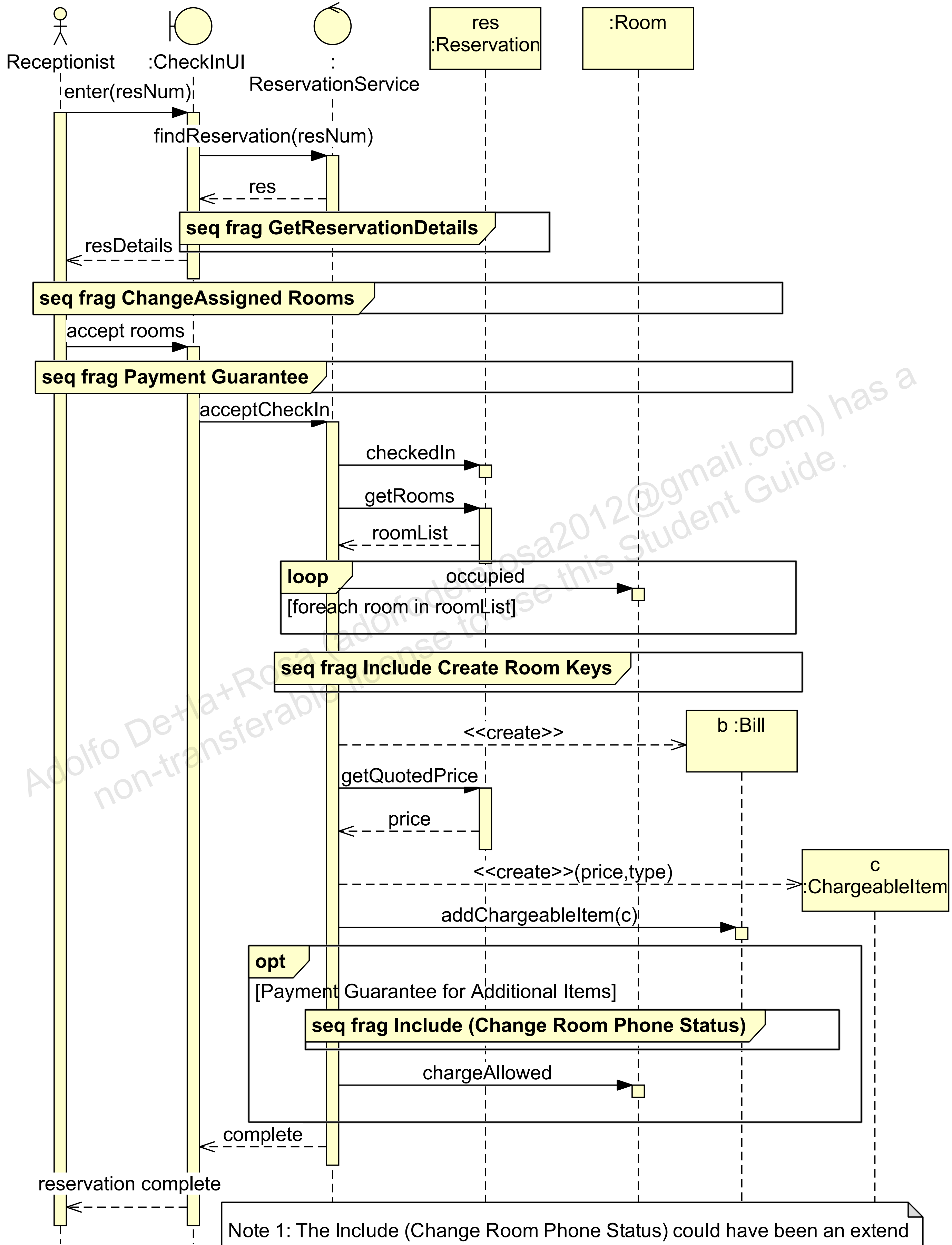

2.1: frag Payment Guarantee
1.4: frag ChangeAssignedRooms
1.2: frag getResDetails
2: accept rooms
2.3: reservation complete
1: enter(resNum)
1.3: resDetails
Receptionist
:CheckInUI
1.1: findReservation(resNum)
1.1.1: res
2.2: acceptCheckIn
2.2.11: complete
2.2.9: [Payment Guarantee for Additonal Items]:frag ICRPS
2.2.4: frag Include CreateRoomKey
2.2.1: checkedIn
2.2.2: getRooms
2.2.2.1: roomList
:ReservationService
res
:Reservation
2.2.6.1: price
2.2.6: getQuotedPrice
2.2.7: <<create>>(price,type)
:ChargeableItem
2.2.5: <create>>
2.2.10: [Payment Guarantee for Additional Item]:chargeAllowed
2.2.8: addChargeableItem
2.2.3: *occupied
b :Bill
:Room
Key:
frag ICRPS : frag Include Change Room Phone Status

Activity 3: Identifying Sequence Diagram Elements

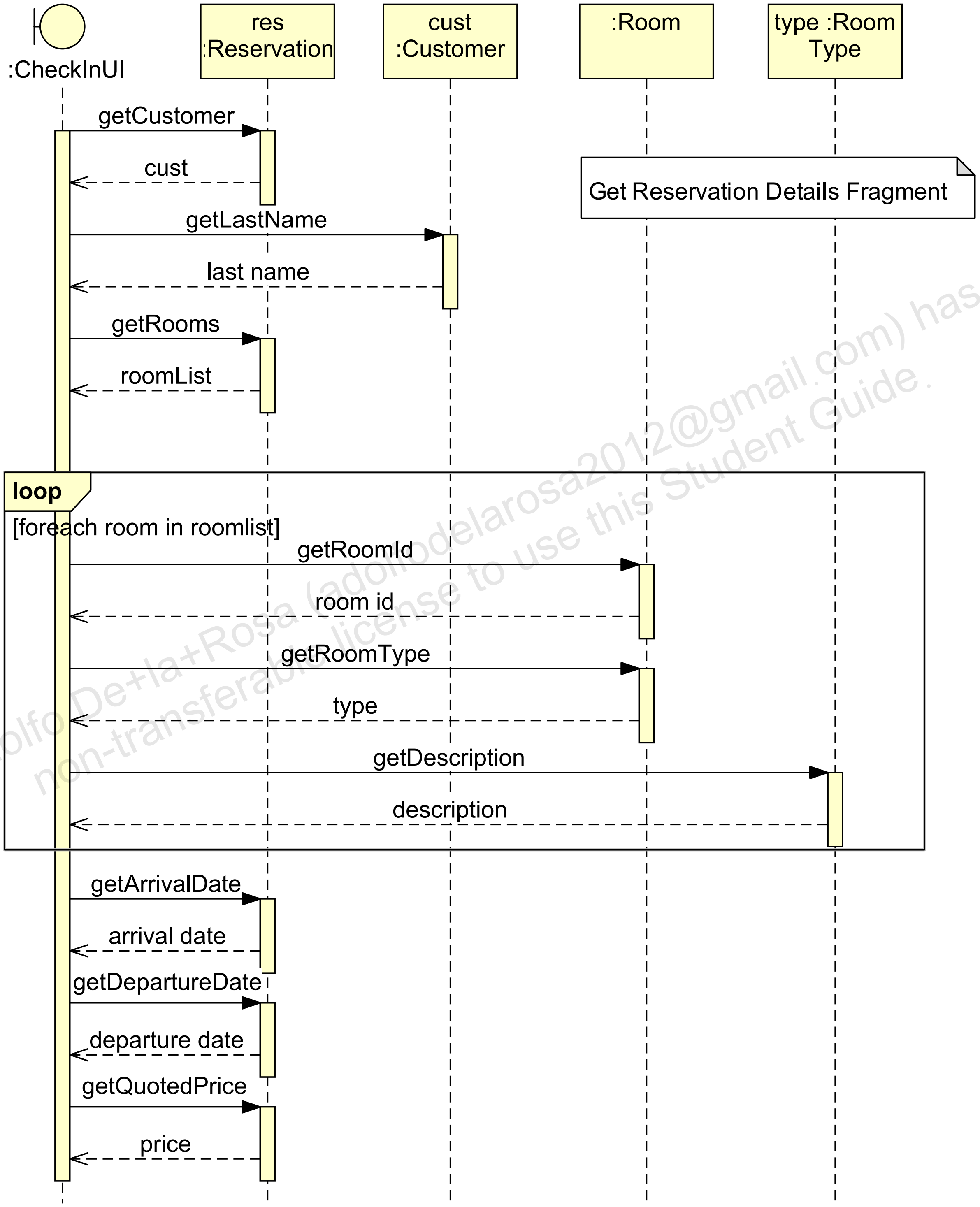
Self-Check – Write the name of each Sequence diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	Actor node
	Object node
	Activation bar
	Lifeline
	Return arrow
	Message arrow

Activity 4: Creating a Sequence Diagram

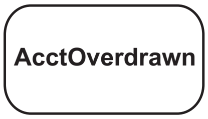
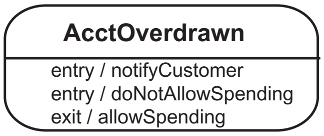
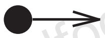
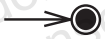
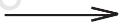


Note 1: The Include (Change Room Phone Status) could have been an extend stereotype due to its optional nature.
Note 2: The Include (Change Room Phone Status) was moved from being earlier in the sequence so it could share the optional fragment.

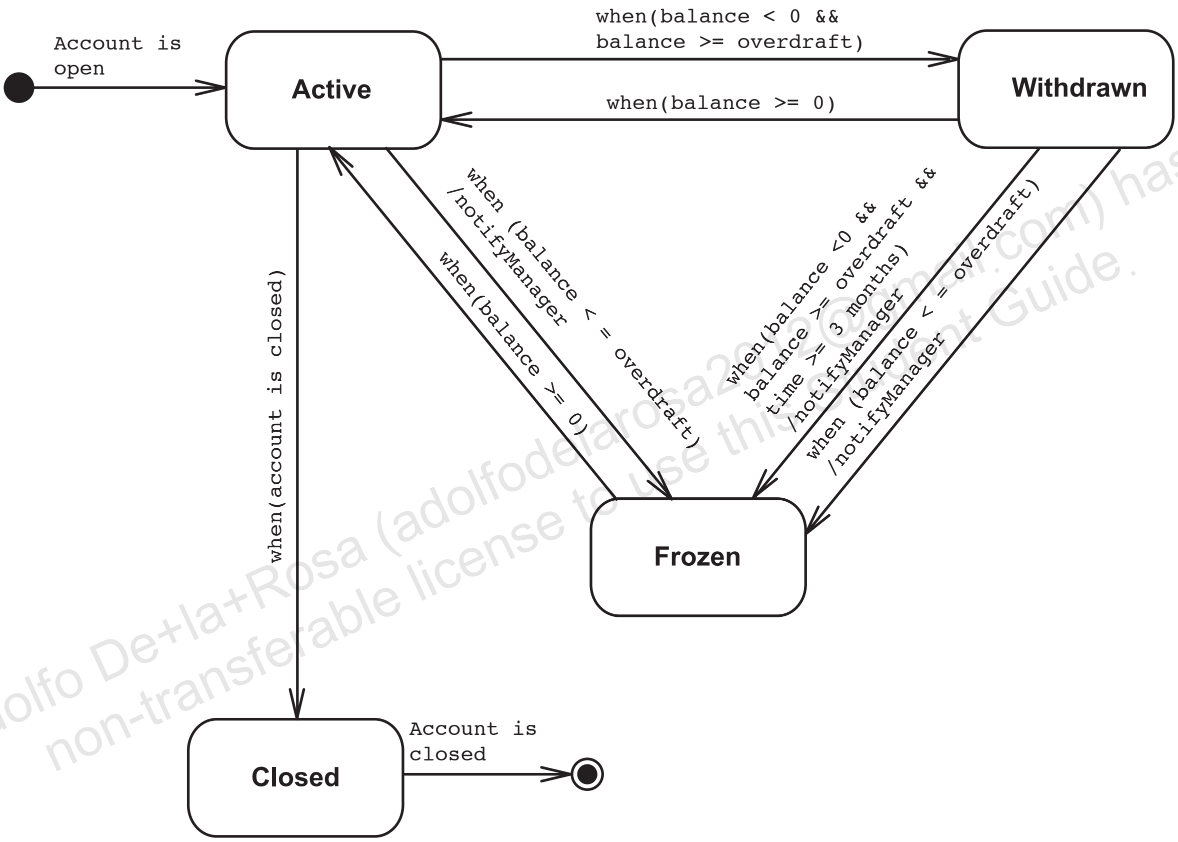

:CheckInUI
res :Reservation
cust :Customer
:Room
type :Room Type
Get Reservation Details Fragment
getCustomer
cust
getLastName
last name
getRooms
roomList
loop
[foreach room in roomlist]
getRoomId
room id
getRoomType
type
getDescription
description
getArrivalDate
arrival date
getDepartureDate
departure date
getQuotedPrice
price

Activity 1: Identifying State Machine Diagram Elements

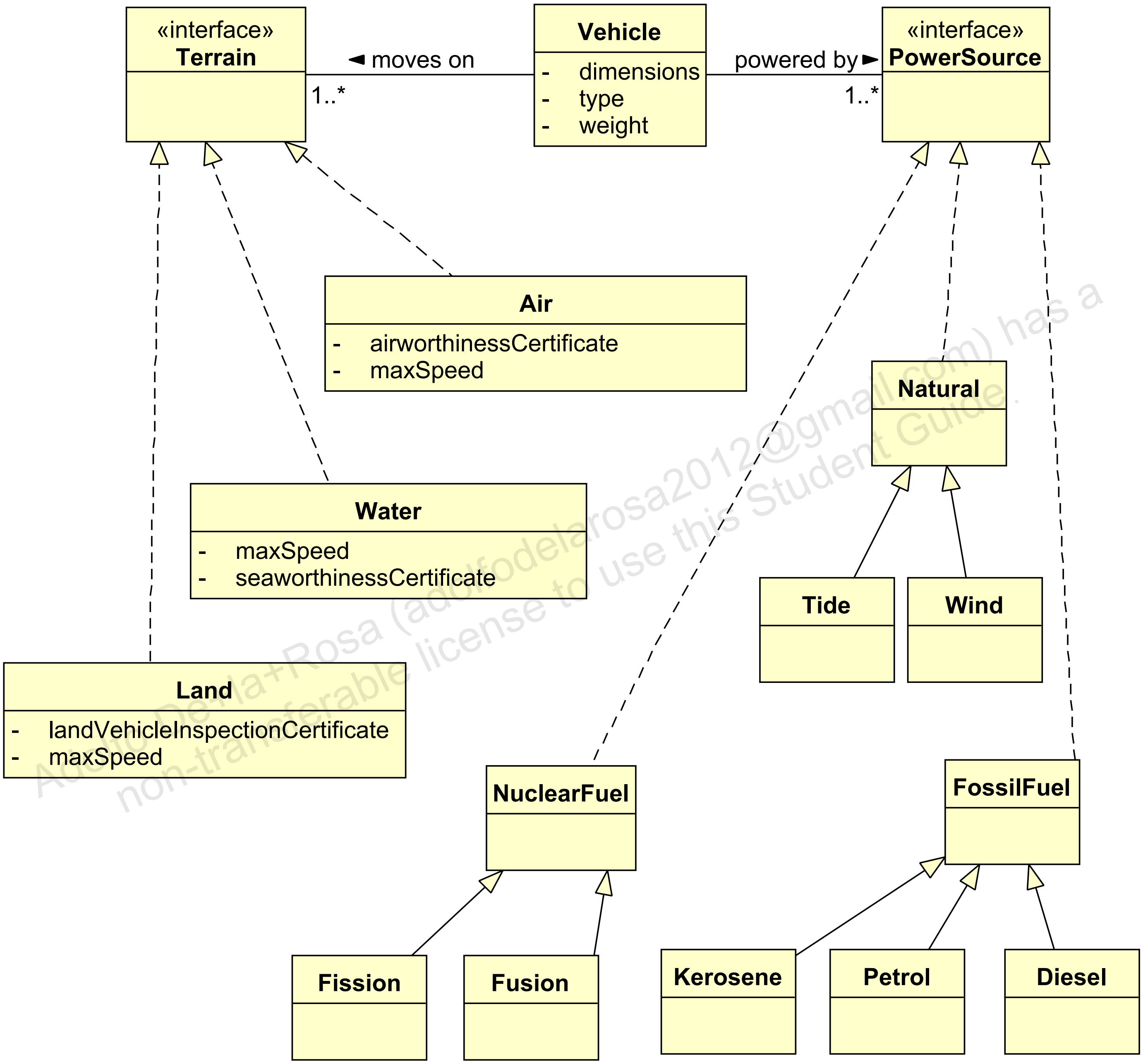
Self-Check – Write the name of each State Machine diagram symbol in the space allotted next to each symbol.

Symbol	Symbol Name
	State node
 AcctOverdrawn entry / notifyCustomer entry / doNotAllowSpending exit / allowSpending	State node with internal events
	Start state
	Stop state
	Transition arrow

Activity 2: Creating a State Diagram

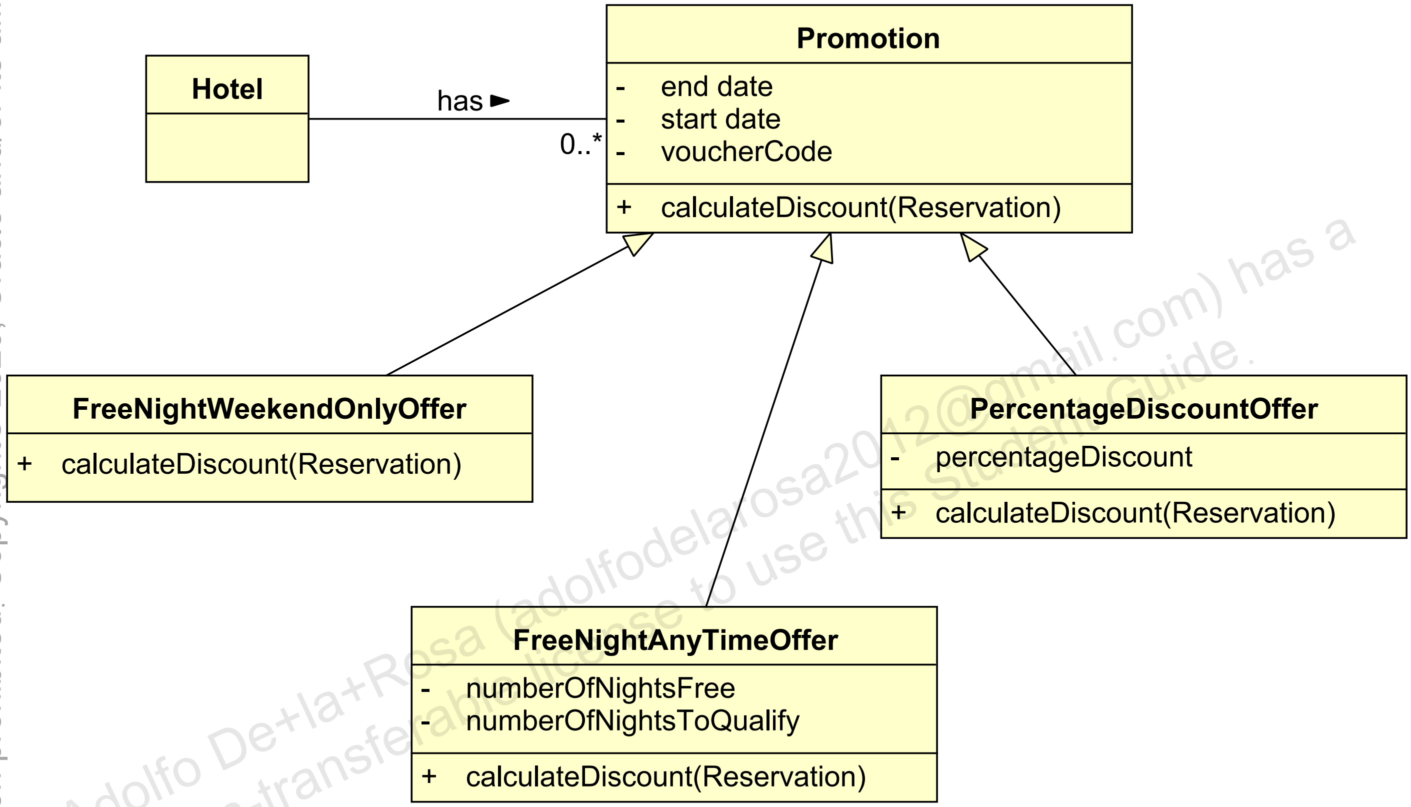


Activity 1: Applying CRP



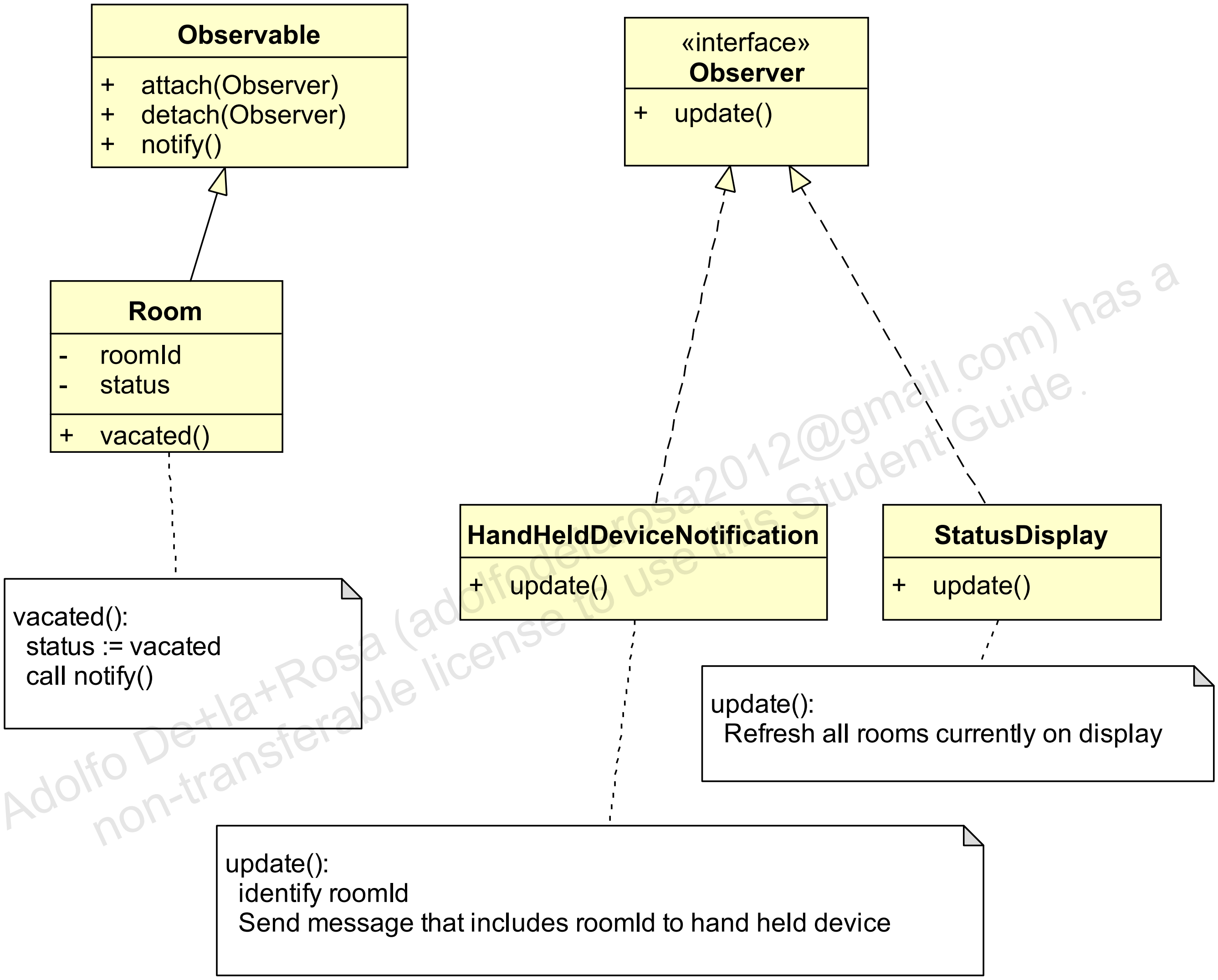
Note1: Terrain can be a class, in which case maxSpeed can be stored in the base class Terrain. However, if the units of measurement are different, this might not be the best approach.
Note 2: You need subclasses of PowerSource only if each subclass has different attributes or behaviour.

Activity 2: Applying the Strategy Pattern


Hotel
has ►
0..*
Promotion
- end date
- start date
- voucherCode
+ calculateDiscount(Reservation)
FreeNightWeekendOnlyOffer
+ calculateDiscount(Reservation)
PercentageDiscountOffer
- percentageDiscount
+ calculateDiscount(Reservation)
FreeNightAnyTimeOffer
- numberOfNightsFree
- numberOfNightsToQualify
+ calculateDiscount(Reservation)

Note 1: It may be possible to have FreeNightAnyTimeOffer and FreeNightWeekendOnlyOffer as the same class or share a common FreeNightOffer class.
Note 2: CalculateDiscount is passed the Reservation object, so it can get any information it needs that may affect whether the offer is valid for a reservation, for example the dates of the stay.
Note 3: Although the Promotion class has a voucher code, the default value could be that no voucher is needed.

Activity 3: Applying the Observer Pattern



Observable

- + attach(Observer)
- + detach(Observer)
- + notify()

«interface»
Observer

- + update()

Room

- - roomId
- - status
- + vacated()

HandHeldDeviceNotification

- + update()

StatusDisplay

- + update()

```
vacated():
  status := vacated
  call notify()
```

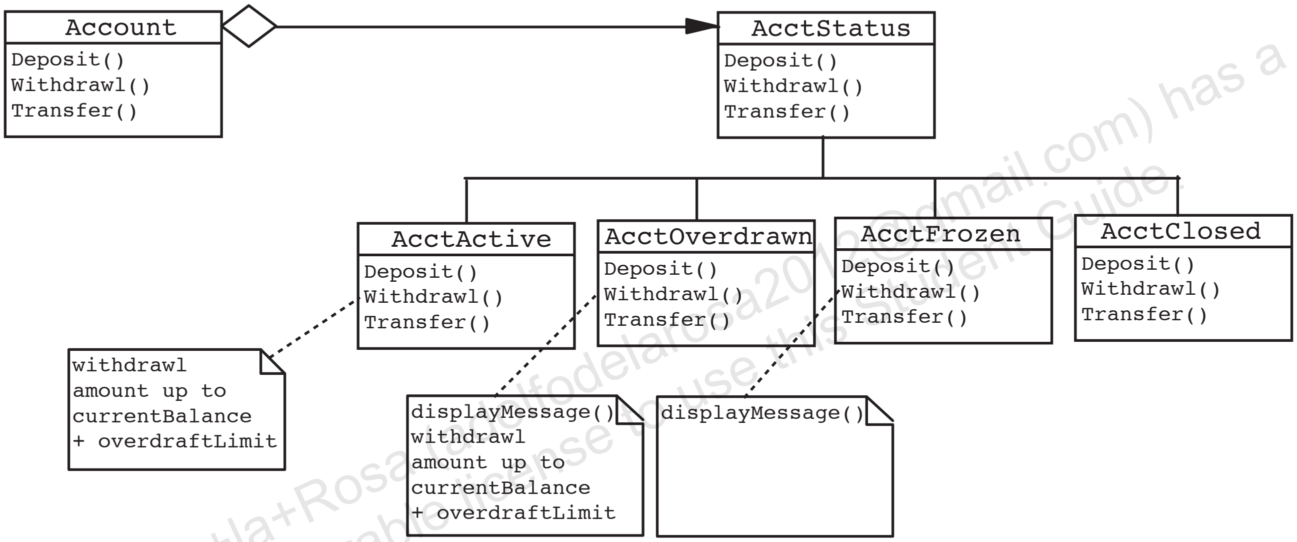
```
update():
  Refresh all rooms currently on display
```

```
update():
  identify roomId
  Send message that includes roomId to hand held device
```

Note 1:
It is possible to send an object as a parameter to the update method. This object can be the Room object, in which case it will be easier to identify the room.

Note 2:
Java^{TM} uses different methods.

Activity 4: Applying the State Pattern



Activity 1: Identifying Tiers, Layers, and Systemic Qualities

Self-Check – Match the layers with their definitions.

Layer	**Definition**
Application	Provides the APIs that application components implement
Virtual Platform	Consists of the operating system
Upper Platform	Includes computing components such as servers, storage, and network devices
Lower Platform	Provides a concrete implementation of components to satisfy the functional requirements
Hardware Platform	Consists of products such as web and containers and middleware

Answers: Virtual Platform, Lower Platform, Hardware Platform, Application, Upper Platform

Self-Check – Match the tiers with their definitions.

Tier	**Definition**
Client	Provides services and entities
Presentation	All back-end components, such as a Database Management System (DBMS) or Enterprise Information System (EIS)
Business	Usually described as “thin”; often is a web browser.
Integration	Provides the Hyper Text Markup Language (HTML) pages and forms sent to a web browser and process the user’s requests
Resource	Provides components that tie the business tier to the resource tier

Answers: Business, Resource, Client, Presentation, Integration

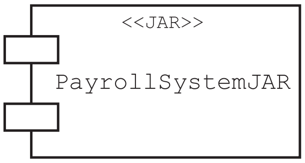
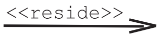
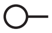
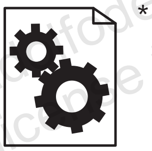
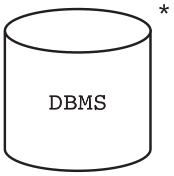
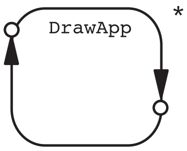
Self-Check – Match the systemic qualities with their definitions.

Systemic Quality	Definition
Developmental	Addresses the requisite qualities as the system evolves
Manifest	Addresses the qualities reflected in the execution of the system
Evolutionary	Addresses the requisite qualities in production
Operational	Addresses the requisite qualities during system development

Answers: Evolutionary, Manifest, Operational, Developmental

Activity 2: Exploring Component Diagrams

Self-Check – Write the name of each Component diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	Component node
	Dependency arrow
	Interface icon
	Executable file component
	DBMS component
	Application component

✓ **Self-Check –** Select the statement or statements about UML Component diagrams that are TRUE.

a. ___ A component represents a software unit.

b. ___ There are descriptor and instance forms of Component diagrams.

c. ___ Components cannot be abstract.

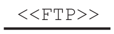
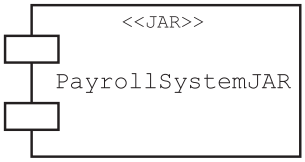
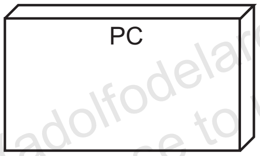
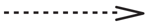
d. ___ Component diagrams show the organizations and dependencies among components.

e. ___ You can create your own component icons to extend the UML.

Answers: a, d, and e

Activity 3: Understanding Deployment Diagrams

Self-Check – Write the name of each Deployment diagram symbol in the space next to each symbol.

Symbol	Symbol Name
	Communication link
	Component node
	Hardware node
	Dependency arrow

Self-Check – Select the statement or statements about UML Deployment diagrams that are TRUE.

a. ___ You can assign your own icons to represent hardware in Deployment diagrams.

b. ___ There are descriptor and instance forms of Deployment diagrams.

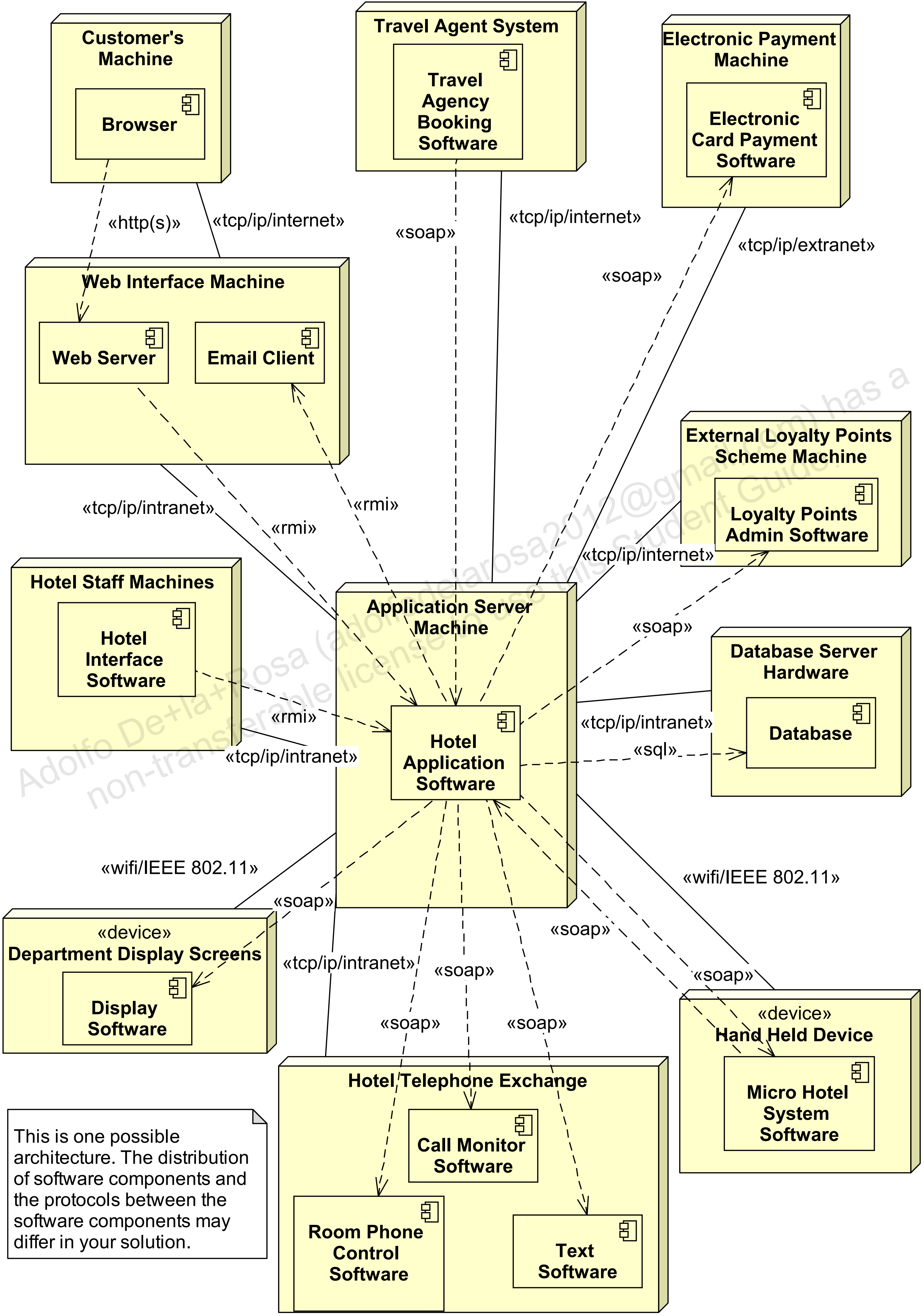
c. ___ Descriptor Deployment diagrams show a particular deployment of a system.

d. ___ The «`deploy`» stereotype can be used to document components within a node.

e. ___ There is only a descriptor form of Deployment diagrams.

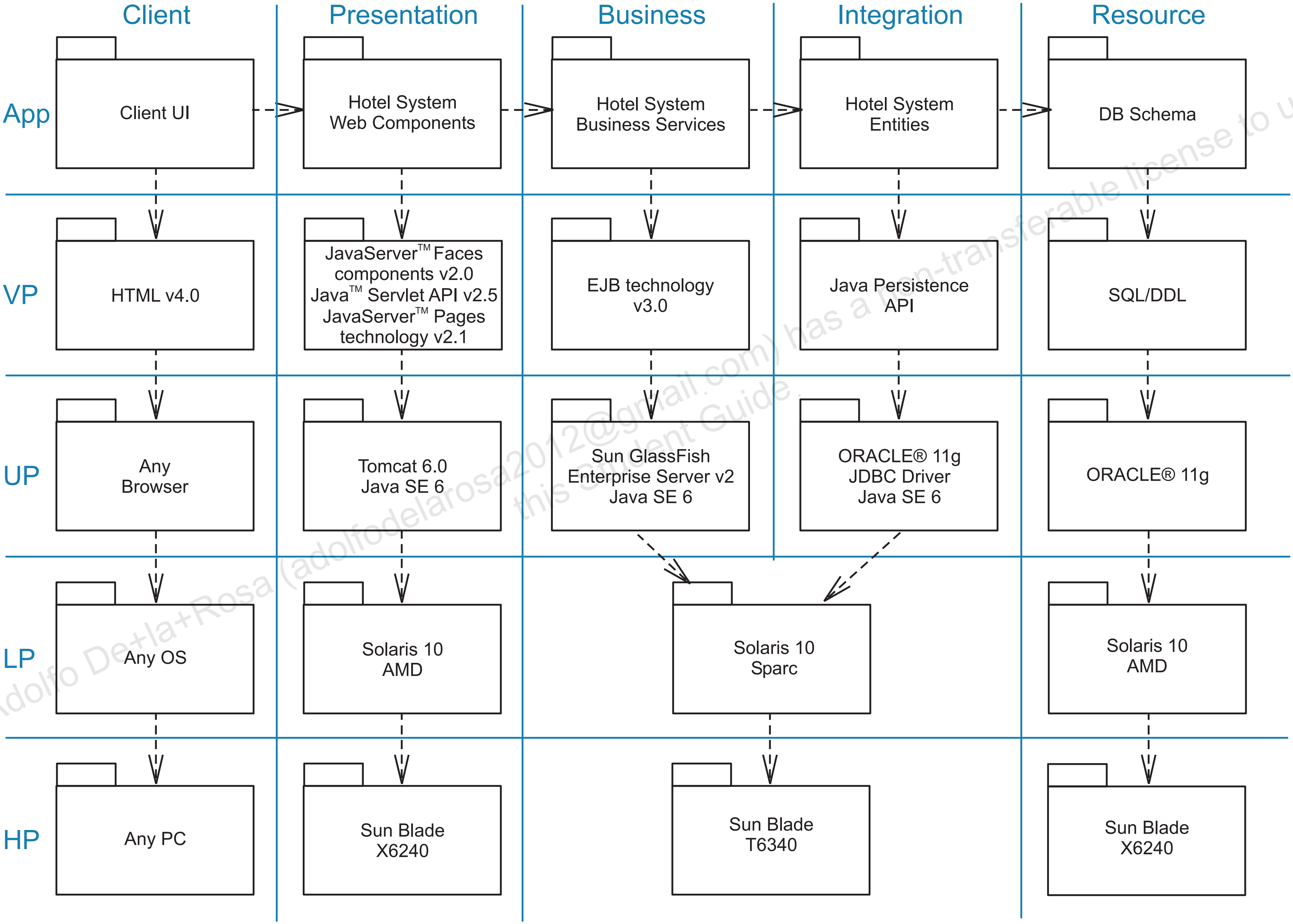
Answers: a, b, and d

Activity 4: Creating a High-Level Deployment Diagram



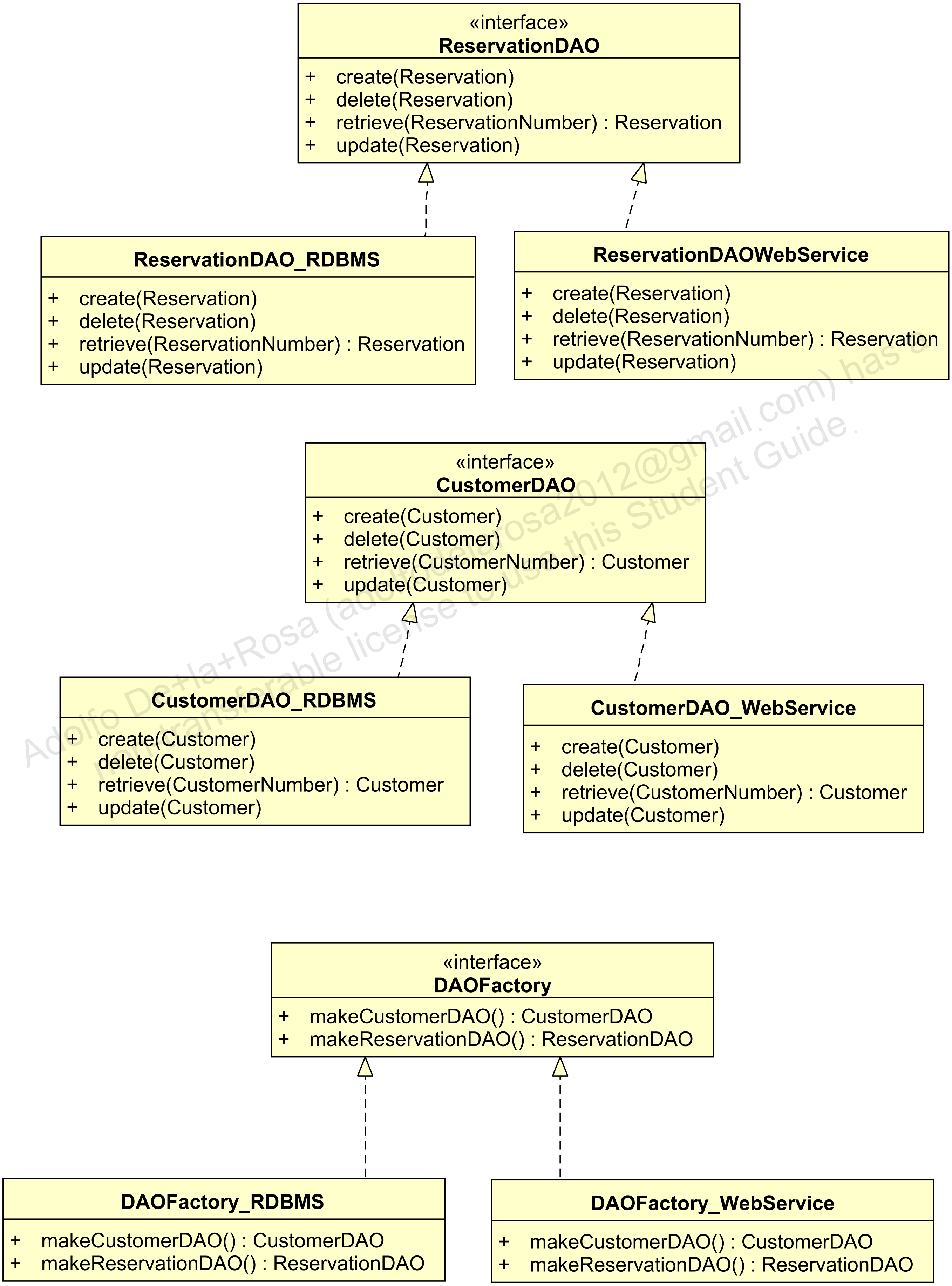
This is one possible architecture. The distribution of software components and the protocols between the software components may differ in your solution.

Activity 1: Update and Extend a Tiers and Layers diagram

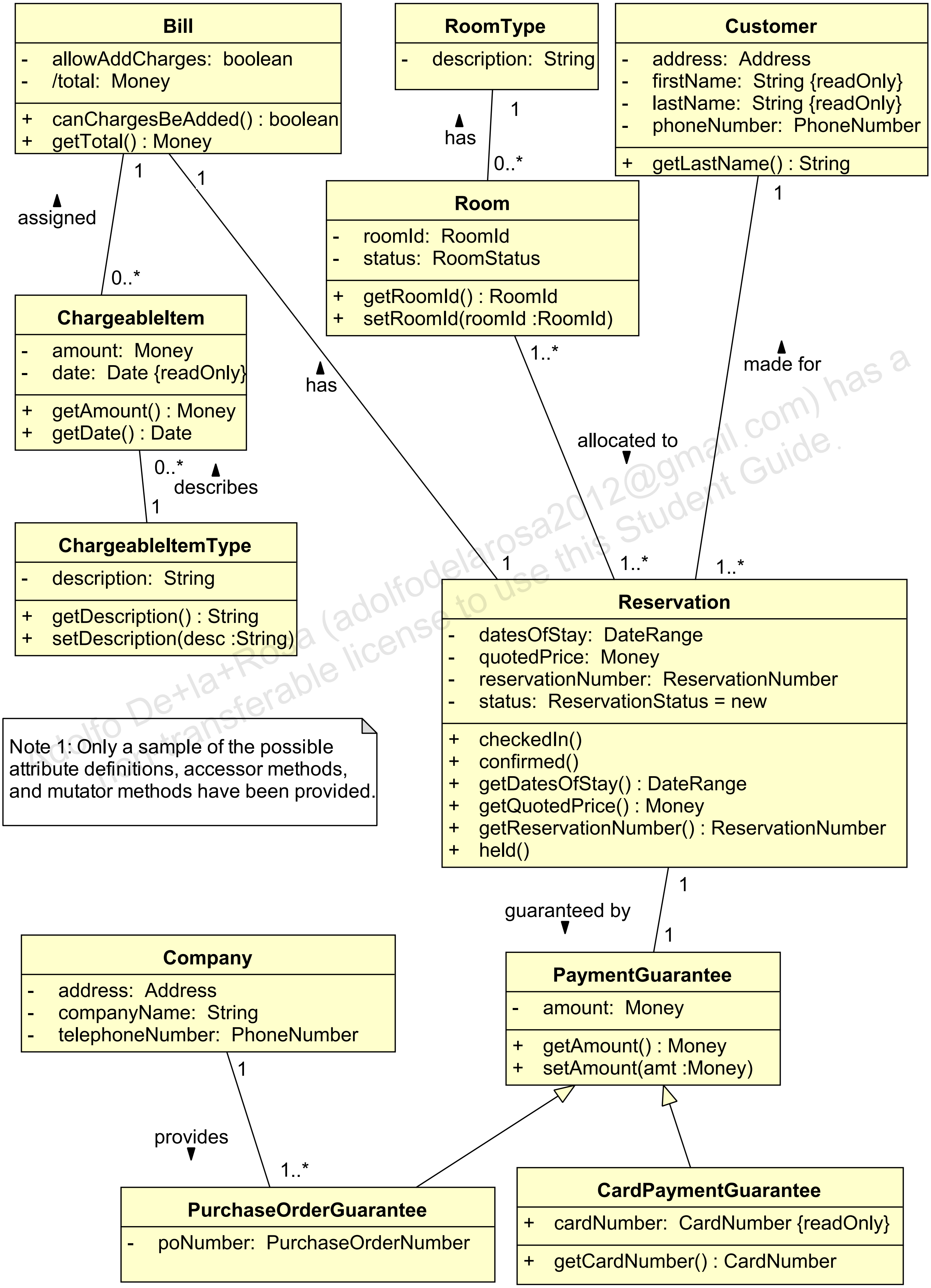


	Client	Presentation	Business	Integration	Resource
App	Client UI	Hotel System Web Components	Hotel System Business Services	Hotel System Entities	DB Schema
VP	HTML v4.0	JavaServer™ Faces components v2.0 Java™ Servlet API v2.5 JavaServer™ Pages technology v2.1	EJB technology v3.0	Java Persistence API	SQL/DDL
UP	Any Browser	Tomcat 6.0 Java SE 6	Sun GlassFish Enterprise Server v2 Java SE 6	ORACLE® 11g JDBC Driver Java SE 6	ORACLE® 11g
LP	Any OS	Solaris 10 AMD	Solaris 10 Sparc		Solaris 10 AMD
HP	Any PC	Sun Blade X6240	Sun Blade T6340		Sun Blade X6240

Activity 2: Create DAOFactory and DAO Classes

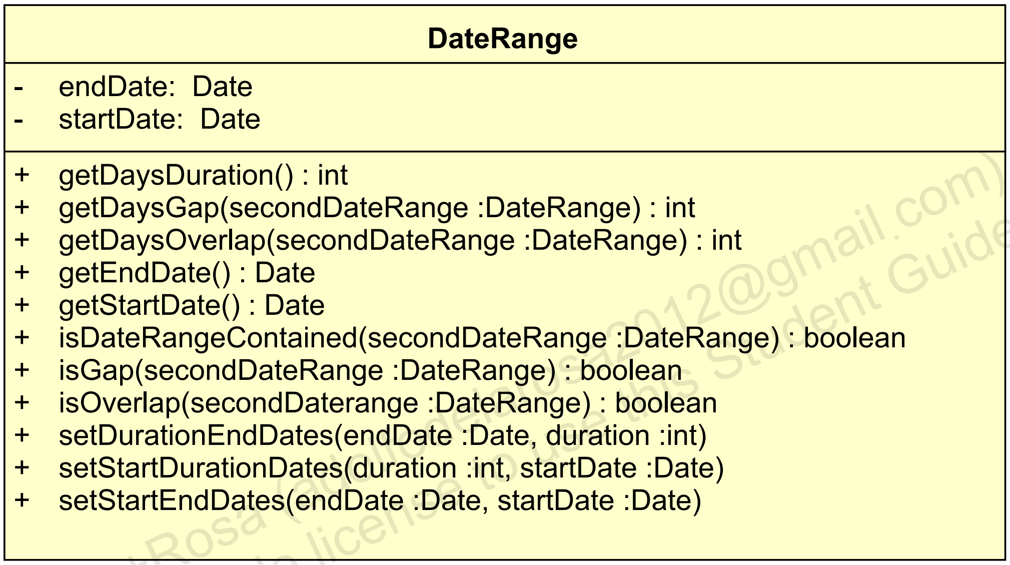


Activity 1: Refining Attributes



Note 1: Only a sample of the possible attribute definitions, accessor methods, and mutator methods have been provided.

Activity 2: Refining Associations using Aggregation and Composition



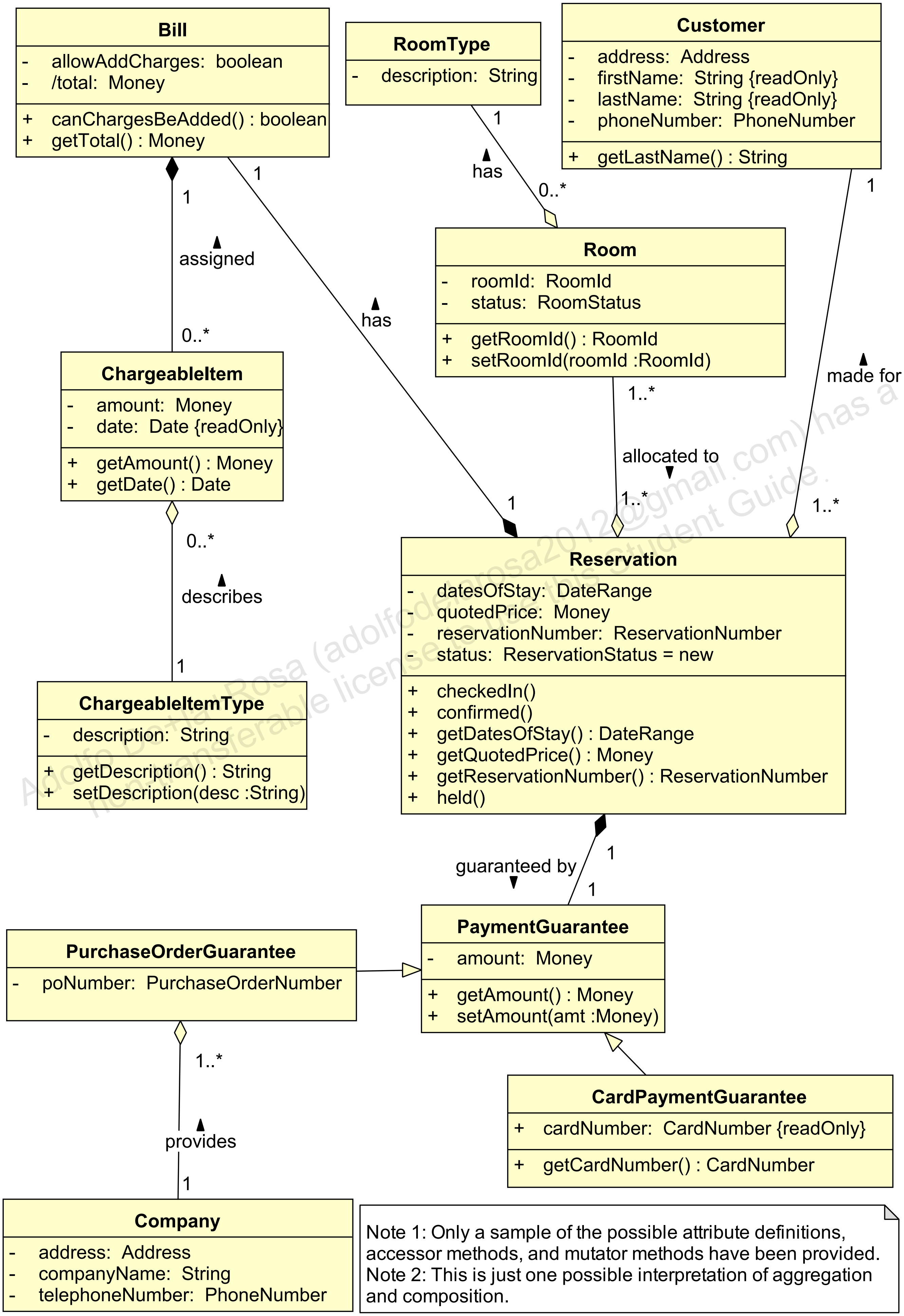
DateRange

- endDate: Date
- startDate: Date

+ getDaysDuration() : int
+ getDaysGap(secondDateRange :DateRange) : int
+ getDaysOverlap(secondDateRange :DateRange) : int
+ getEndDate() : Date
+ getStartDate() : Date
+ isDateRangeContained(secondDateRange :DateRange) : boolean
+ isGap(secondDateRange :DateRange) : boolean
+ isOverlap(secondDaterange :DateRange) : boolean
+ setDurationEndDates(endDate :Date, duration :int)
+ setStartDurationDates(duration :int, startDate :Date)
+ setStartEndDates(endDate :Date, startDate :Date)

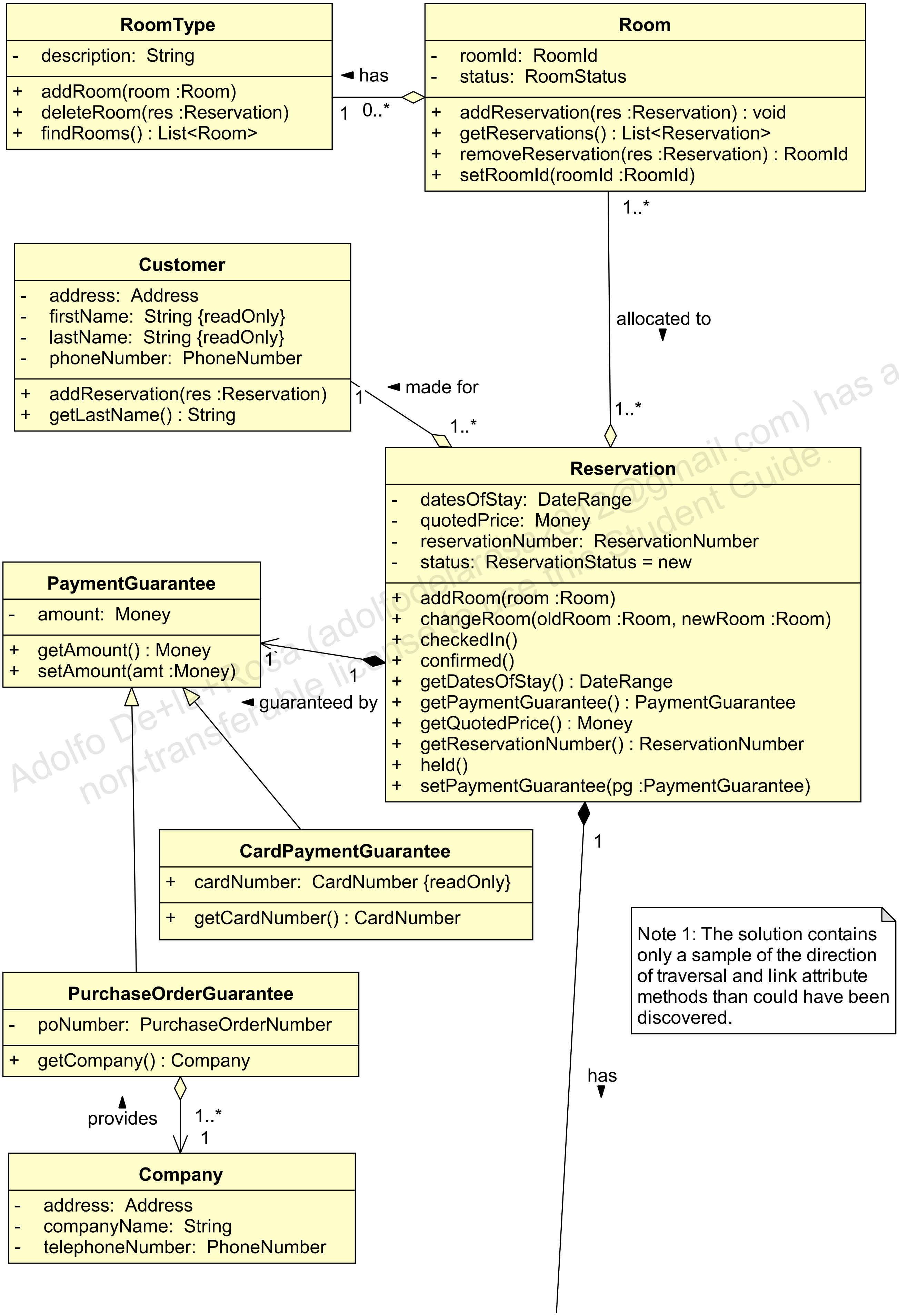
Note 1: The solution shows only a subset of methods that will be useful in the Hotel System domain and in other domains.

Note 2: You can also have class (static) methods for many of these methods.

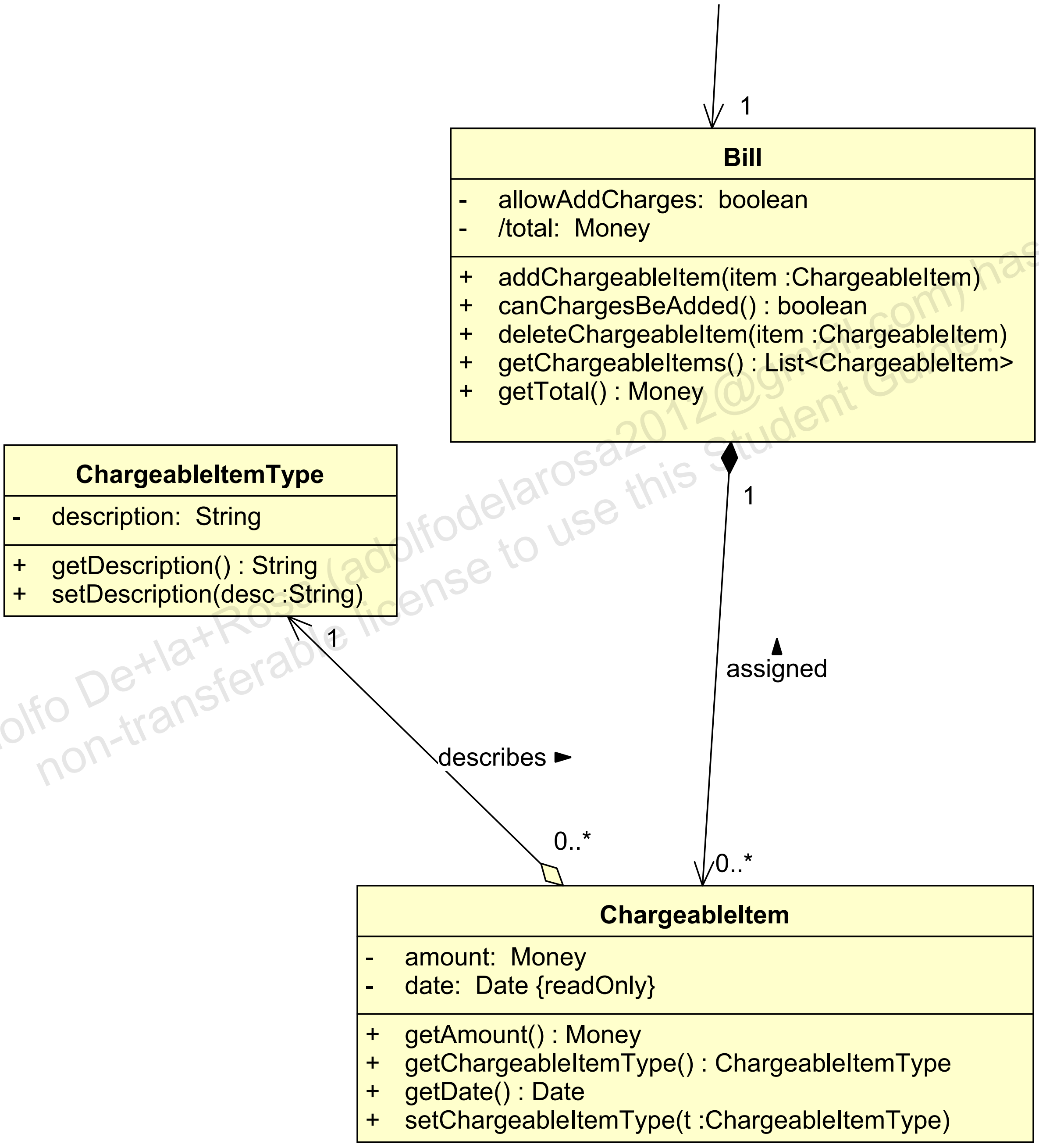


Note 1: Only a sample of the possible attribute definitions, accessor methods, and mutator methods have been provided.
Note 2: This is just one possible interpretation of aggregation and composition.

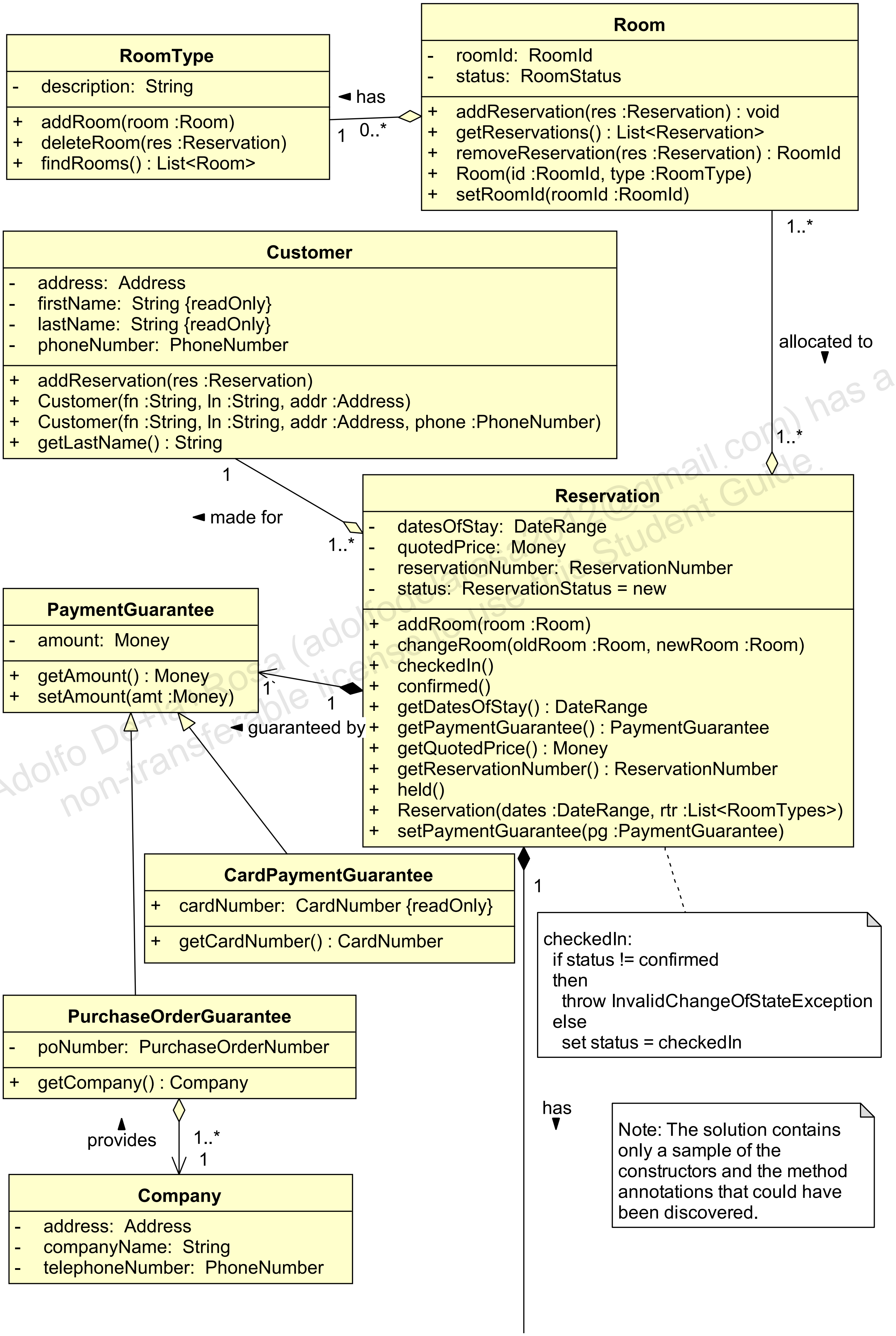
Activity 3: Refining the Direction of Traversal

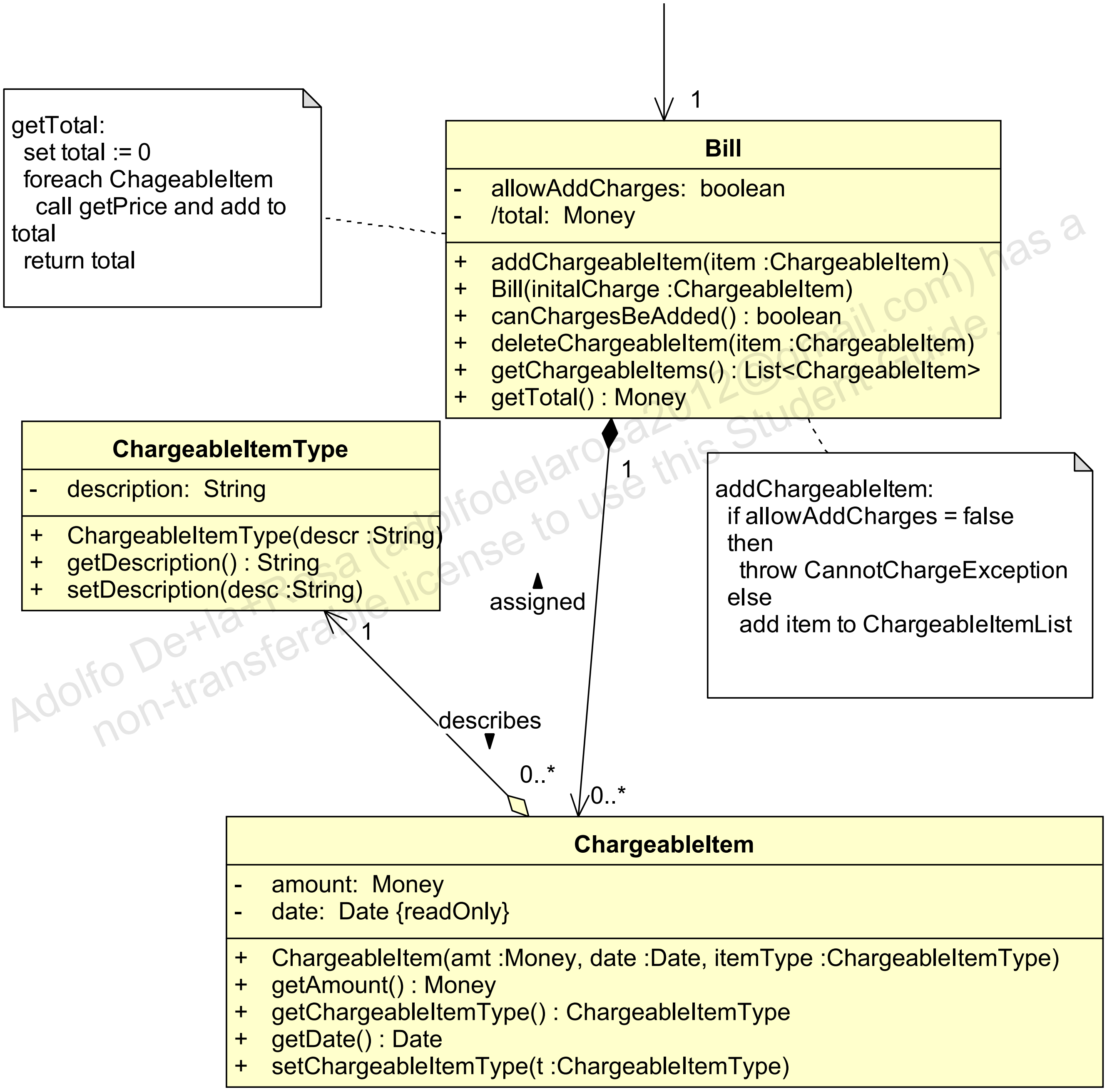

RoomType
- description: String
+ addRoom(room :Room)
+ deleteRoom(res :Reservation)
+ findRooms() : List<Room>
◄ has
1
0..*
Room
- roomId: RoomId
- status: RoomStatus
+ addReservation(res :Reservation) : void
+ getReservations() : List<Reservation>
+ removeReservation(res :Reservation) : RoomId
+ setRoomId(roomId :RoomId)
1..*
allocated to
▼
1..*
Customer
- address: Address
- firstName: String {readOnly}
- lastName: String {readOnly}
- phoneNumber: PhoneNumber
+ addReservation(res :Reservation)
+ getLastName() : String
1
◄ made for
1..*
Reservation
- datesOfStay: DateRange
- quotedPrice: Money
- reservationNumber: ReservationNumber
- status: ReservationStatus = new
+ addRoom(room :Room)
+ changeRoom(oldRoom :Room, newRoom :Room)
+ checkedIn()
+ confirmed()
+ getDatesOfStay() : DateRange
+ getPaymentGuarantee() : PaymentGuarantee
+ getQuotedPrice() : Money
+ getReservationNumber() : ReservationNumber
+ held()
+ setPaymentGuarantee(pg :PaymentGuarantee)
PaymentGuarantee
- amount: Money
+ getAmount() : Money
+ setAmount(amt :Money)
1`
1
◄ guaranteed by
1
CardPaymentGuarantee
+ cardNumber: CardNumber {readOnly}
+ getCardNumber() : CardNumber
PurchaseOrderGuarantee
- poNumber: PurchaseOrderNumber
+ getCompany() : Company
▲
provides
1..*
1
Company
- address: Address
- companyName: String
- telephoneNumber: PhoneNumber
has
▼

Note 1: The solution contains only a sample of the direction of traversal and link attribute methods than could have been discovered.


1
Bill
- allowAddCharges: boolean
- /total: Money
+ addChargeableItem(item :ChargeableItem)
+ canChargesBeAdded() : boolean
+ deleteChargeableItem(item :ChargeableItem)
+ getChargeableItems() : List<ChargeableItem>
+ getTotal() : Money
ChargeableItemType
- description: String
+ getDescription() : String
+ setDescription(desc :String)
1
1
assigned ▲
describes ►
0..*
0..*
ChargeableItem
- amount: Money
- date: Date {readOnly}
+ getAmount() : Money
+ getChargeableItemType() : ChargeableItemType
+ getDate() : Date
+ setChargeableItemType(t :ChargeableItemType)

Activity 4: Refining Business Methods and Constructors

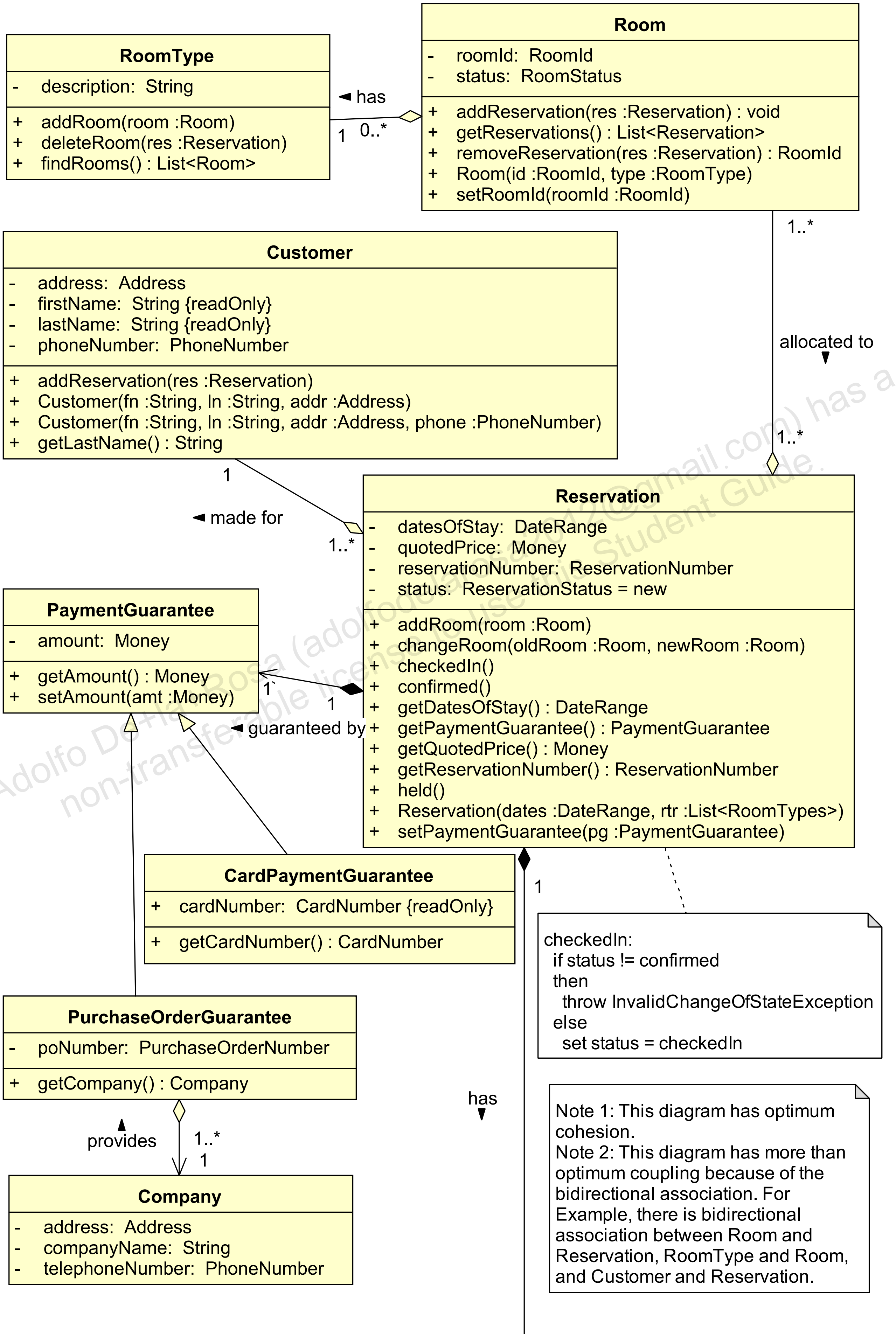

RoomType
- description: String
+ addRoom(room :Room)
+ deleteRoom(res :Reservation)
+ findRooms() : List<Room>
◄ has
1
0..*
Room
- roomId: RoomId
- status: RoomStatus
+ addReservation(res :Reservation) : void
+ getReservations() : List<Reservation>
+ removeReservation(res :Reservation) : RoomId
+ Room(id :RoomId, type :RoomType)
+ setRoomId(roomId :RoomId)
1..*
allocated to ▼
1..*
Customer
- address: Address
- firstName: String {readOnly}
- lastName: String {readOnly}
- phoneNumber: PhoneNumber
+ addReservation(res :Reservation)
+ Customer(fn :String, ln :String, addr :Address)
+ Customer(fn :String, ln :String, addr :Address, phone :PhoneNumber)
+ getLastName() : String
1
◄ made for
1..*
Reservation
- datesOfStay: DateRange
- quotedPrice: Money
- reservationNumber: ReservationNumber
- status: ReservationStatus = new
+ addRoom(room :Room)
+ changeRoom(oldRoom :Room, newRoom :Room)
+ checkedIn()
+ confirmed()
+ getDatesOfStay() : DateRange
+ getPaymentGuarantee() : PaymentGuarantee
+ getQuotedPrice() : Money
+ getReservationNumber() : ReservationNumber
+ held()
+ Reservation(dates :DateRange, rtr :List<RoomTypes>)
+ setPaymentGuarantee(pg :PaymentGuarantee)
PaymentGuarantee
- amount: Money
+ getAmount() : Money
+ setAmount(amt :Money)
1`
1
◄ guaranteed by
CardPaymentGuarantee
+ cardNumber: CardNumber {readOnly}
+ getCardNumber() : CardNumber
1
checkedIn:
if status != confirmed
then
throw InvalidChangeOfStateException
else
set status = checkedIn
PurchaseOrderGuarantee
- poNumber: PurchaseOrderNumber
+ getCompany() : Company
▲ provides
1..*
1
has ▼
Note: The solution contains only a sample of the constructors and the method annotations that could have been discovered.
Company
- address: Address
- companyName: String
- telephoneNumber: PhoneNumber


1
Bill
- allowAddCharges: boolean
- /total: Money
+ addChargeableItem(item :ChargeableItem)
+ Bill(initalCharge :ChargeableItem)
+ canChargesBeAdded() : boolean
+ deleteChargeableItem(item :ChargeableItem)
+ getChargeableItems() : List<ChargeableItem>
+ getTotal() : Money
ChargeableItemType
- description: String
+ ChargeableItemType(descr :String)
+ getDescription() : String
+ setDescription(desc :String)
1
assigned ▲
1
describes ▼
0..*
0..*
ChargeableItem
- amount: Money
- date: Date {readOnly}
+ ChargeableItem(amt :Money, date :Date, itemType :ChargeableItemType)
+ getAmount() : Money
+ getChargeableItemType() : ChargeableItemType
+ getDate() : Date
+ setChargeableItemType(t :ChargeableItemType)

```
getTotal:
  set total := 0
  foreach ChageableItem
    call getPrice and add to
total
  return total
```

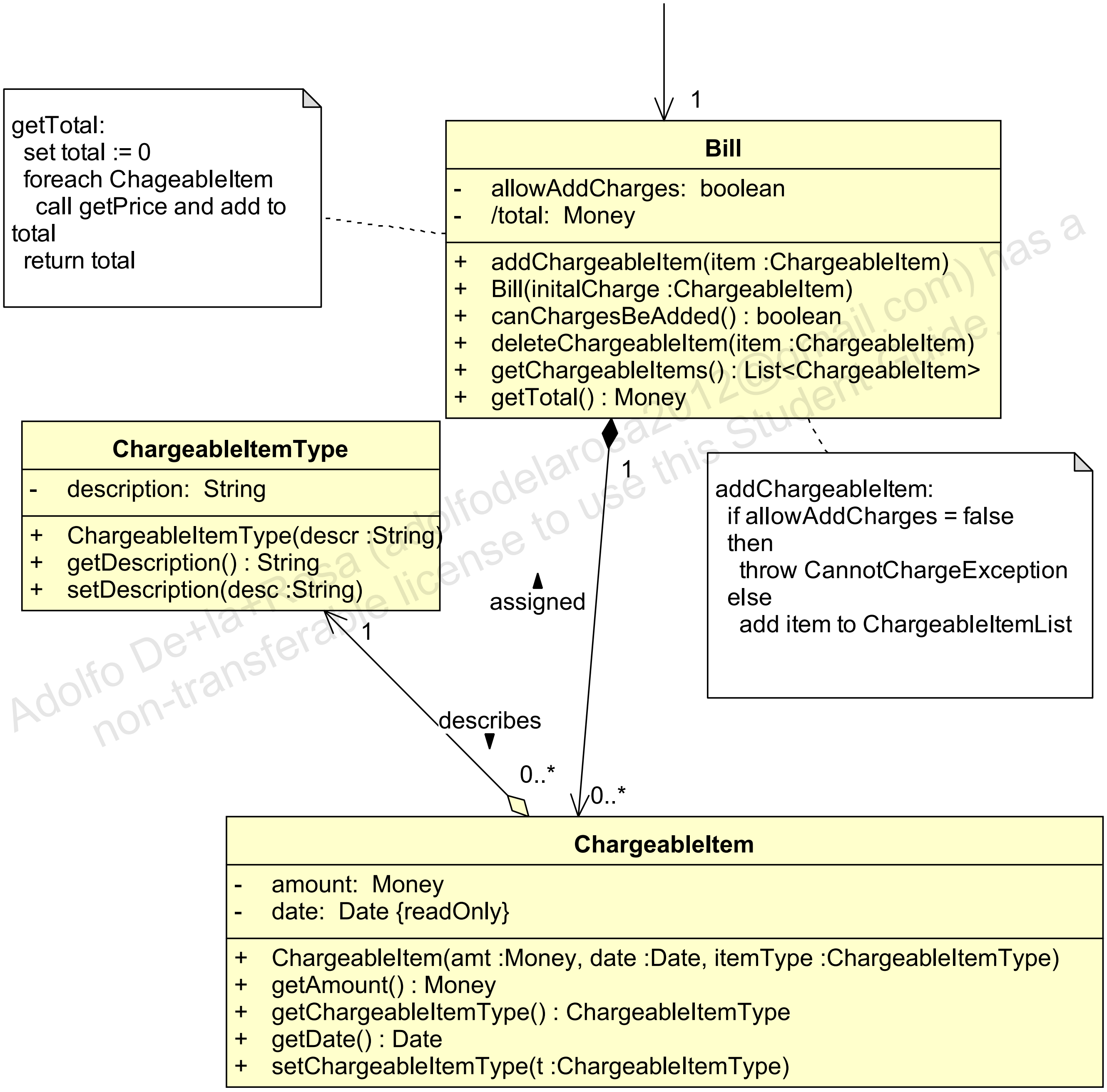
```
addChargeableItem:
  if allowAddCharges = false
  then
    throw CannotChargeException
  else
    add item to ChargeableItemList
```

Activity 5: Checking your Class Diagram for High Cohesion and Low Coupling

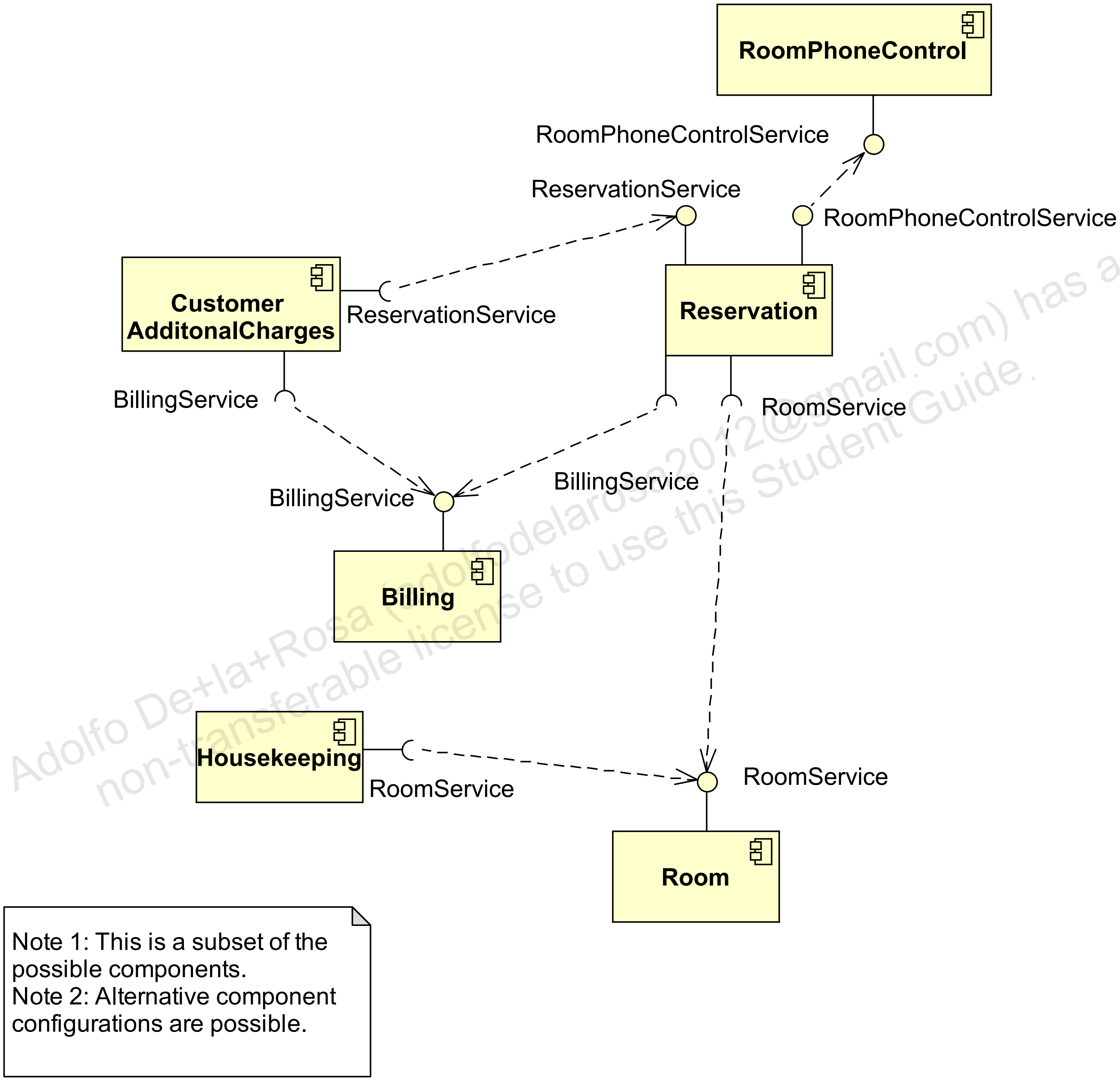

RoomType
- description: String
+ addRoom(room :Room)
+ deleteRoom(res :Reservation)
+ findRooms() : List<Room>
◄ has
1
0..*
Room
- roomId: RoomId
- status: RoomStatus
+ addReservation(res :Reservation) : void
+ getReservations() : List<Reservation>
+ removeReservation(res :Reservation) : RoomId
+ Room(id :RoomId, type :RoomType)
+ setRoomId(roomId :RoomId)
1..*
allocated to ▼
1..*
Customer
- address: Address
- firstName: String {readOnly}
- lastName: String {readOnly}
- phoneNumber: PhoneNumber
+ addReservation(res :Reservation)
+ Customer(fn :String, ln :String, addr :Address)
+ Customer(fn :String, ln :String, addr :Address, phone :PhoneNumber)
+ getLastName() : String
1
◄ made for
1..*
Reservation
- datesOfStay: DateRange
- quotedPrice: Money
- reservationNumber: ReservationNumber
- status: ReservationStatus = new
+ addRoom(room :Room)
+ changeRoom(oldRoom :Room, newRoom :Room)
+ checkedIn()
+ confirmed()
+ getDatesOfStay() : DateRange
+ getPaymentGuarantee() : PaymentGuarantee
+ getQuotedPrice() : Money
+ getReservationNumber() : ReservationNumber
+ held()
+ Reservation(dates :DateRange, rtr :List<RoomTypes>)
+ setPaymentGuarantee(pg :PaymentGuarantee)
PaymentGuarantee
- amount: Money
+ getAmount() : Money
+ setAmount(amt :Money)
1`
1
◄ guaranteed by
CardPaymentGuarantee
+ cardNumber: CardNumber {readOnly}
+ getCardNumber() : CardNumber
1
PurchaseOrderGuarantee
- poNumber: PurchaseOrderNumber
+ getCompany() : Company
has ▼
▲ provides
1..*
1
Company
- address: Address
- companyName: String
- telephoneNumber: PhoneNumber

```
checkedIn:
 if status != confirmed
 then
  throw InvalidChangeOfStateException
 else
  set status = checkedIn
```

Note 1: This diagram has optimum cohesion.
Note 2: This diagram has more than optimum coupling because of the bidirectional association. For Example, there is bidirectional association between Room and Reservation, RoomType and Room, and Customer and Reservation.


1
Bill
- allowAddCharges: boolean
- /total: Money
+ addChargeableItem(item :ChargeableItem)
+ Bill(initalCharge :ChargeableItem)
+ canChargesBeAdded() : boolean
+ deleteChargeableItem(item :ChargeableItem)
+ getChargeableItems() : List<ChargeableItem>
+ getTotal() : Money
getTotal:
set total := 0
foreach ChageableItem
call getPrice and add to total
return total
ChargeableItemType
- description: String
+ ChargeableItemType(descr :String)
+ getDescription() : String
+ setDescription(desc :String)
addChargeableItem:
if allowAddCharges = false
then
throw CannotChargeException
else
add item to ChargeableItemList
1
assigned
1
describes
0..*
0..*
ChargeableItem
- amount: Money
- date: Date {readOnly}
+ ChargeableItem(amt :Money, date :Date, itemType :ChargeableItemType)
+ getAmount() : Money
+ getChargeableItemType() : ChargeableItemType
+ getDate() : Date
+ setChargeableItemType(t :ChargeableItemType)

Activity 6: Creating Components with Interfaces



Note 1: This is a subset of the possible components.
Note 2: Alternative component configurations are possible.

Activity 1: Identifying Methodology Characteristics

Self-Check – Select the statement or statements about Use-Case-driven methodologies that are true.

a. ___ Focus on relationships between actors and the system

b. ___ Based on the notion that software performs activities for users

c. ___ Uses non-functional requirements to drive structure of the system

d. ___ Must be iterative

e. ___ Focus on the systemic qualities, such as reliability and scalability

Answers: a, b

- Answer c is wrong because NFRs are not relevant to use cases.
- Answer d is wrong because a methodology that is iterative is independent of being Use-Case-driven.
- Answer e is wrong because systemic qualities are NFRs and are not relevant to use cases.

Self-Check – Select the statement or statements about Architecture-centric methodologies that are true.

a. ___ Focus on relationships between actors and the system

b. ___ Based on the notion that software performs activities for users

c. ___ Uses non-functional requirements to drive the architecture of the system

d. ___ Must be iterative

e. ___ Focus on the systemic qualities, such as reliability and scalability

Answers: c, e

- Answer a is wrong because Architecture-centric is focused on systemic qualities, not actors and their use cases.
- Answer b is wrong for the same reason as answer a.
- Answer d is wrong because a methodology can be Architecture-centric without being iterative.

Activity 2: Defining the Five Object-Oriented Methodologies

Self-Check – Match the object-oriented software development methodology terms with their definitions.

Term	Definition
Waterfall	An iterative software development process, created by Booch, Jacobson, and Rumbaugh, that is freely available for use.
eXtreme Programming (XP)	Coding and testing are the key activities within this methodology.
Unified Software Development Process (UP)	Team-oriented framework, where each Sprint produces a working version of the software.
Rational Unified Process (RUP)	This methodology uses a single phase in which all workflows proceed in a linear fashion.
Scrum	A commercial implementation of the UP methodology.

Answers: UP, XP, Scrum, Waterfall, RUP

Activity 3: Producing an Iteration Plan

Abstract of a Suggested Hotel System Iteration Plan

Note: Your plan might vary as the assessment of risk and priority is subjective.

Elaboration		**Construction**					
Iter#2	**Iter#3**	**Iter#4**	**Iter#5**	**Iter#6**	**Iter#7**	**Iter#8**	**Iter#9**
Create Reservation	Perform Card Payment Transaction	Perform Payment Cash Transaction	Update Reservation	Delete Reservation	Request List of Vacated Rooms	Retrieve Reservation Statistics	Publish Special Promotions to Customers
Simulation Customer Data	Add Charge to Customer's Bill	Create Customer	Identify Existing Reservation	Create Promotion	Mark Room Ready for Use	Modify Promotion	Notify Guest of Today's Reservation
Check In Customer	Check Out Customer	Change Room Phone Status	Adjust Customer's Bill	Register Customer Loyalty Card	Generate Nightly Reports	Generate Discrepancy Report	Modify Customer
Simulation Change Room Phone Status				Add Loyalty Points			

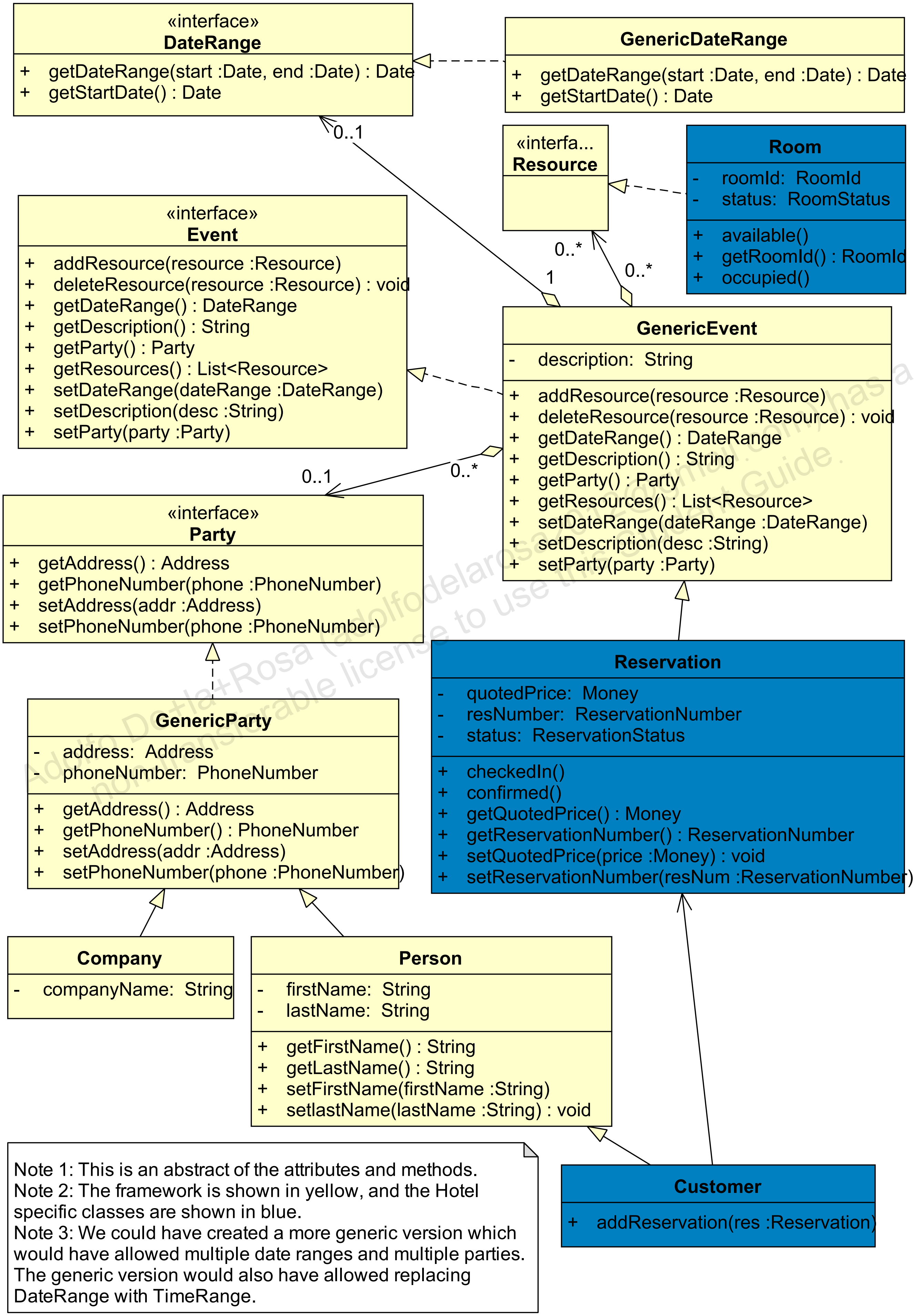
Exercise Solution for Module 14, Activity 4

Methodology Selection : UP, SunTone AM, or RUP
Justification: Members have flexible jobs, medium/large scale project, company is process-oriented (ISO).

Methodology Selection: XP or Waterfall
Justification: Company culture favors experimentation, teams are in close proximity, and the company has two experienced programmers. Could use waterfall because similar system has already been built by the engineers at a previous company.

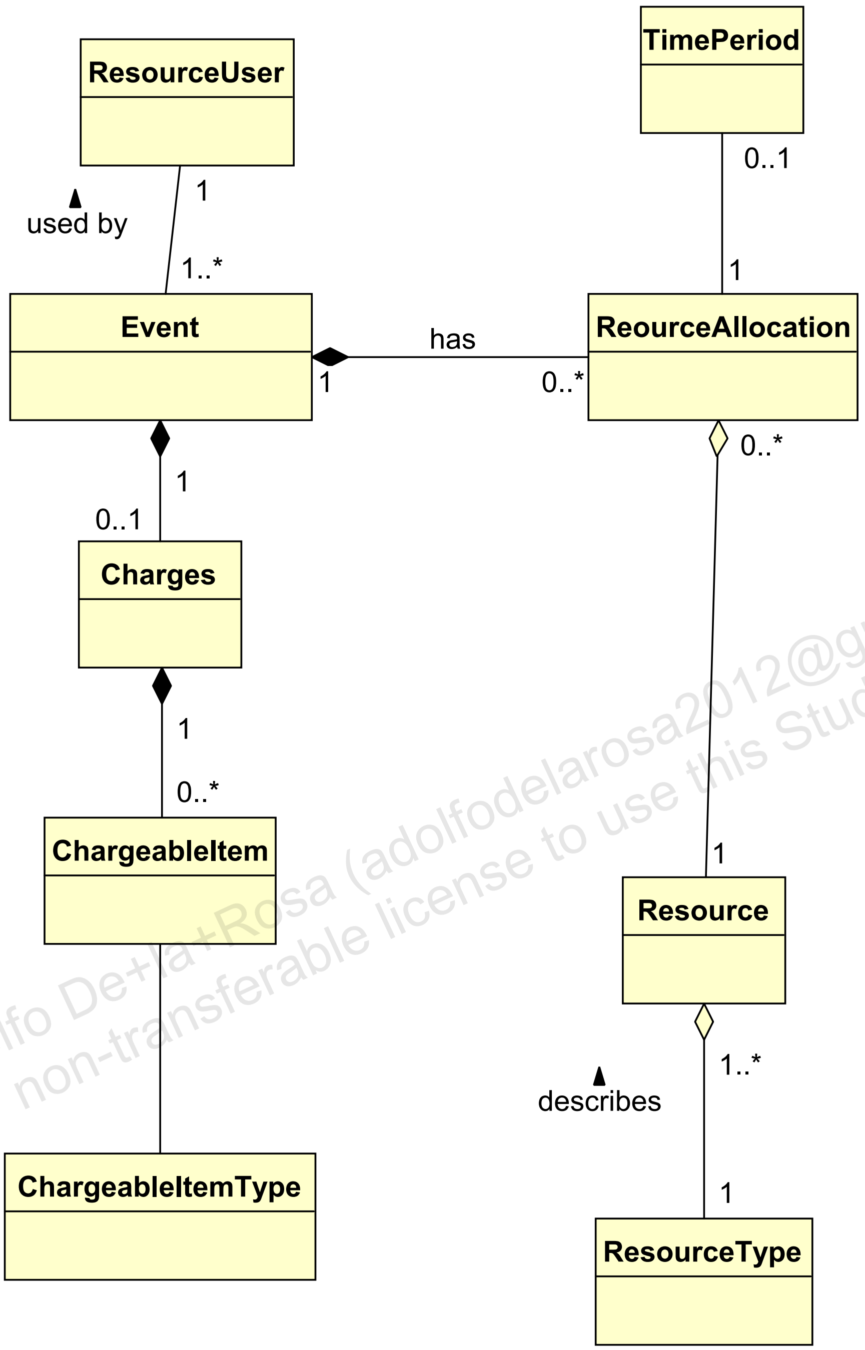
Methodology Selection: RUP or UP
Justification: Members have flexible jobs, medium/large scale project, company is process-oriented (PLC), and it has license for Rational Rose.

Activity 1: Creating a Conceptual Framework


«interface»
DateRange
+ getDateRange(start :Date, end :Date) : Date
+ getStartDate() : Date
GenericDateRange
+ getDateRange(start :Date, end :Date) : Date
+ getStartDate() : Date
0..1
«interfa...
Resource
Room
- roomId: RoomId
- status: RoomStatus
+ available()
+ getRoomId() : RoomId
+ occupied()
0..*
1
0..*
«interface»
Event
+ addResource(resource :Resource)
+ deleteResource(resource :Resource) : void
+ getDateRange() : DateRange
+ getDescription() : String
+ getParty() : Party
+ getResources() : List<Resource>
+ setDateRange(dateRange :DateRange)
+ setDescription(desc :String)
+ setParty(party :Party)
GenericEvent
- description: String
+ addResource(resource :Resource)
+ deleteResource(resource :Resource) : void
+ getDateRange() : DateRange
+ getDescription() : String
+ getParty() : Party
+ getResources() : List<Resource>
+ setDateRange(dateRange :DateRange)
+ setDescription(desc :String)
+ setParty(party :Party)
0..1
0..*
«interface»
Party
+ getAddress() : Address
+ getPhoneNumber(phone :PhoneNumber)
+ setAddress(addr :Address)
+ setPhoneNumber(phone :PhoneNumber)
Reservation
- quotedPrice: Money
- resNumber: ReservationNumber
- status: ReservationStatus
+ checkedIn()
+ confirmed()
+ getQuotedPrice() : Money
+ getReservationNumber() : ReservationNumber
+ setQuotedPrice(price :Money) : void
+ setReservationNumber(resNum :ReservationNumber)
GenericParty
- address: Address
- phoneNumber: PhoneNumber
+ getAddress() : Address
+ getPhoneNumber() : PhoneNumber
+ setAddress(addr :Address)
+ setPhoneNumber(phone :PhoneNumber)
Company
- companyName: String
Person
- firstName: String
- lastName: String
+ getFirstName() : String
+ getLastName() : String
+ setFirstName(firstName :String)
+ setlastName(lastName :String) : void
Customer
+ addReservation(res :Reservation)

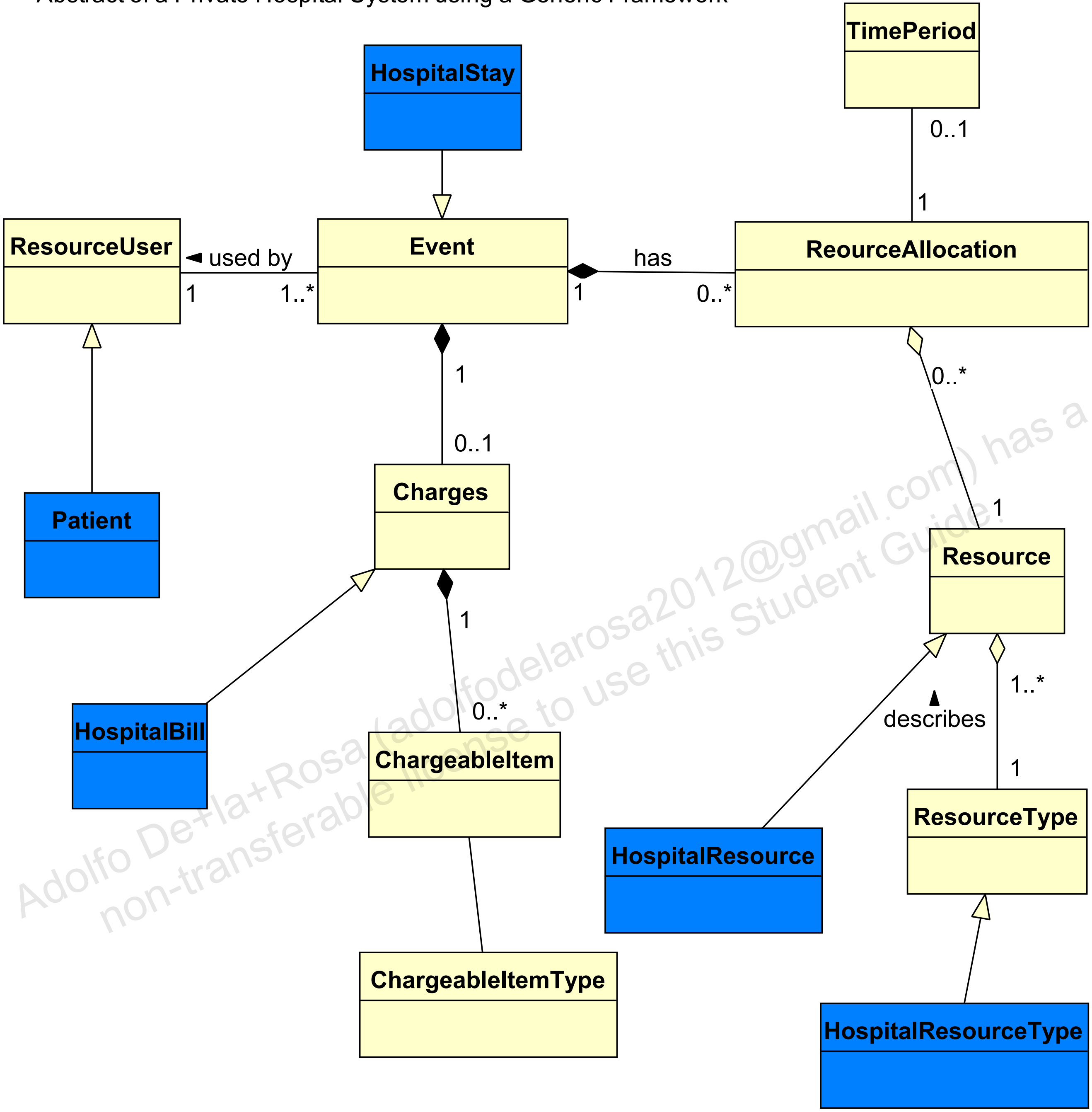
Note 1: This is an abstract of the attributes and methods.
Note 2: The framework is shown in yellow, and the Hotel specific classes are shown in blue.
Note 3: We could have created a more generic version which would have allowed multiple date ranges and multiple parties. The generic version would also have allowed replacing DateRange with TimeRange.

Activity 2: Identifying Potential Frameworks



Note 1: This is one possible solution.
Note 2: In a Private Hospital System, you may need multiple resources for different TimeRanges, for example: your room, the operating theatre, an intensive care nurse.
Note 3: Some resources, such as medication, may not need to be allocated for a TImeRange. Therefore, the TimeRange is optional.
Note 4: In some cases, it might be useful to assocate ChargeableItem with ResourceAllocation.

Abstract of a Private Hospital System using a Generic Framework



Note 1: This is one possible solution.
Note 2: The Framework classes are shown in Yellow, and the specific Private Hospital classes are shown in Blue.
Note 3: In a Private Hospital System, you may need multiple resources for different TimeRanges, for example: your room, the operating theatre, an intensive care nurse.
Note 4: Some resources, such as medication, may not need to be allocated for a TImeRange. Therefore, the TimeRange is optional.
Note 5: In some cases, it might be useful to assocate ChargeableItem with ResourceAllocation.