



Object Oriented Design

CSE3324 - Distributed Client
Server Programming



Goals

- To learn about the design phase of the development process (Chapter 12 of *Big Java*)
- To learn how to discover classes and member functions
- Design documentation expectations for the group project



Software Development

- Many software engineers break the development process down into the following phases:
 - ☐ Analysis
 - ☐ Design
 - ☐ Implementation
 - ☐ Testing
 - ☐ Deployment

Waterfall Model

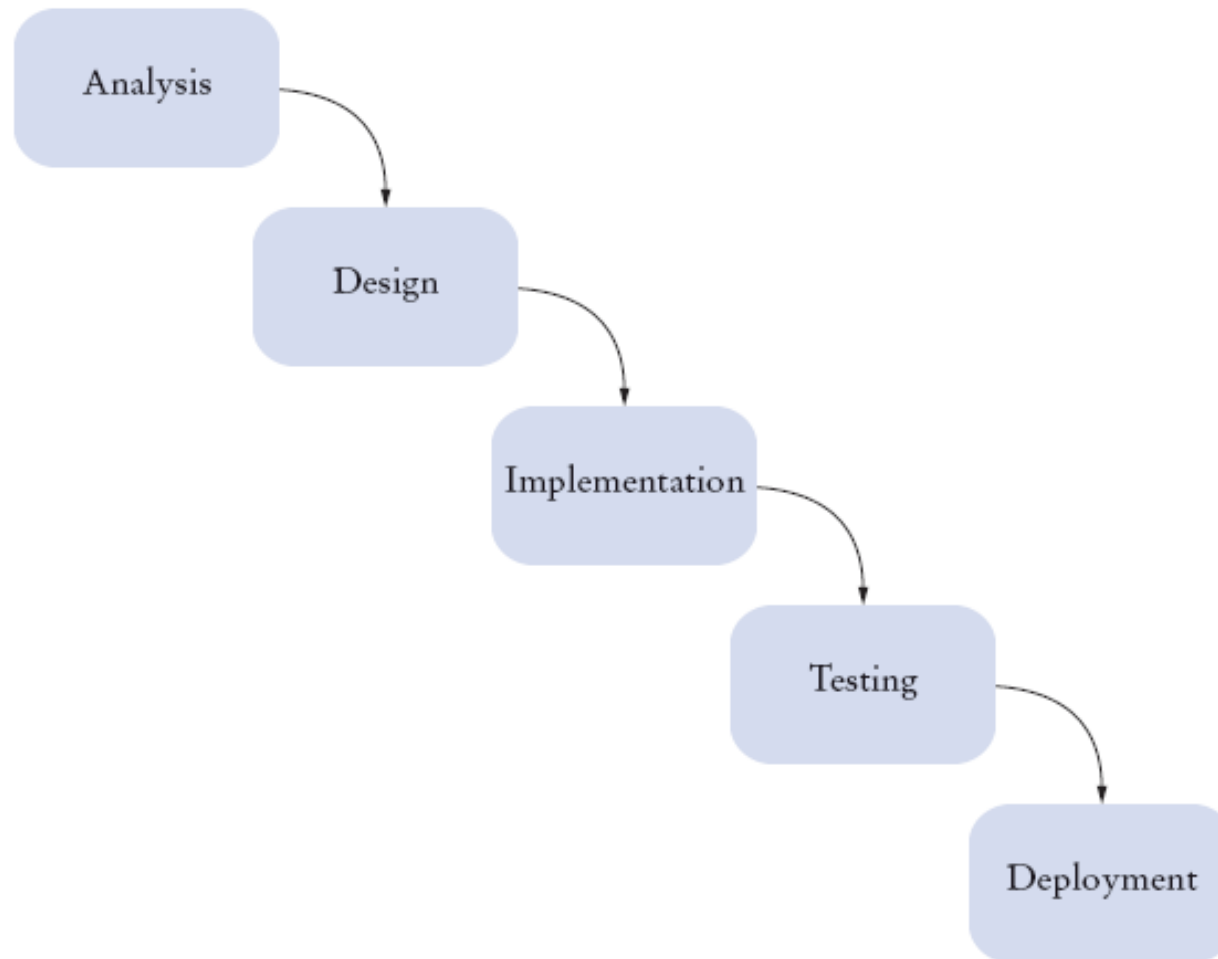


Figure 1 The Waterfall Model



Analysis (aka Requirements Analysis)

- Decide what the project is supposed to accomplish (you do not think about how the program will accomplish its tasks)
- The output of the analysis phase is a requirements document that
 - describes in complete detail what the program will be able to do
 - sets performance criteria - how many inputs the program must be able to handle in what time, or what its maximum memory and disk storage requirements are



Specifying Functional Requirements

- For each scenario you must specify
 - Validity checks on the inputs
 - Exact sequence of operations
 - Responses to abnormal situations
 - Relationship of outputs to inputs, including
 - Input/output sequences
 - Formulas for input to output conversion



Design Phase

- Now we will focus on taking the results of analysis and bridging the gap to implementation
- Develop a plan for the implementation of the system
- Discover the structures that underlie the problem to be solved




Object Oriented Design

- Discover classes.
- Determine the responsibilities of each class.
- Describe the relationships between the classes.
- Output: a description of the classes and member functions with diagrams that show relationships among the classes



Discovering Classes

- A simple rule for finding classes is to look for nouns in the task description.




Joe's Automotive Shop services foreign cars, and specializes in servicing cars made by Mercedes, Porsche, and BMW.

Which is a candidate class?

A. Joe

B. Car

C. Mercedes



When a customer brings a car to the shop, the manager gets the customer's name, address, and telephone number.

Which is a candidate class?

A. Customer's Name

B. Manager

C. Customer



Responsibilities

- Once a set of classes had been identified, you need to define the behavior (member functions) for each classes.
- A simple rule for finding member functions is to look for verbs in the task description, and then match the verbs to the appropriate objects.



Responsibilities

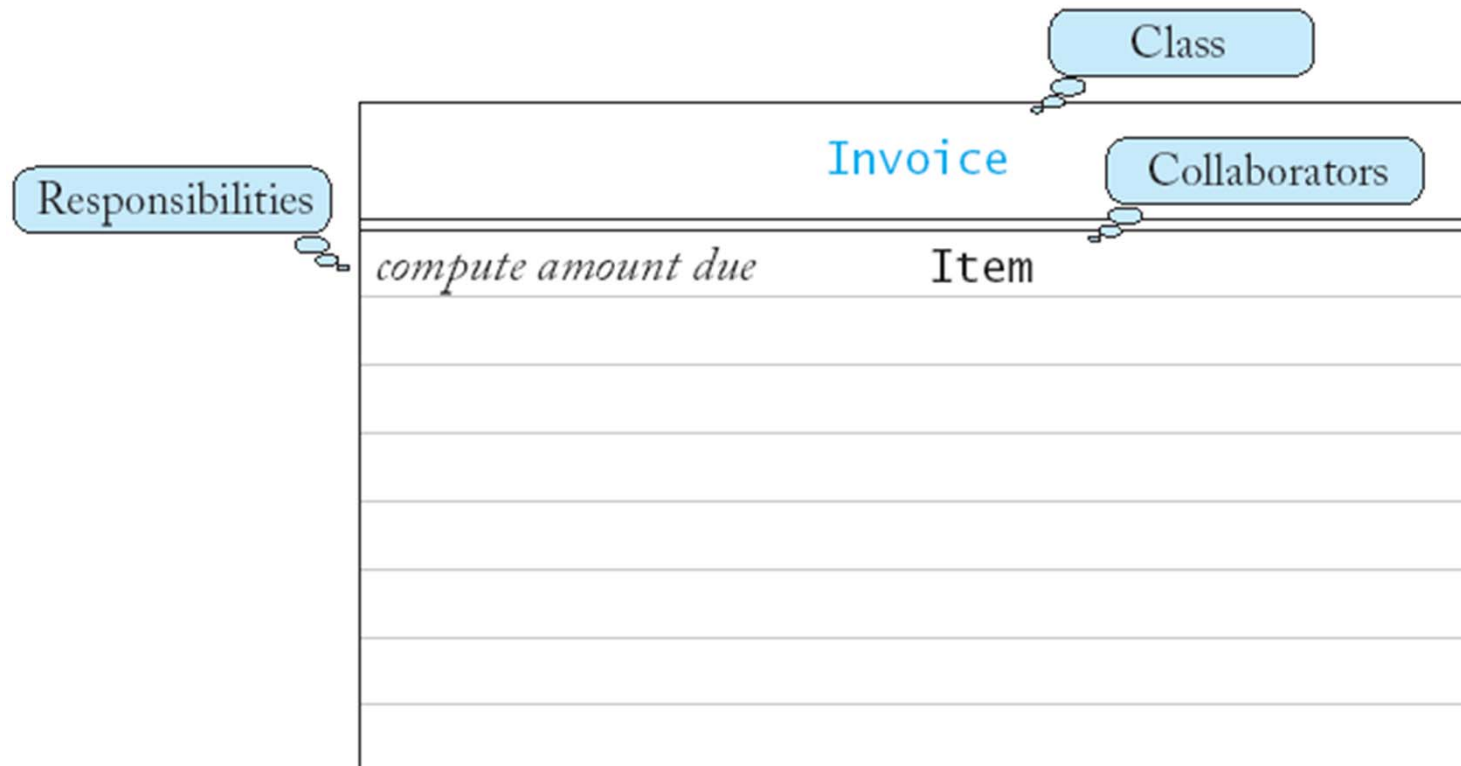
- To create an invoice we need to compute the amount due.
 - ☐ Do customers compute the amount due?
 - ☐ Do invoices total up the amount due?
 - ☐ Do items total themselves up?



CRC Cards

- In the CRC card method, you create an index card for each class.
- CRC stands for "classes", "responsibilities", "collaborators."
- As you think about the verbs in the task description, pick the card of the class that you think should be responsible, and write that responsibility on the left-hand side of the card.
- For each responsibility, you record on the right-hand side of the card which other classes (the collaborators) are needed to fulfill it.

Example CRC Card





CRC Cards

- As you list responsibilities and collaborators, think about the member functions the collaborators might need to help with the task.
- Example: To compute the total, the invoice needs to ask each item about its total price.
 - Therefore the Item class is a collaborator.
 - Does the Item class have a "get total price" member function? If not, add one to its card.



CRC Cards

- Many people find it helpful to group cards on a table so that the collaborators are close to each other, and to simulate tasks by moving tokens (such as a coin) from one card to the next to indicate which object is currently active.
- Responsibilities that you list on the CRC card are on a high level, and may need two or more member functions for carrying out.
- The CRC method is an informal process:
 - Be creative and discover the classes and their responsibilities.
 - Cross out, move, or split responsibilities.
 - Rip cards up if they become too messy.



What makes a good class?

- A class should represent a single concept.
 - ☐ Point
 - ☐ Circle
 - ☐ Time
 - ☐ Product
 - ☐ Employee
- Class names should be nouns (function names should be verbs).



What makes a bad class?

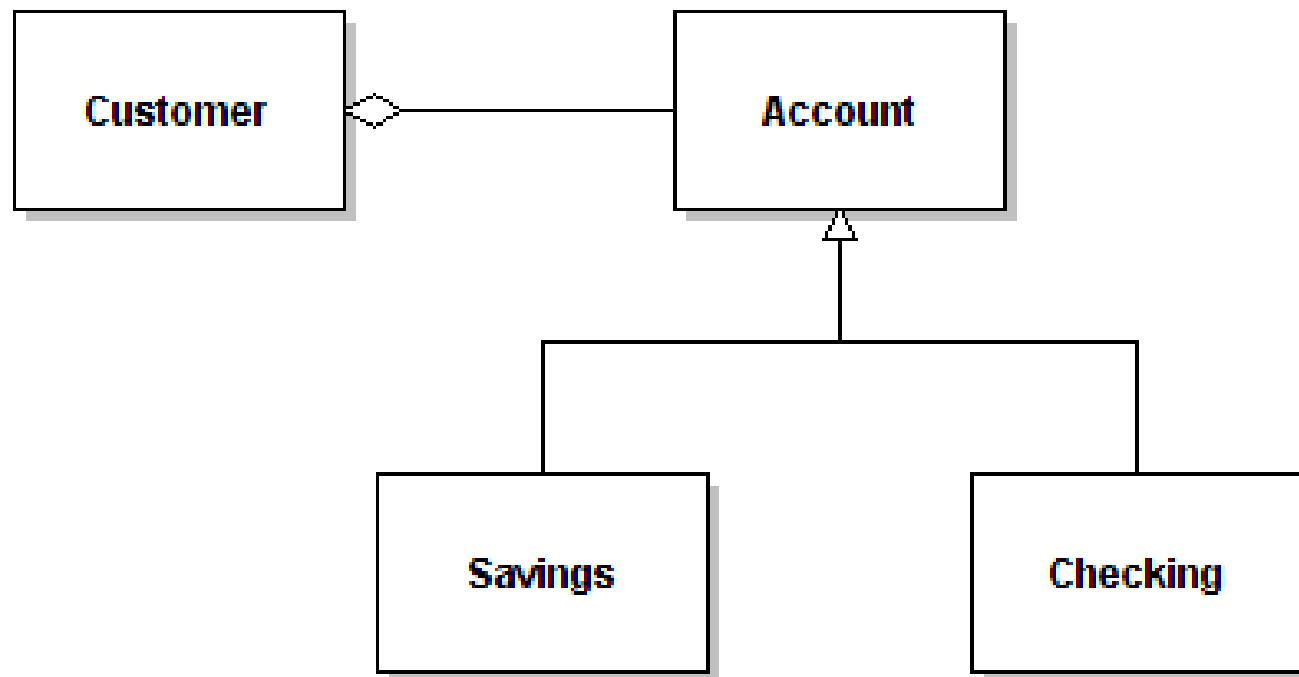
- If you can't tell from the class name what an object of the class is supposed to do, then you are probably not on the right track.
 - Example: What does an instance of a PaycheckProgram class do?
- Do not turn an action into a class.
 - Example: ComputePaycheck is a function, not a class.



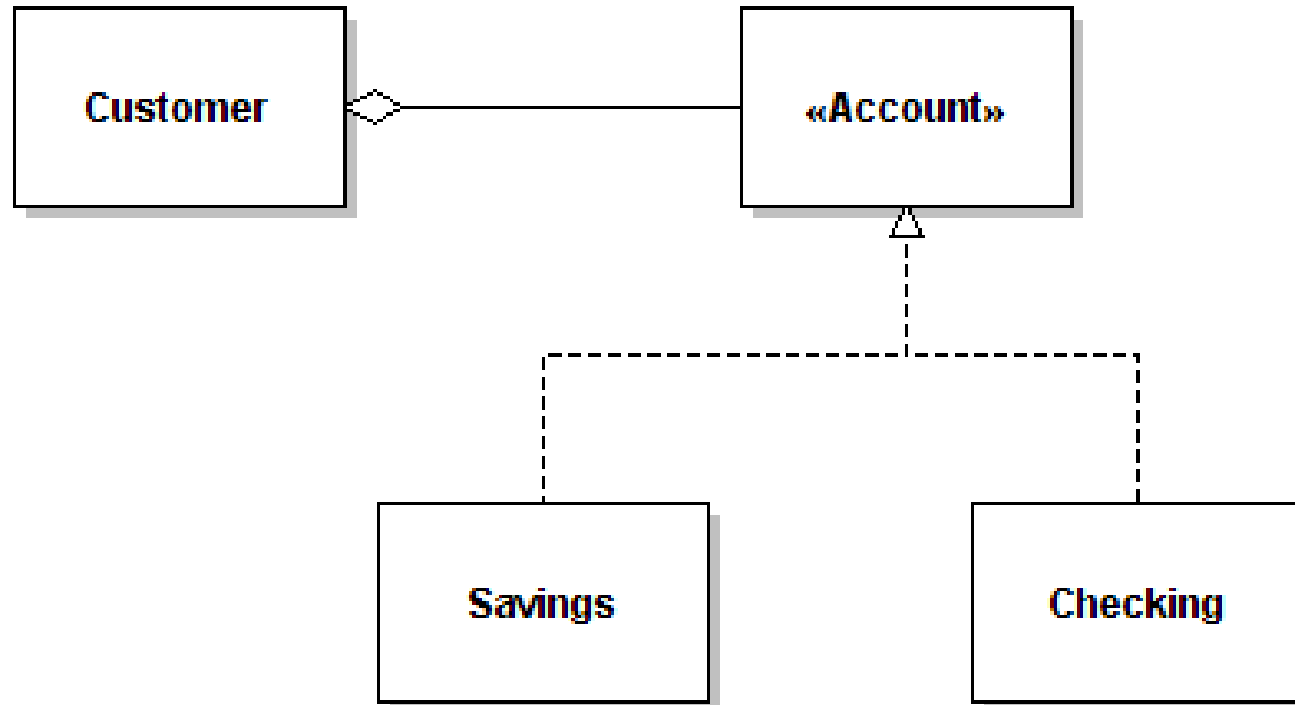
Relationships between Classes

- When designing a program, it is useful to document the relationships between classes.
 - For classes with common behavior, you can save effort by placing the common behavior in a base class.
 - For classes that are not related to each other, you can assign different programmers to implement each of them.
- Types of relationships
 - Inheritance / Implementing an Interface
 - Aggregation (Composition)

Relationships (Aggregation and Inheritance)



Relationships (Aggregation and Interfaces)

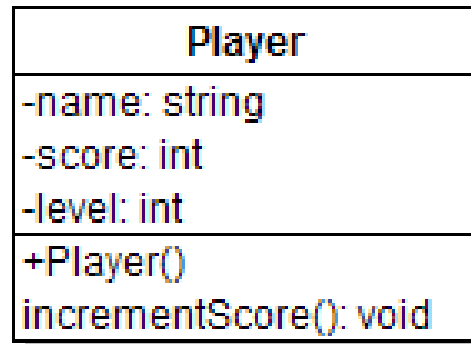


Detailed Class Diagrams

Game
-gamePlayer: Player -aClock: Clock -aTime: Time
+Game() +playGame(): void +readPlayerInfo(): void +playRound(): void

Player
-name: string -score: int -level: int
+Player() +getName(): string +getScore(): int +getLevel(): int +setName(n: string): void +setLevel(lev: int): void +incrementScore(): void

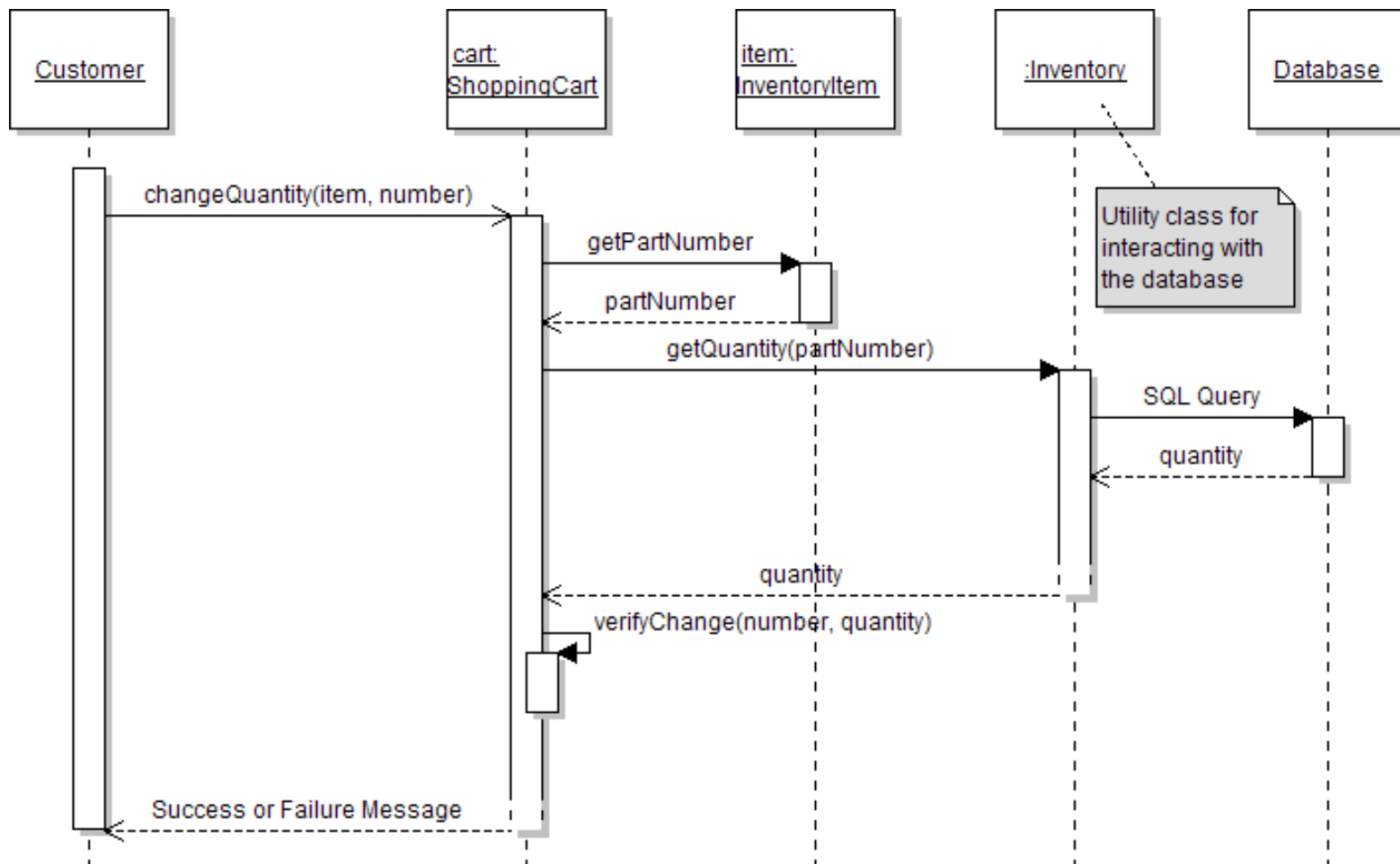
Detailed Class Diagrams



- For this project's design document:
 - Do not include setters and getters for the attributes
 - Give a short description for each method shown
 - Examples:
 - `Player()`: Default constructor. Sets name to an empty string. Sets score and level to 0.
 - `incrementScore()`: Increments score by the value held by level.

Sequence Diagrams

- Shows how objects interact to fulfill the functionality described in a scenario





Practice

- Draw a sequence diagram to illustrate the normal operation of the use case for “Change DVDs” for a DVD player that involves four objects: user, controller, motor and tray.
- Your sequence should describe the scenario in which the user ejects the current disk, inserts a new one, and selects play again.
- Assume the DVD player software has to send signals to the motor (to tell it when to spin) and to the tray (to open and close), and in both cases needs to await confirmation from the device before doing anything else.



Documenting Design for the Group Project

■ Concentrate on the "Model"

- Classes needed to implement the program regardless of user interface
- Ecommerce example: Customer, Manager, Shopping Cart, Inventory, Items, Order, etc.
- These classes will typically be used by Servlets or PHP pages



Documenting Design for the Group Project

- Overview class diagram showing relationships (inheritance, aggregation)
- Detailed class diagrams for each of the classes in the model
 - do not include setters and getters
 - include a short description of each method beneath the class diagram
- Sequence diagrams for four usage scenarios other than login and logout (examples: checkout, create account)



Your group's design document must contain ...

- A title page and table of contents
- Section 1: Overview class diagram showing relationships
- Section 2: Detailed class diagrams for each of the model classes
 - A subsection for each class
- Section 3: Sequence diagrams for four scenarios
 - A subsection for each scenario
- Proposed database layout (Appendix A)
- Current task and role assignments including presentations (Appendix B)



Title Page

■ Must identify the

- ☐ Project's name
- ☐ Document's name and version number
 - Design Document, version 1
 - Design Document, version 2
- ☐ Date
- ☐ Group members
- ☐ Group number
- ☐ Lab instructor's name



Database Design – Appendix A

- Show table names, attributes within each table, primary and foreign keys
- Similar to an ER diagram, but can be expressed in just text
- This section will not be graded for correctness is version 1



Database Design – Example

Appendix A – Database Design

("*" = Primary Key, "#" = Foreign Key)

A.1 USER Table

- *USER_ID
- NAME
- EMAIL
- PASSWORD
- B_STREET
- B_CITY
- B_STATE
- B_ZIP

A.2 INVENTORY Table

- *ITEM_ID
- NAME
- DESC
- PRICE
- QTY

A.3 ORDERS Table

- *ORDER_ID
- #USER_ID
- SHIPPING
- ORDER_TOTAL
- S_STREET
- S_CITY
- S_STATE
- S_ZIP

A.4 ORDER_ITEMS Table

- #ORDER_ID
- #ITEM_ID
- PRICE
- QTY
- ITEM_TOTAL



References

- *Big Java*, Chapter 12
- *Big C++*, Appendix E: UML Class Diagrams
- Violet UML Editor
 - <http://www.horstmann.com/violet/>
- UML Sequence Diagrams
 - <http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>
 - <http://www.ibm.com/developerworks/rational/library/3101.html>
 - <http://www.developer.com/design/article.php/3080941>
 - <http://www.websequencediagrams.com/>