**Knowledge and Thoery: Object Oriented Principles and Design**

**Part 1 – Introduction:**

***Step 1 – Handle Ambiguity:*** Questions are purposely given with a lot of ambiguity to test if you will just build and make assumptions or if you will make efforts to gain clarity. Ask who, what, where, when, why and how.

***Step 2 – Define the Core Objects for Classes:*** For example, suppose we are asked to do the object-oriented design for a restaurant. Our core objects might be things like Table, Guest, Order, Meal, Employee, and Server. Find out what is the minimum information we need to run this restaurant based on the requirements.

* For example, does it matter which tables are available? Do people have preferences for which table they get? Can we just have an integer variable called “tables available” and then put incoming guests anywhere randomly? Do we need to keep track of which guests are at which table or can we just iterate over the tables periodically to ensure customers there are OK.
* What about bills? Do customers just need a double value for the amount they owe, or do they need an entire price breakdown? This would mean you would need a static method called “get bill” that takes in an order and returns a string of the price breakdown.

You don’t want to over complicate the design but you also want to meet requirements.

***Step 3 – Analyze Relationships for Class Attributes:***Do any objects inherit from any others or have composition? Are relationships many-to-many or one-to-many?For example, in the restaurant question, we may come up with the following design:

* Party should have an array of Guests.
* Server and Host inherit from Employee.
* Each Table has one Party, but each Party may have multiple Tables.
* There is one Host for the Restaurant.

Make sure you verify all your relationships. For example, ask “does each party have multiple tables?”.

***Step 4 – Think about relevant information for class attributes:*** For example, what do we need to know about a meal? Perhaps what the meal consists of and also what is the price of the meal. These are going to be the attributes of the meal class.

***Step 5 – Investigate Actions for Methods****:*  For example, a Party walks into the Restaurant, and a Guest requests a Table from the Host. The Host looks up the Reservation and, if it exists, assigns the Party to a Table. Otherwise, the Party is added to the end of the list. When a Party leaves, the

Table is freed and assigned to a new Party in the list.

***Step 6 – Go Over all the Use Cases****:* For example, have a customer enter, order food, be served, billed, pay, leave, etc. See if there is any required information that your missing. For example, you might

**Part 2 – Relevant Design Patterns:**

***Singleton Design Pattern***: The Singleton pattern ensures that a class has only one instance that any object can access statically. It can be useful in cases where you have a "global"object with exactly one instance.

|  |
| --- |
| **public** **class** Restaurant {  **private** **static** Restaurant *restaurant* = **null**; //single instance  **public** **static** Restaurant getlnstance() {  **if** (*restaurant* == **null**) {  *restaurant* = **new** Restaurant();  }  **return** *restaurant*;  }  } |

***Factory Design Pattern***: The Factory Method offers a way of creating an instance of a class, with its subclasses deciding which class to instantiate.

|  |
| --- |
| **public abstract** **class** CardGame {  **public** **static** CardGame initializeCardGame(GameType type) {  **if** (type == GameType.Poker) {  **return** **new** PokerGame();  } **else** **if** (type == GameType.BlackDack) {  **return** **new** BlackJackGame();  }  **return** **null**;  }  } |

**Part 3 – Design an Online Book Reader System:**

|  |  |  |
| --- | --- | --- |
| **Class** | **Attributes** | **Methods** |
| **Online Book Reader System** | * Contains a library * Contains a hash table of users with their corresponding user id’s. * Contains a display. | * Contains facilities to add, find and remove users from the book reader system. * Decides whether a user can view a book based on the user’s subscription. |
| **Library** | * Contains a collection of books in a hash map. * Each record in the hashmap maps a book with its corresponding book id (a special ID made to make it easy to retrieve books). | * Contains facilities to add, find and remove books from the library. |
| **Book** | * Contains a book ID, title, author, published date, contents, etc. |  |
| **User** | * Contains user id, password, membership type, name, address, etc. | * Contains facilities to change password, renew membership, etc. |
| **Display** | * Contains a book that is currently being displayed. | * Provides facilities to display a page, change the page, scroll up, etc. All the features of a PDF reader. |

**Final Notes:** Your ability to come up with “good” Object Oriented Designs is based on

1. The very basic knowledge of object oriented program (i.e. all the patterns and characteristics used above).
2. Your knowledge of the system you are designing (i.e. you will make a better design for a poker game if you are a poker player because you know what are the key use cases, what are the things that are being done a lot (i.e. card shuffling) etc.
3. Whether your preferences match the interviewers preferences. There are so many different ways to design something. If your interviewer is not familiar with your design pattern, you get marked as “overcomplicates designs, does not produce clarity”. If your code does not encompass design patterns that your interviewers deems should be used, you get marked as “does not essential OO principles, design patterns, lacks experience, bad design”, etc. The best way to hack the game is to see what their guidelines are for good code and good designs.