Design Patterns Results for Christine En-Tse Cheng

Submitted Dec 17 at 3:56p.m.

Unanswered

Question 1

0 / 1 pts

What is the primary purpose of design patterns in software development?

- ➔ To provide reusable solutions to common design problems
- To make code look more complex
- To help make software easier to understand
- To decrease the number of classes in a program

Unanswered

Question 2

0 / 1 pts

Which of the following is **NOT** one of the three categories of design patterns?

- Behavioural
- Structural
- Creational
- ➔ Functional

Unanswered

Question 3

0 / 1 pts

Match each description of a design pattern category to its name.

- ✗ **Patterns that manage object instantiation.** [ ]
  - Creational
- ✗ **Patterns that define how objects interact and communicate** [ ]
  - Behavioural
- ✗ **Patterns that describe how classes and objects are composed to form larger structures** [ ]
  - Structural

Other Incorrect Match Options:
- Functional

Unanswered

Question 4

0 / 1 pts

Match each design pattern to its category.

- ✗ **Observer** [ ]

Behavioural

Façade

Structural

Adapter

Structural

Factory

Creational

Builder

Creational

Strategy

Behavioural

Other Incorrect Match Options:

- Functional

Unanswered

Question 5

0 / 1 pts

What problem does the Simple Factory design pattern aim to solve?

- Reducing the number of classes in an application.
- → Encapsulating object creation logic.
- Ensuring thread safety in object creation.
- Managing multiple instances of a single class.

Unanswered

Question 6

0 / 1 pts

When would using a Builder pattern be most appropriate?

- When you want only one instance of a class
- When you want objects to be notified of changes in another object
- When you want to provide multiple implementations of the same functionality
- → When creating an object requires complex initialization or many optional parameters

Unanswered

Question 7

0 / 1 pts

In the Strategy pattern, how are different algorithms typically implemented?

- → As subclasses of a common interface

- As a single, switch-based method in the main class
- As individual objects passed to the main class
- As static methods in a utility class

Unanswered

Question 8

0 / 1 pts

When would using an Observer pattern be most appropriate?

- ➜ When an object's state changes and all dependent objects should automatically update
- When you want to create a series of related algorithms
- When you want to allow an object to have various configurations without subclassing
- When creating objects with a common interface but different implementations

Unanswered

Question 9

0 / 1 pts

Which of the following design patterns is most clearly demonstrated by this `RoomScheduler` class?

```
public class RoomScheduler {

  private BookingInfo info;
  private CalendarInteracter ci;
  private RoomBooker booker;
  private BookingGateway gateway;

  public RoomScheduler (BookingInfo info, CalendarInteracter ci,
      RoomBooker booker, BookingGateway gateway) {
    this.info = info;
    this.ci = ci;
    this.booker = booker;
    this.gateway = gateway;
  }

  public void checkAvailability () {
    ci.checkAvailability(info);
  }

  public void makeBooking () {
    booker.makeBooking(info);
  }

  public void recordBooking () {
    gateway.recordBooking(info);
  }
}
```

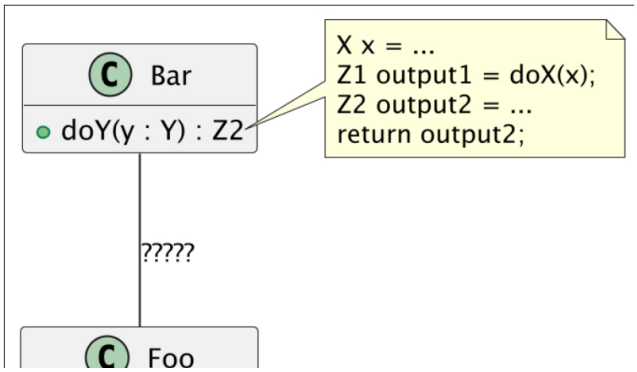
- Abstract Factory
- Builder
- ➜ Façade
- Simple Factory

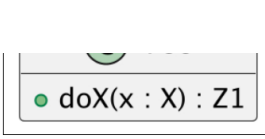
Unanswered

Question 10

0 / 1 pts

Which Design Pattern is described by this UML Class Diagram? The names have been changed and details omitted to obscure which pattern it is.





- Strategy
- Façade
- Adapter (with delegation)
- Adapter (with inheritance)