M2006C3MII

5:58 PM

Software Patterns History

In general, the concept of *patterns* was originally articulated by Christopher Alexander and colleagues in the late 1970s . They had 253 patterns.

Somewhat later, it was recognized that there are many similarities between software design and architectural design:

- Both are creative processes that unfold within a large design space, which comprises all possible designs.
- The resulting design must satisfy the customer's needs.
- The resulting design must be feasible to engineer.
- The designers must satisfy many competing constraints and requirements.
- The designers must seek certain intrinsic yet unquantifiable qualities, such as elegance and extensibility.

Pioneering work in this area was done by the so-called "Gang of Four". [See their seminal text on the subject: *Design Patterns (Elements of Reusable Object-Oriented Software)*, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1995]

Type here to search 28°C ENG 17:57 03-08-2023

M2006C3MII

5:59 PM

Design Patterns became a best-seller, is now regarded as a "ground-breaking" work, and the rest is history: Design patterns have been accepted as a very useful tool by the software development community at large.

Many additional patterns have been identified and described, and more appear all the time.

In a very small way, you are familiar with a number of "mini design patterns". A very small pattern in a programming language is sometimes called an *idiom*. Can you think of any examples?

Patterns originated as an architectural concept by Christopher Alexander (1977/79). In 1987, Kent Beck and Ward Cunningham began experimenting with the idea of applying patterns to programming and presented their results at the OOPSLA conference that year. In the following years, Beck, Cunningham and others followed up on this work.

Design patterns gained popularity in computer science after the book *Design Patterns: Elements of Reusable Object-Oriented Software* was published in 1994 by the so-called "Gang of Four" (Gamma et al.).

Type here to search 28°C ENG 17:57 03-08-2023

M2006C3MII

5:59 PM

Introduction to Design Patterns:

Instead of defining what is design pattern lets define what we mean by design and what we mean by pattern. Design is blue print or sketch of something so it can be defined as creation of something in mind. Moving to pattern, we can define it as guideline, or something that repeats. Now the definition of design pattern becomes creating something in mind that repeats or in other words capturing design ideas as a "pattern" to the problems.

Some problem patterns happen over and over again in a given context and *Design Pattern* provides a core of the solution in such a way that you can use the core solution every time but implementation should and may vary and the main reason behind that is we have the core solution and not the exact solution. We can discuss an example here about database normalization. Normalization is a pattern (Core solution to database design) but what level of normalization you need (exact solution) depends on your requirement and context.

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced

Type here to search

28°C ENG 17:57 03-08-2023

M2006C3MII

5:59 PM

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time. Each design pattern describes a problem which occurs over and over again in our software development environment, and then describes the core of the problem.

Design Patterns are the best solutions for the re-occurring problems in the application programming environment.

- Nearly a universal standard.
- Responsible for design pattern analysis in other areas, including GUIs.
- Mainly used in Object Oriented programming.

In software development process, a design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be

Type here to search

28°C ENG 17:57 03-08-2023

M2006C3MII

5:59 PM

- Mainly used in Object Oriented programming.

In software development process, a design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Summary:

- Designing object oriented software is hard, and designing reusable object oriented software is even harder.

Type here to search 28°C ENG 17:57 03-08-2023

M2006C3MII

5:59 PM

- It takes a long time for novices to learn what good object oriented design is all about. Experiences designers know obviously know something that inexperienced ones don't.
- Recording experience in designing object oriented software is design pattern.
- Expert designers find recurring pattern of classes and communicating objects in many object oriented system and can apply them to solve design problems without having to rediscover them.
- Design patterns make it easier to reuse successful design and architect uses.

Que 1: What is a Design Pattern? Explain the format for describing a design pattern.

Ans:

- Design pattern describes a problem which occurs over and over again in environment, and then describes the core of the problem
- Design Patterns are the best solutions for the re-occurring problems in the application

Type here to search 28°C ENG 17:57 03-08-2023

M2006C3MII

5:59 PM

Que 1: What is a Design Pattern? Explain the format for describing a design pattern.

Ans:

- Design pattern describes a problem which occurs over and over again in environment, and then describes the core of the problem
- Design Patterns are the best solutions for the re-occurring problems in the application programming environment.
  - Nearly a universal standard.
  - Responsible for design pattern analysis in other areas, including GUIs.
  - Mainly used in Object Oriented programming.

Description of design patterns:

- Graphical notations, while important and useful, aren't sufficient.
- They capture the end product of the design process as relationships between classes and

Type here to search

28°C ENG 17:57 03-08-2023

M2006C3MII

5:59 PM

Description of design patterns:

- Graphical notations, while important and useful, aren't sufficient.
- They capture the end product of the design process as relationships between classes and objects.
- By using a consistent format we describe the design pattern.
- Each pattern is divided into sections according to the following template.

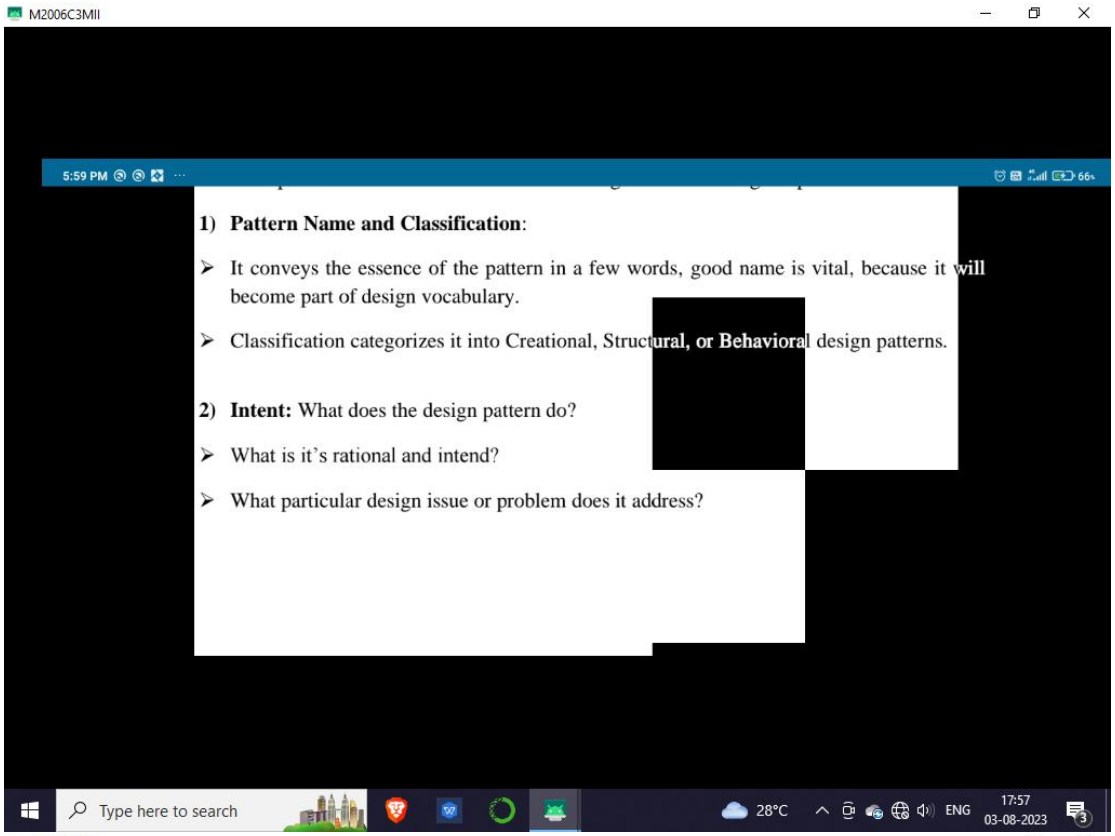
1) **Pattern Name and Classification**:

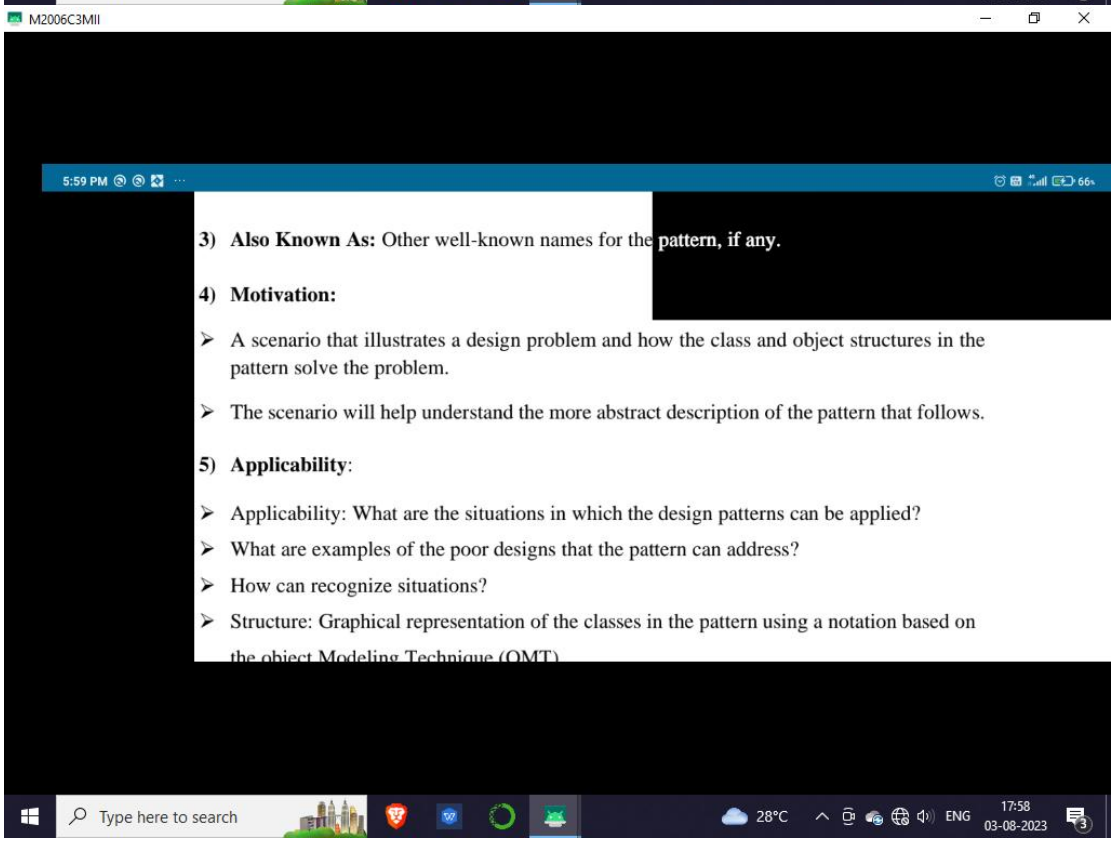
- It conveys the essence of the pattern in a few words, good name is vital, because it will become part of design vocabulary.
- Classification categorizes it into Creational, Structural, or Behavioral design patterns.

2) **Intent:** What does the design pattern do?

Type here to search

28°C ENG 17:57 03-08-2023


M2006C3MII
5:59 PM
1) Pattern Name and Classification:
➢ It conveys the essence of the pattern in a few words, good name is vital, because it will become part of design vocabulary.
➢ Classification categorizes it into Creational, Structural, or Behavioral design patterns.
2) Intent: What does the design pattern do?
➢ What is it's rational and intend?
➢ What particular design issue or problem does it address?
Type here to search
28°C
ENG
17:57
03-08-2023


M2006C3MII
5:59 PM
3) Also Known As: Other well-known names for the pattern, if any.
4) Motivation:
➢ A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem.
➢ The scenario will help understand the more abstract description of the pattern that follows.
5) Applicability:
➢ Applicability: What are the situations in which the design patterns can be applied?
➢ What are examples of the poor designs that the pattern can address?
➢ How can recognize situations?
➢ Structure: Graphical representation of the classes in the pattern using a notation based on the object Modeling Technique (OMT)
Type here to search
28°C
ENG
17:58
03-08-2023

M2006C3MII

5:59 PM

5) Applicability:

- Applicability: What are the situations in which the design patterns can be applied?
- What are examples of the poor designs that the pattern can address?
- How can recognize situations?
- Structure: Graphical representation of the classes in the pattern using a notation based on the object Modeling Technique (OMT).
- Participants: The classes and/or objects participating in the design pattern and their responsibilities.

6) Structure:

- Graphical representation of the classes in the pattern using a notation based on the object Modeling Technique(OMT).

Type here to search 28°C ENG 17:58 03-08-2023

M2006C3MII

6:00 PM

6) Structure:

- Graphical representation of the classes in the pattern using a notation based on the object Modeling Technique(OMT).
- It can also use interaction diagrams to illustrate sequences of requests and collaborations between objects.

7) Participants:

- The classes and/or objects participating in the design pattern and their responsibilities.

8) Collaborations:

- How the participants collaborate to carry out their responsibilities.

Type here to search 28°C ENG 17:58 03-08-2023

M2006C3MII

6:00 PM

9) Consequences:

- How does the pattern support its objectives?
- What are the trade-offs and result of using the pattern ?
- What aspect of the system structure does it let vary independently?

10) Implementation:

- What pitfalls, hints, or techniques should be aware of when implementing the pattern?
- Are there language-specific issues?

Type here to search 28°C ENG 17:58 03-08-2023

M2006C3MII

6:00 PM

10) Implementation:

- What pitfalls, hints, or techniques should be aware of when implementing the pattern?
- Are there language-specific issues?

11) Sample Code:

- Code fragments that illustrate how might implement the pattern in c++ or Smalltalk.

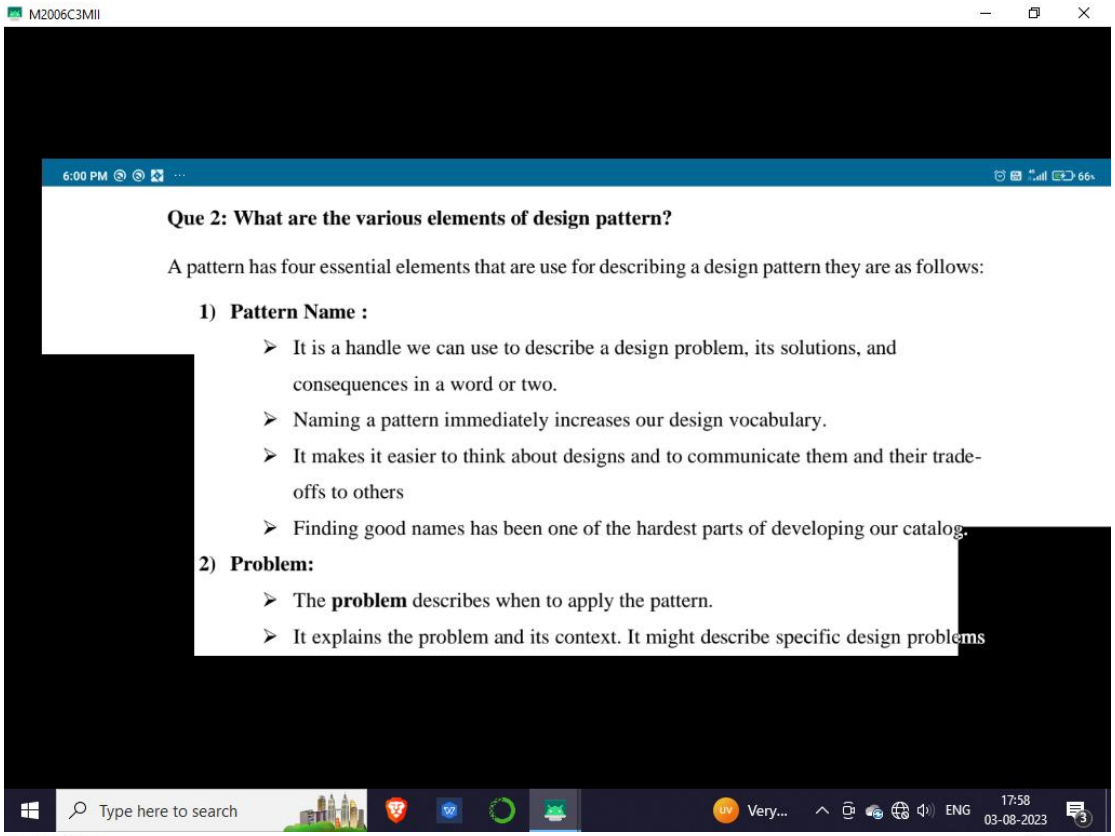
12) Known Uses:

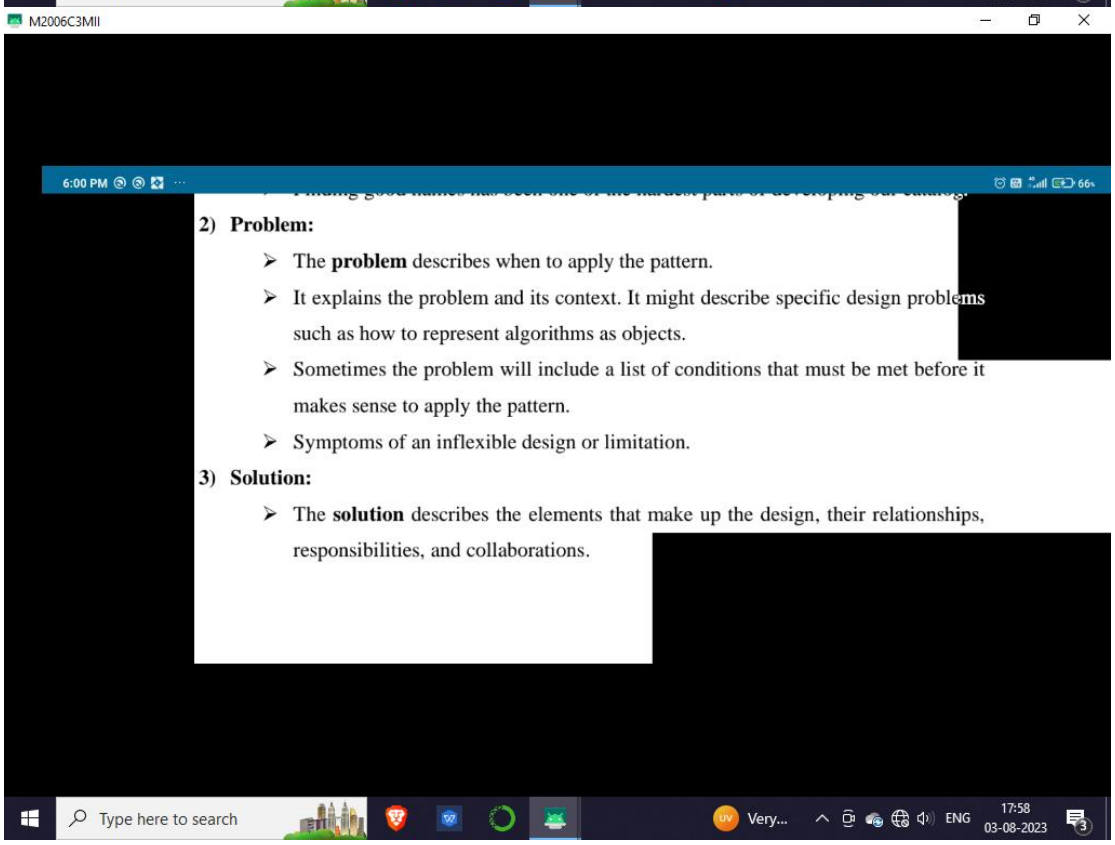
- Examples of the pattern found in real systems.

13) Related Patterns:

- What design patterns are closely related to this one? What are the implementation differences? With Which other patterns should this one be used?

Type here to search Very... ENG 17:58 03-08-2023


M2006C3MII
6:00 PM
Que 2: What are the various elements of design pattern?
A pattern has four essential elements that are use for describing a design pattern they are as follows:
1) Pattern Name :
➢ It is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.
➢ Naming a pattern immediately increases our design vocabulary.
➢ It makes it easier to think about designs and to communicate them and their trade-offs to others
➢ Finding good names has been one of the hardest parts of developing our catalog.
2) Problem:
➢ The problem describes when to apply the pattern.
➢ It explains the problem and its context. It might describe specific design problems
Type here to search
Very...
ENG
17:58
03-08-2023


M2006C3MII
6:00 PM
2) Problem:
➢ The problem describes when to apply the pattern.
➢ It explains the problem and its context. It might describe specific design problems such as how to represent algorithms as objects.
➢ Sometimes the problem will include a list of conditions that must be met before it makes sense to apply the pattern.
➢ Symptoms of an inflexible design or limitation.
3) Solution:
➢ The solution describes the elements that make up the design, their relationships, responsibilities, and collaborations.
Type here to search
Very...
ENG
17:58
03-08-2023

M2006C3MII

6:00 PM

- The solution doesn't describe a particular concrete design or implementation, because a pattern is like a template that can be applied in many different situations.

4) **Consequences:**

- The **consequences** are the results and trade-offs of applying the pattern.
- They are critical for evaluating design alternatives and for understanding the costs and benefits of applying the pattern.
- They may address language and implementation issues as well.
- The consequences of a pattern include its impact on a system's flexibility, extensibility, or portability.

- The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities.
- Each design pattern focuses on a particular object-oriented design problem or issue.

Type here to search

Very... ENG 17:58 03-08-2023

M2006C3MII

6:00 PM

- The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities.
- Each design pattern focuses on a particular object-oriented design problem or issue.
- It describes when it applies, whether it can be applied in view of other design constraints, and the consequences and trade-offs of its use.

Que 3: Explain the usages of design pattern.

Usage of Design Pattern:

Design Patterns have following usages in software development.

1) **Common platform for developers:**

Design patterns provide a standard terminology and are specific to particular scenario. patterns allow developers to communicate using well-known, well understood names for software

Type here to search

Very... ENG 17:58 03-08-2023

M2006C3MII

6:00 PM

Que 3: Explain the usages of design pattern.

Usage of Design Pattern:

Design Patterns have following usages in software development.

1) **Common platform for developers:**

Design patterns provide a standard terminology and are specific to particular scenario. patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

Type here to search

Very...

ENG

17:58
03-08-2023

M2006C3MII

6:00 PM

2) **Best practices for software development process:**

Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

3) **Design patterns can speed up the development process:**

Type here to search

Very...

ENG

17:58
03-08-2023

M2006C3MII

6:00 PM

3) **Design patterns can speed up the development process:**

Appropriate design pattern providing tested, proven development paradigms and thus software development process speed up. Design patterns helps to prevent delicate issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Que:4 Explain the catalog of design pattern along with its classification

The Catalog of Design Patterns:

The catalog beginning on contains 23 design patterns. Their names and intents are listed next to give you an overview.

1. **Abstract Factory**

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Type here to search

Very...

ENG

17:58

03-08-2023

M2006C3MII

6:00 PM

1. **Abstract Factory**

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

2. **Adapter**

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

3. **Bridge**

Decouple an abstraction from its implementation so that the two can vary independently.

4. **Builder**

Separate the construction of a complex object from its representation so that the same construction process can create different representations.

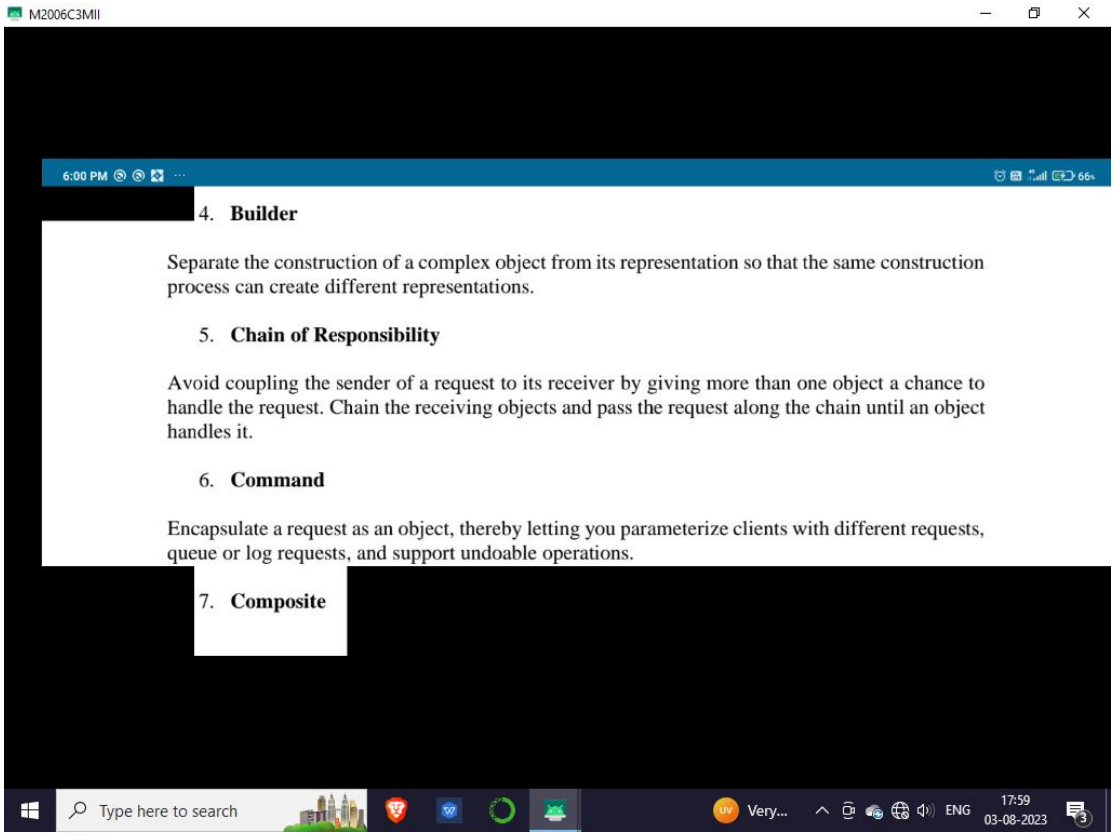
Type here to search

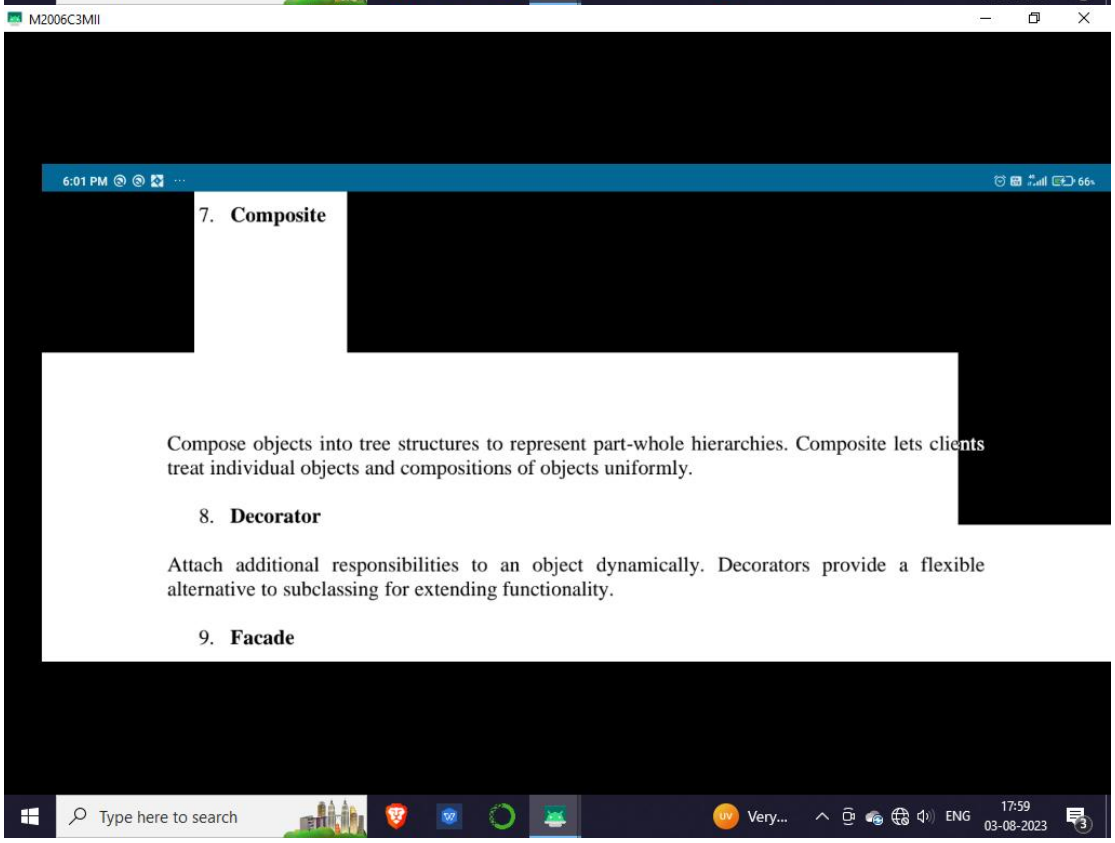
Very...

ENG

17:59

03-08-2023


M2006C3MII
6:00 PM
4. **Builder**
Separate the construction of a complex object from its representation so that the same construction process can create different representations.
5. **Chain of Responsibility**
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
6. **Command**
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
7. **Composite**
Type here to search
Very...
ENG
17:59
03-08-2023


M2006C3MII
6:01 PM
7. **Composite**
Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
8. **Decorator**
Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.
9. **Facade**
Type here to search
Very...
ENG
17:59
03-08-2023

M2006C3MII

6:01 PM

9. **Facade**

Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

10. **Factory Method**

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

11. **Flyweight**

Use sharing to support large numbers of fine-grained objects efficiently.

12. **Interpreter**

Given a language, define a represention for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Type here to search

Very...

ENG

17:59
03-08-2023

M2006C3MII

6:01 PM

12. **Interpreter**

Given a language, define a represention for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

13. **Iterator**

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

14. **Mediator**

Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

15. **Memento**

Type here to search

Very...

ENG

17:59
03-08-2023

M2006C3MII

6:01 PM

15. **Memento**

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

16. **Observer**

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

17. **Prototype**

Type here to search

Rain... ENG 17:59 03-08-2023

M2006C3MII

6:01 PM

17. **Prototype**

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

18. **Proxy**

Provide a surrogate or placeholder for another object to control access to it.

19. **Singleton**

Ensure a class only has one instance, and provide a global point of access to it.

Type here to search

Rain... ENG 17:59 03-08-2023

M2006C3MII

18. **Proxy**

Provide a surrogate or placeholder for another object to control access to it.

19. **Singleton**

Ensure a class only has one instance, and provide a global point of access to it.

20. **State**

Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

21. **Strategy**

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

22. **Template Method**

Type here to search Rain... ENG 17:59 03-08-2023

M2006C3MII

the algorithm vary independently from clients that use it.

22. **Template Method**

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

23. **Visitor**

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

Organizing the Catalog

Design patterns vary in their granularity and level of abstraction. Because there are many design

Type here to search Rain... ENG 17:59 03-08-2023

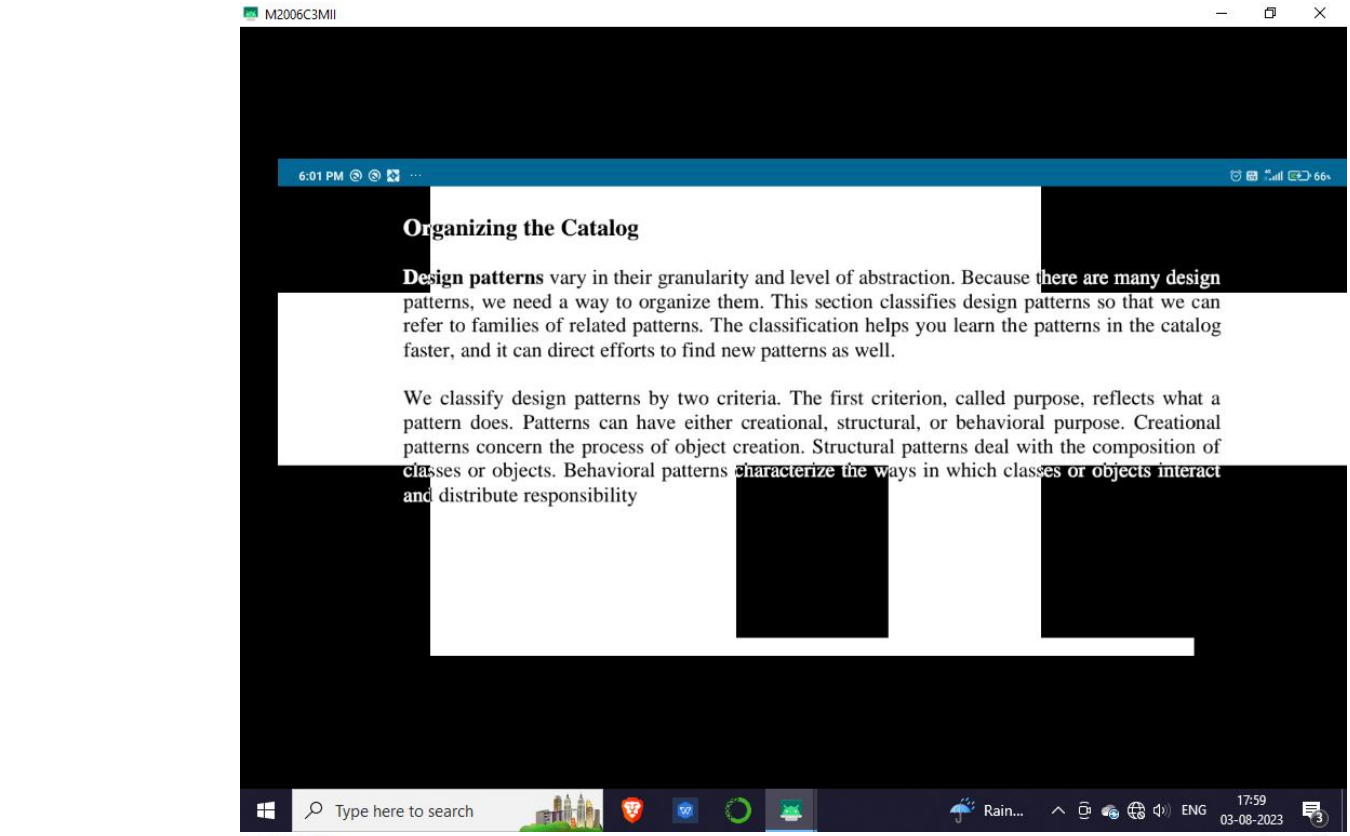
M2006C3MII

6:01 PM

Organizing the Catalog

Design patterns vary in their granularity and level of abstraction. Because there are many design patterns, we need a way to organize them. This section classifies design patterns so that we can refer to families of related patterns. The classification helps you learn the patterns in the catalog faster, and it can direct efforts to find new patterns as well.

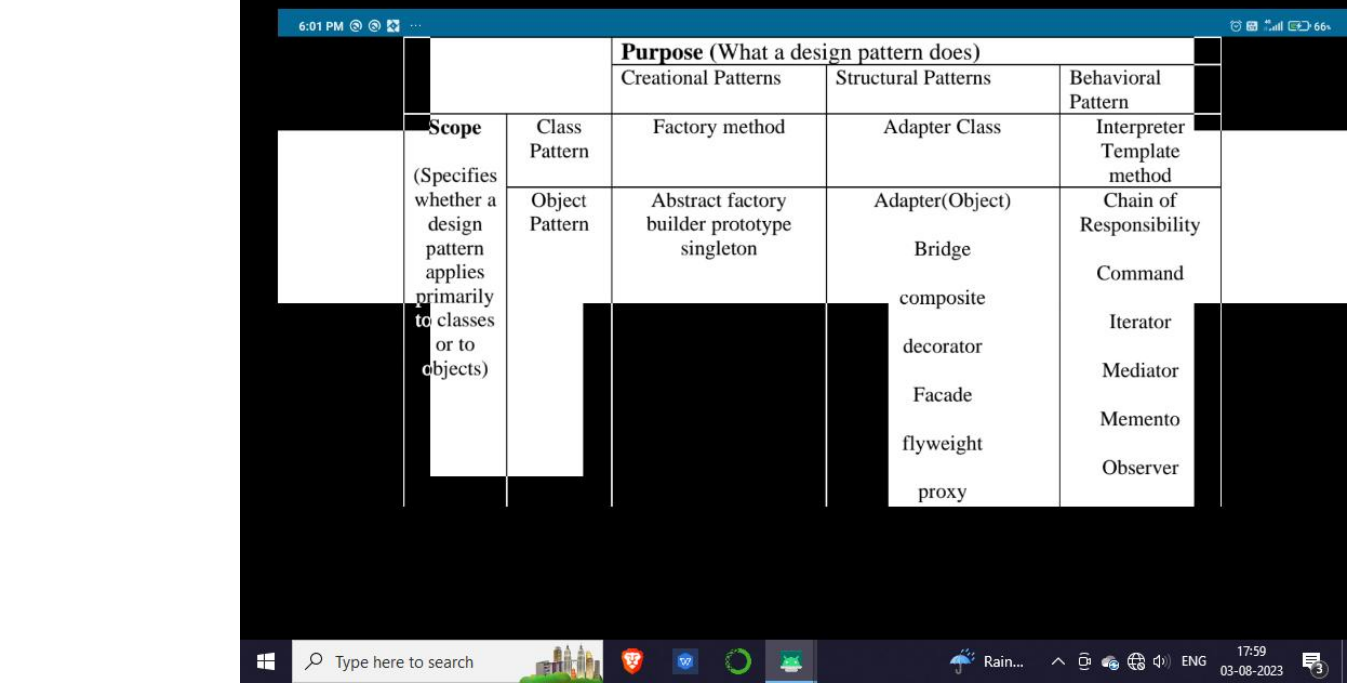
We classify design patterns by two criteria. The first criterion, called purpose, reflects what a pattern does. Patterns can have either creational, structural, or behavioral purpose. Creational patterns concern the process of object creation. Structural patterns deal with the composition of classes or objects. Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility



M2006C3MII

6:01 PM

		Purpose (What a design pattern does)		
		Creational Patterns	Structural Patterns	Behavioral Pattern
Scope (Specifies whether a design pattern applies primarily to classes or to objects)	Class Pattern	Factory method	Adapter Class	Interpreter Template method
	Object Pattern	Abstract factory builder prototype singleton	Adapter(Object) Bridge composite decorator Facade flyweight proxy	Chain of Responsibility Command Iterator Mediator Memento Observer



M2006C3MII

(Specifies whether a design pattern applies primarily to classes or to objects)	Object Pattern	Abstract factory builder prototype singleton	Adapter(Object) Bridge composite decorator Facade flyweight proxy	Method Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Type here to search Rain... ENG 17:59 03-08-2023

M2006C3MII

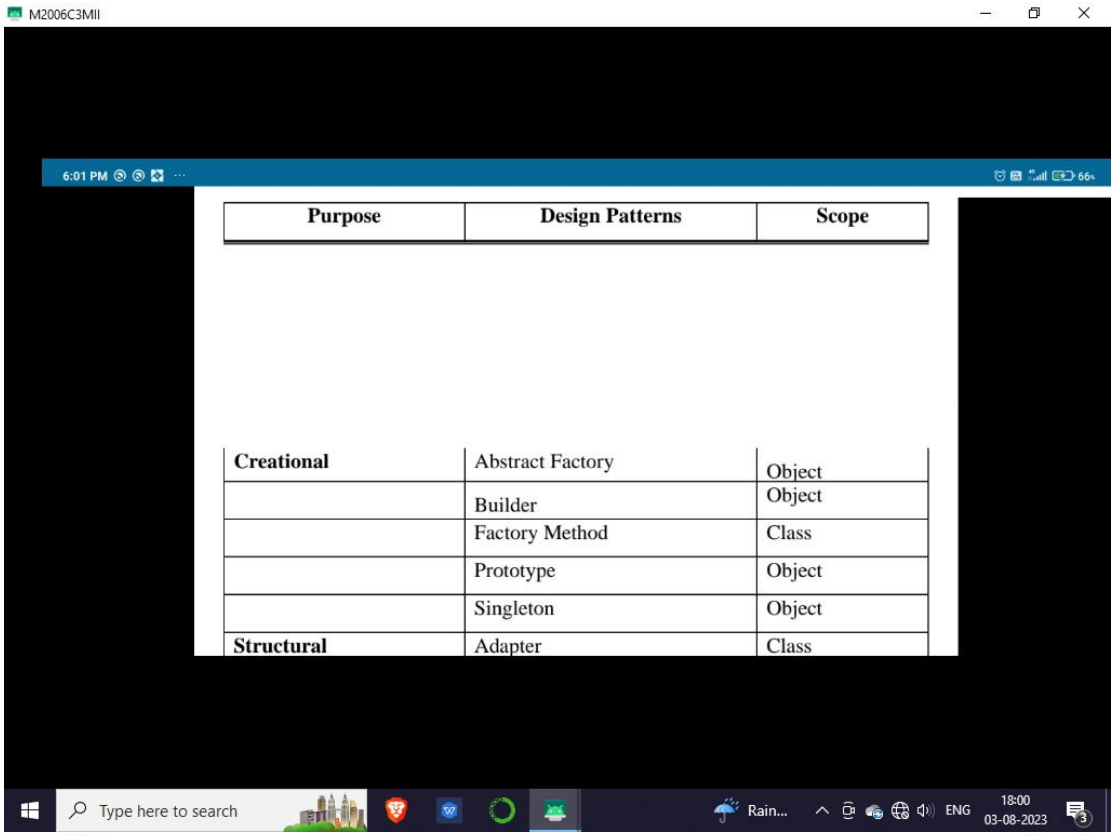
Classification of Design Patterns

Design patterns can be classified by two criteria:

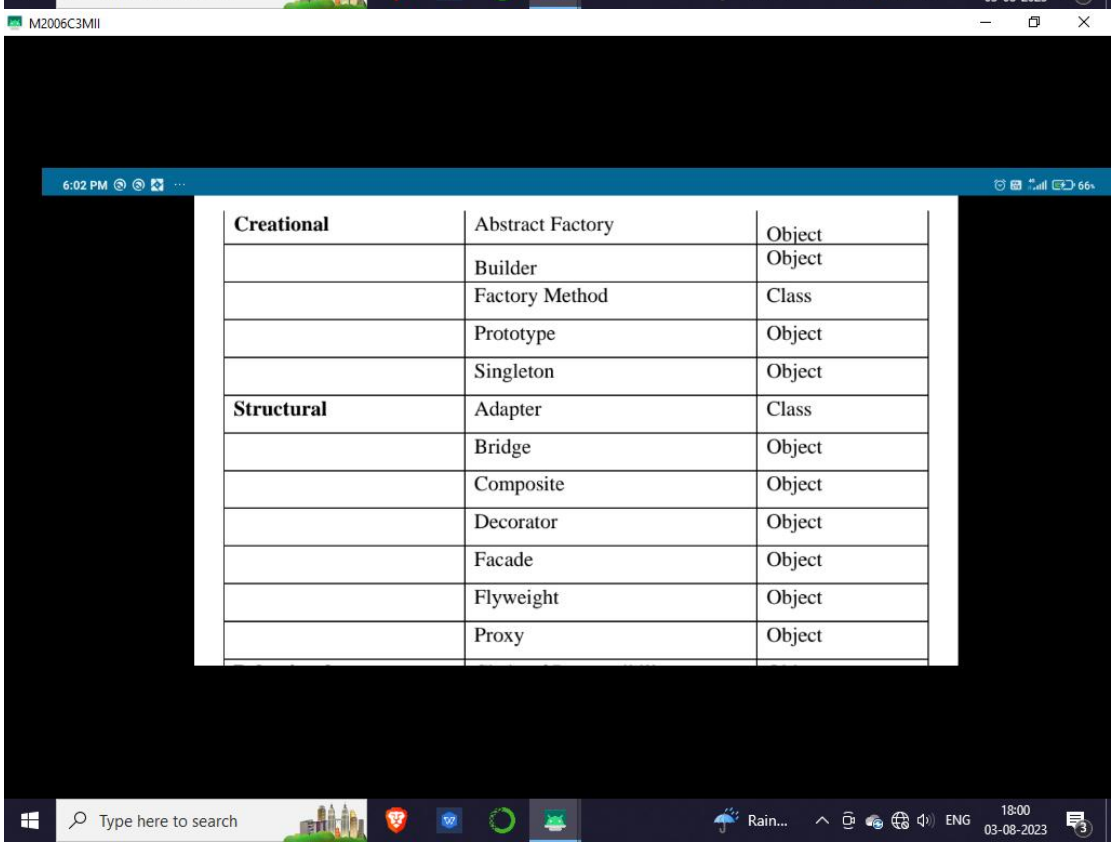
- **Purpose**, which reflects what a pattern does. Patterns can have creational, structural or behavioural purpose;
  - Creational patterns concern the process of object creation
  - Structural patterns deal with the composition of classes and objects
  - Behavioural patterns characterize the ways in which classes and objects interact and distribute responsibility.
- **Scope**, which specifies whether the pattern applies primarily to classes or to objects:
  - Class patterns deal with relationships between classes and their subclasses. These relationships are established through inheritance, so they are static.
  - Object patterns deal with object relationships, which can be changed at run-time and are more dynamic.

The following table shows the classification of design patterns. Note, that those are the patterns introduced by the Gang of Four.

Type here to search Rain... ENG 18:00 03-08-2023



Purpose	Design Patterns	Scope
Creational	Abstract Factory	Object
	Builder	Object
	Factory Method	Class
	Prototype	Object
	Singleton	Object
Structural	Adapter	Class



Purpose	Design Patterns	Scope
Creational	Abstract Factory	Object
	Builder	Object
	Factory Method	Class
	Prototype	Object
	Singleton	Object
Structural	Adapter	Class
	Bridge	Object
	Composite	Object
	Decorator	Object
	Facade	Object
	Flyweight	Object
	Proxy	Object

M2006C3MII

6:02 PM

Behavioral	Chain of Responsibility	Object
	Command	Object
	Interpreter	Class
	Iterator	Object
	Mediator	Object
	Memento	Object
	Observer	Object
	State	Object
	Strategy	Object
	Template Method	Class
	Visitor	Object

Type here to search Rain... ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

Que:5 How Design Patterns Solve Design Problems OR Properties of Design Patterns/ OR List and explain any 4 design problems. Also explain how design patterns solve these design problems faced by object – oriented software designers.

Design patterns solve many of the day-to-day problems object-oriented designers face, and in many different ways. Here are several of these problems and how design patterns solve them

1) Finding Appropriate Objects

- Object-oriented programs are made up of objects. The hard part about object-oriented design is decomposing a system into objects. Design patterns can help in this process by identifying less obvious abstractions and the objects that capture them.
- For example, objects that represent a process or algorithm don't occur in nature, but they can be crucial in a flexible design. The Strategy pattern describes how to implement families of algorithms to solve a particular problem. Those algorithms can be interchanged at run-time, since they are objects now and are subject to polymorphism.

2) Determining Object Granularity

- Usually, objects vary in size and number. They can represent everything down to the

Type here to search Rain... ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

2) Determining Object Granularity

- Usually, objects vary in size and number. They can represent everything down to the hardware or all the way up to entire applications.
- Design patterns can help to determine proper object granularity. For example, the Facade pattern describes how to represent complete subsystems as objects, and the Flyweight pattern describes how to support huge numbers of objects at the finest granularities.
- A number of other patterns, such as Composite, describe how to decompose an object into smaller objects.

3) Specifying Object Interfaces :

- Every operation declared by an object specifies the operation's name, the objects it takes as parameters, and the operation's return value. This is known as the operation's signature.
- The set of all signatures defined by an object's operations is called the interface to the object.

Type here to search 28°C ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

3) Specifying Object Interfaces :

- Every operation declared by an object specifies the operation's name, the objects it takes as parameters, and the operation's return value. This is known as the operation's signature.
- The set of all signatures defined by an object's operations is called the interface to the object.
- An object's interface characterizes the complete set of requests that can be sent to the object. Any request that matches a signature in the object's interface may be sent to the object.
- Interfaces are fundamental in object oriented systems. Objects are known only through their interfaces.
- There is no way to know anything about an object or to ask it to do anything without going through its interface.
- An object's interface says nothing about its implementation—different objects are free to implement requests differently. That means two objects having completely different implementations can have identical interfaces.

Type here to search 28°C ENG 18:00 03-08-2023

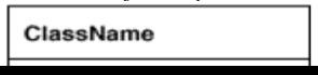
M2006C3MII

6:02 PM

- Design patterns help programmers to define interfaces by identifying their key elements and the kind of data that get sent across an interface. A design pattern can also tell what not to put in the interface.
- For example, the Memento pattern describes how to encapsulate and save the internal state of an object so that the object can be restored to that state later. The pattern stipulates that Memento objects must define two interfaces:
  - ✓ A restricted one that lets clients hold and copy mementos
  - ✓ A privileged one that only the original objects can use to store and retrieve state in the memento.

4) Specifying Object Implementations:

- An object's implementation is defined by its class.
- The class specifies the object's internal data and representation and defines the operations the object can perform.



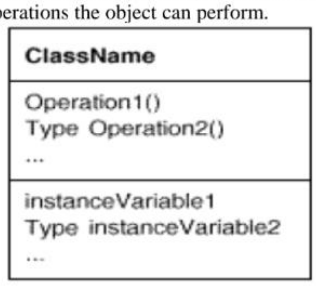
Type here to search 28°C ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

4) Specifying Object Implementations:

- An object's implementation is defined by its class.
- The class specifies the object's internal data and representation and defines the operations the object can perform.



- Many similar instances of an object can be created by instantiating a class.
- Design pattern help to identify class and object implementation details in terms of object instantiation or new classes can be defined in terms of existing classes using inheritance.

Type here to search 28°C ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

5) Designing for Change

- Designing a system that is robust to changes is a rather hard task to do. However, a design that doesn't take changes into account risks major redesigns in the future. Design patterns can ensure that a system can change in specific ways.
- Each design pattern lets some aspect of the system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change.
- Let us look on some examples:

- Creating an object by specifying a class explicitly commits to a particular implementation instead of a particular interface. That means if we in the future want to change the object that we use we need also to implement the client code again. On the other hand using the design patterns, such as Abstract Factory pattern lets you avoid this problem.
- Dependence on hardware and software platform. Clients that know how an object is represented, stored, located, or implemented might need to be changed when the object

Type here to search 28°C ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

changes. Hiding this information from clients keeps changes from cascading. An example is again Abstract Factory used to create different look-and-feel components across different operating systems.

Que: 6 Explain the advantages and disadvantages of design patterns

Advantages:

1) Common Design Vocabulary

Computer scientists and programmers create names for algorithms and data structures. Similarly, design patterns should provide a common vocabulary for designers. Such vocabulary can be used to communicate, document and explore design decisions.

2) They are highly flexible and can be used in practically any type of application or domain
3) They provide the solutions that help to define the system architecture.
4) They capture the software engineering experiences.
5) They provide transparency to the design of an application.

Type here to search 28°C ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

1) Common Design Vocabulary

Computer scientists and programmers create names for algorithms and data structures. Similarly, design patterns should provide a common vocabulary for designers. Such vocabulary can be used to communicate, document and explore design decisions.

2) They are highly flexible and can be used in practically any type of application or domain
3) They provide the solutions that help to define the system architecture.
4) They capture the software engineering experiences.
5) They provide transparency to the design of an application.
6) They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
7) Design patterns don't guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.
8) Speed up the development process by providing tested, proven development paradigm.

Type here to search 28°C ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

2) They are highly flexible and can be used in practically any type of application or domain
3) They provide the solutions that help to define the system architecture.
4) They capture the software engineering experiences.
5) They provide transparency to the design of an application.
6) They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
7) Design patterns don't guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.
8) Speed up the development process by providing tested, proven development paradigm.
9) They have a proven track record as they are already widely used and thus reduce the technical risk to the project
10) They are language neutral and so can be applied to any language that supports object-orientation

Type here to search 28°C ENG 18:00 03-08-2023

M2006C3MII

6:02 PM

Disadvantages:

- Patterns do not lead to direct code reuse.
- Patterns are Complex in nature
- Teams may suffer from patterns overload.
- Integrating patterns into a software development process is a human-intensive activity.
- They can decrease understandability by adding indirection or increasing the amount of code.
- **They may introduce inefficiency into the final product or solution.** What was the best solution last year is not guaranteed to be the best solution this year.

Type here to search 28°C ENG 18:01 03-08-2023

M2006C3MII

6:03 PM

Que: 7) How to Select a Design Pattern: OR Explain in short several approaches to find an appropriate design pattern to solve a problem.

Ans:

With more than 20 design patterns in the catalog to choose from, it might be hard to find the one that addresses a particular design problem, especially if the catalog is new and unfamiliar. Here are several different approaches to finding the design pattern that's right for your problem:

1. ***Consider how design patterns solve design problems***.
   Design patterns help you find appropriate objects, determine object granularity, specify object interfaces, and several other ways in which design patterns solve design problems. Referring to these discussions can help guide your search for the right pattern.
2. ***Scan Intent sections***.
   Lists the Intent sections from all the patterns in the catalog. Read through each pattern's

Type here to search 28°C ENG 18:01 03-08-2023

M2006C3MII

6:03 PM

1. ***Consider how design patterns solve design problems***.
   Design patterns help you find appropriate objects, determine object granularity, specify object interfaces, and several other ways in which design patterns solve design problems. Referring to these discussions can help guide your search for the right pattern.
2. ***Scan Intent sections***.
   Lists the Intent sections from all the patterns in the catalog. Read through each pattern's intent to find one or more that sound relevant to your problem.
3. ***Study how patterns interrelate***.
   Relationships between design patterns graphically. Studying these relationships can help direct you to the right pattern or group of patterns.
4. ***Study patterns of like purpose***. The catalog has three chapters, one for creational patterns, and another for structural patterns, and a third for behavioral patterns. Each chapter starts off with introductory comments on the patterns and concludes with a section that compares

Type here to search

28°C ENG 18:01 03-08-2023

M2006C3MII

6:03 PM

4. ***Study patterns of like purpose***. The catalog has three chapters, one for creational patterns, and another for structural patterns, and a third for behavioral patterns. Each chapter starts off with introductory comments on the patterns and concludes with a section that compares and contrasts them. These sections give you insight into the similarities and differences between patterns of like purpose.
5. ***Examine a cause of redesign***. Look at the causes of redesign, and then look at the patterns that help you avoid the causes of redesign.
6. ***Consider what should be variable in your design***. This approach is the opposite of focusing on the causes of redesign. Instead of considering what might *force* a change to a design, consider what you want to be *able* to change without redesign

Que:8) How to use a Design Pattern?

Type here to search

28°C ENG 18:01 03-08-2023

M2006C3MII

6:03 PM

Que:8) How to use a Design Pattern?

First choose a correct design pattern based on problem statement and then use a step-by-step approach to applying a design pattern effectively:

After understanding what it is and why you would want to use a design pattern, the steps of using a design pattern can be summarized as below:

1. ***Read the pattern once through for an overview***. Pay particular attention to the Applicability and Consequences sections to ensure the pattern is right for your problem.

Type here to search 28°C ENG 18:01 03-08-2023

M2006C3MII

6:03 PM

1. ***Read the pattern once through for an overview***. Pay particular attention to the Applicability and Consequences sections to ensure the pattern is right for your problem.
2. ***Go back and study the Structure, Participants, and Collaborations sections***. Make sure you understand the classes and objects in the pattern and how they relate to one another
3. ***Look at the Sample Code section to see a concrete example of the pattern in code***. Studying the code helps you learn how to implement the pattern.
4. ***Choose names for pattern participants that are meaningful in the application context***. The names for participants in design patterns are usually too abstract to appear directly in an application. Nevertheless, it's useful to incorporate the participant name into the name that appears in the application. That helps make the pattern more explicit in the implementation. For example, if you use the Strategy pattern for a text compositing algorithm, then you might have classes SimpleLayoutStrategy or TeXLayoutStrategy.
5. ***Define the classes.*** Declare their interfaces, establish their inheritance relationships, and

Type here to search 28°C ENG 18:01 03-08-2023

M2006C3MII

6:03 PM

5. ***Define the classes.*** Declare their interfaces, establish their inheritance relationships, and define the instance variables that represent data and object references. Identify existing classes in your application that the pattern will affect, and modify them accordingly.
6. ***Define application-specific names for operations in the pattern***. Here again, the names generally depend on the application. Use the responsibilities and collaborations associated with each operation as a guide. Also, be consistent in your naming conventions. For example, you might use the "Create-" prefix consistently to denote a factory method.
7. ***Implement the operations to carry out the responsibilities and collaborations in the pattern.*** The implementation section offers hints to guide you in the implementation. The examples in the Sample

Que: 9) Explain the meaning of following OOP principles:

i) "Favor object composition over class inheritance"

Type here to search 28°C ENG 18:01 03-08-2023

M2006C3MII

6:03 PM

Que: 9) Explain the meaning of following OOP principles:

i) "Favor object composition over class inheritance"

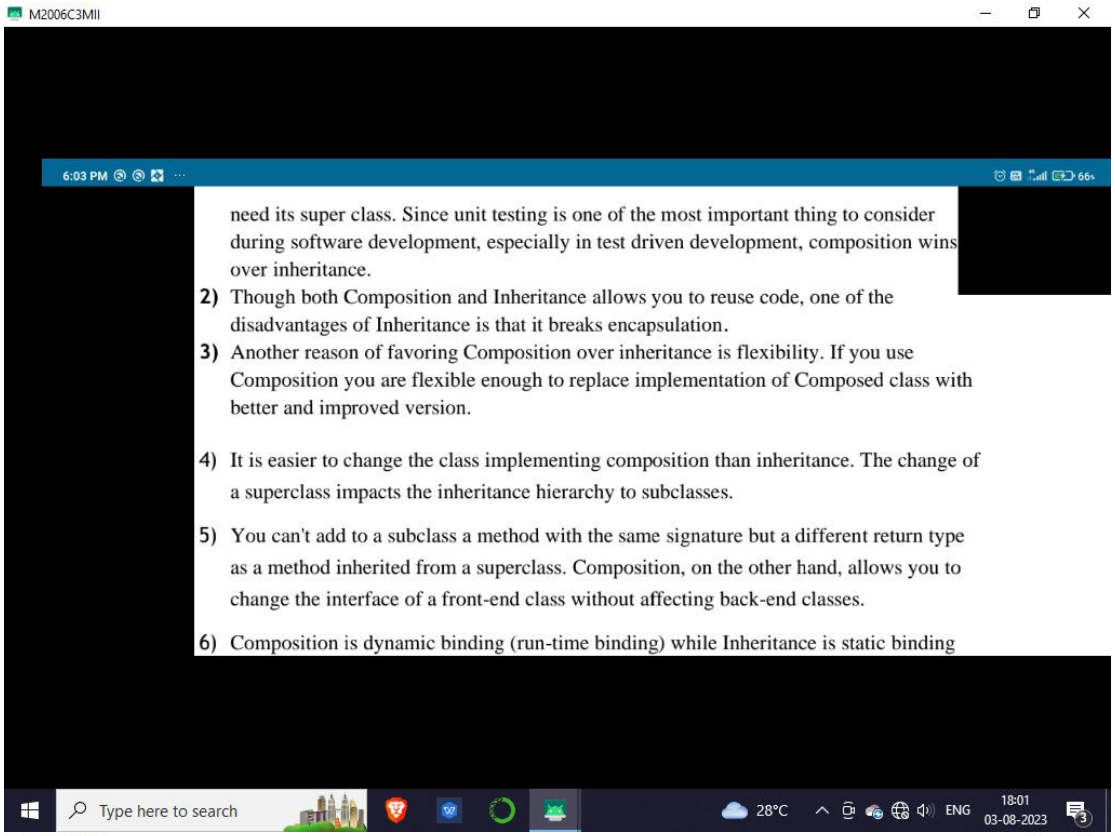
ii) "Program to an interface, not an implementation.

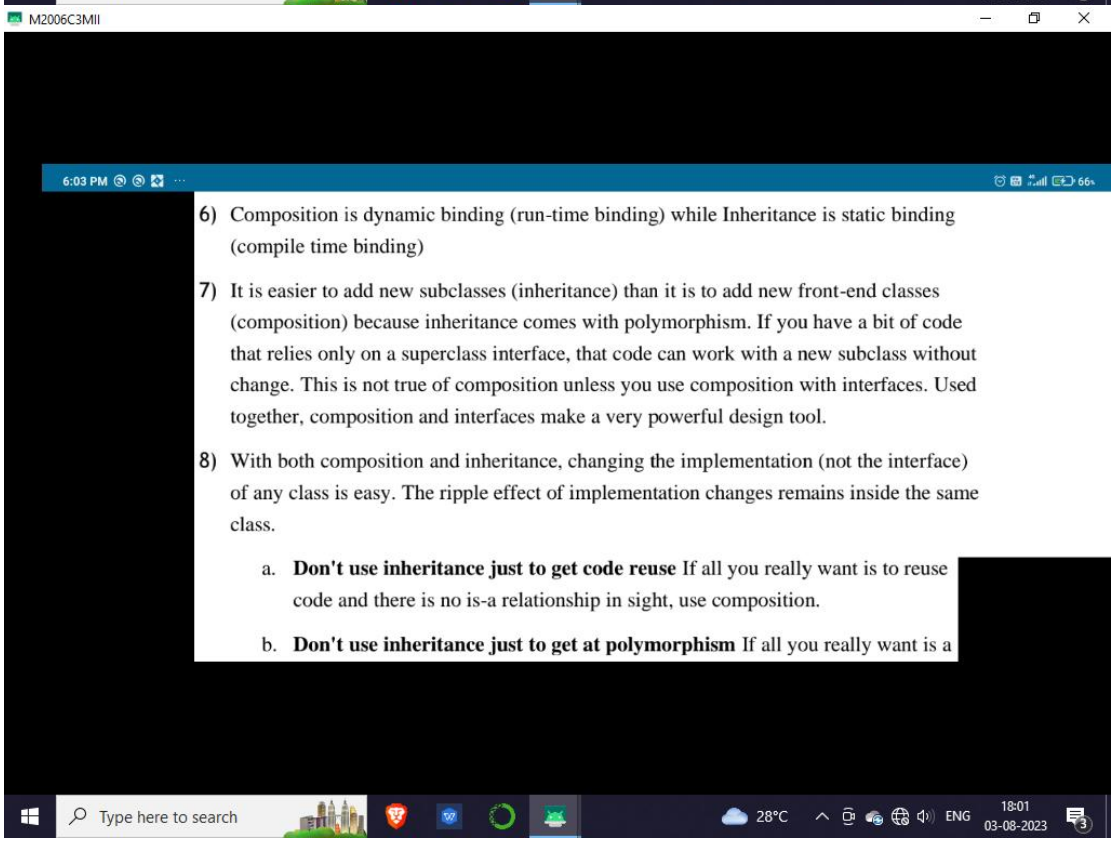
Ans: i) Favor composition over inheritance is a one of the popular object-oriented design principles, which helps to create flexible and maintainable code in object-oriented languages.

Reasons to Prefer Composition over Inheritance:

1) Composition offers better test-ability of a class than Inheritance. If one class is composed of another class, you can easily create Mock Object representing composed class for sake of testing. Inheritance doesn't provide this luxury. In order to test derived class, you must

Type here to search 28°C ENG 18:01 03-08-2023


M2006C3MII
6:03 PM
need its super class. Since unit testing is one of the most important thing to consider during software development, especially in test driven development, composition wins over inheritance.
2) Though both Composition and Inheritance allows you to reuse code, one of the disadvantages of Inheritance is that it breaks encapsulation.
3) Another reason of favoring Composition over inheritance is flexibility. If you use Composition you are flexible enough to replace implementation of Composed class with better and improved version.
4) It is easier to change the class implementing composition than inheritance. The change of a superclass impacts the inheritance hierarchy to subclasses.
5) You can't add to a subclass a method with the same signature but a different return type as a method inherited from a superclass. Composition, on the other hand, allows you to change the interface of a front-end class without affecting back-end classes.
6) Composition is dynamic binding (run-time binding) while Inheritance is static binding
Type here to search
28°C
ENG
18:01
03-08-2023


M2006C3MII
6:03 PM
6) Composition is dynamic binding (run-time binding) while Inheritance is static binding (compile time binding)
7) It is easier to add new subclasses (inheritance) than it is to add new front-end classes (composition) because inheritance comes with polymorphism. If you have a bit of code that relies only on a superclass interface, that code can work with a new subclass without change. This is not true of composition unless you use composition with interfaces. Used together, composition and interfaces make a very powerful design tool.
8) With both composition and inheritance, changing the implementation (not the interface) of any class is easy. The ripple effect of implementation changes remains inside the same class.
a. Don't use inheritance just to get code reuse If all you really want is to reuse code and there is no is-a relationship in sight, use composition.
b. Don't use inheritance just to get at polymorphism If all you really want is a
Type here to search
28°C
ENG
18:01
03-08-2023

M2006C3MII

6:03 PM

a. **Don't use inheritance just to get code reuse** If all you really want is to reuse code and there is no is-a relationship in sight, use composition.

b. **Don't use inheritance just to get at polymorphism** If all you really want is a polymorphism, but there is no natural is-a relationship, use composition with interfaces.

White-box reuse:
–Reuse by sub classing (class inheritance)

–Internals of parent classes are often visible to subclasses

–works statically, compile-time approach

–Inheritance breaks encapsulation

Type here to search 28°C ENG 18:01 03-08-2023

M2006C3MII

6:03 PM

•**Black-box** reuse:

–Reuse by object composition

–Requires objects to have well-defined interfaces

–No internal details of objects are visible

ii) **Program to an interface, not an implementation**
Benefits
–clients remain unaware of the specific types of objects they use.

–clients remain unaware of the classes that implement these objects.
•Manipulate objects solely in terms of interfaces defined by abstract classes!

•**Benefits**:
–Clients remain unaware of the specific types of objects they use.

—Do not declare variables to be instances of particular concrete classes

–Use creational patterns to create actual objects.

Type here to search 28°C ENG 18:02 03-08-2023

M2006C3MII

6:03 PM

•**Benefits**:

–Clients remain unaware of the specific types of objects they use.

—Do not declare variables to be instances of particular concrete classes

–Use creational patterns to create actual objects.

Que:10 compare and contrast the code reusability techniques: Inheritance and Composition

Sr. No	Inheritance	Composition
1	Inheritance is the design technique in object oriented programming to implement **is-a** relationship	Composition is the design technique in object oriented programming to implement **has-a** relationship
2	For.ex: Dog is-a Animal	For.ex: Person has-a Job relationship
3	Inheritance is tightly coupled code reusability technique	Composition is loosely coupled code reusability technique
4	It is less secure than composition	It is more secure than Inheritance
5	It provide less flexibility than	It provides more flexibility than inheritance

Type here to search 28°C ENG 18:02 03-08-2023

M2006C3MII

6:04 PM

Sr. No	Inheritance	Composition
1	Inheritance is the design technique in object oriented programming to implement **is-a** relationship	Composition is the design technique in object oriented programming to implement **has-a** relationship
2	For.ex: Dog is-a Animal	For.ex: Person has-a Job relationship
3	Inheritance is tightly coupled code reusability technique	Composition is loosely coupled code reusability technique
4	It is less secure than composition	It is more secure than Inheritance
5	It provide less flexibility than composition	It provides more flexibility than inheritance
6	Use inheritance only when you are sure that superclass will not be changed	Use Composition when you are not sure about superclass behavior
7	In inheritance we depend heavily on superclass and don't know what all methods of superclass will be used. So we will have to test all the methods of superclass. This is an	One more benefit of composition over inheritance is testing scope. Unit testing is easy in composition because we know what all methods we are using from other class.

Type here to search 28°C ENG 18:02 03-08-2023

M2006C3MII

6:04 PM

	composition	
6	Use inheritance only when you are sure that superclass will not be changed	Use Composition when you are not sure about superclass behavior
7	In inheritance we depend heavily on superclass and don't know what all methods of superclass will be used. So we will have to test all the methods of superclass. This is an extra work and we need to do it unnecessarily because of inheritance.	One more benefit of composition over inheritance is testing scope. Unit testing is easy in composition because we know what all methods we are using from other class.
8	Inheritance is static binding (compile time binding)	Composition is dynamic binding (run-time binding

Type here to search 28°C ENG 18:02 03-08-2023

M2006C3MII

6:04 PM

Que: 11) Select an appropriate answer for the following multiple choice questions.

i) Which of the following is true about a design pattern?

a) Design pattern is a data structure

b) Design pattern is a core of solution to a problem

c) Both A and B

d) None of these

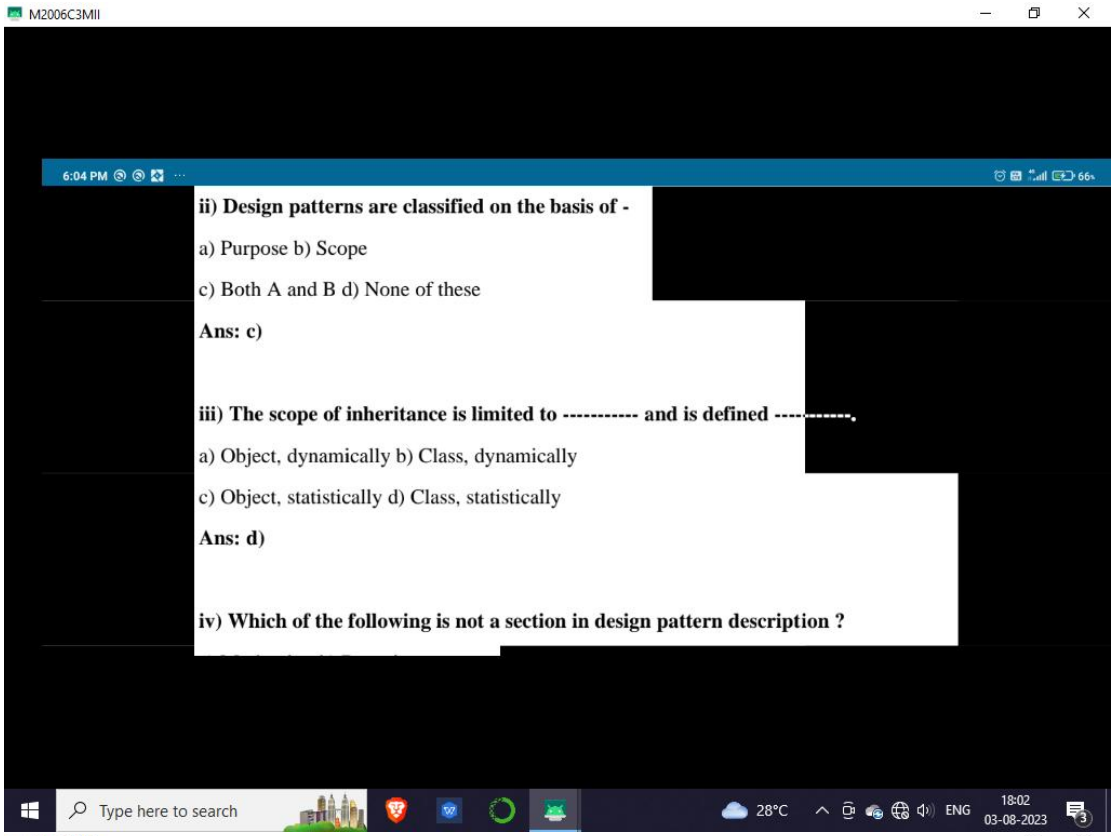
Ans: b

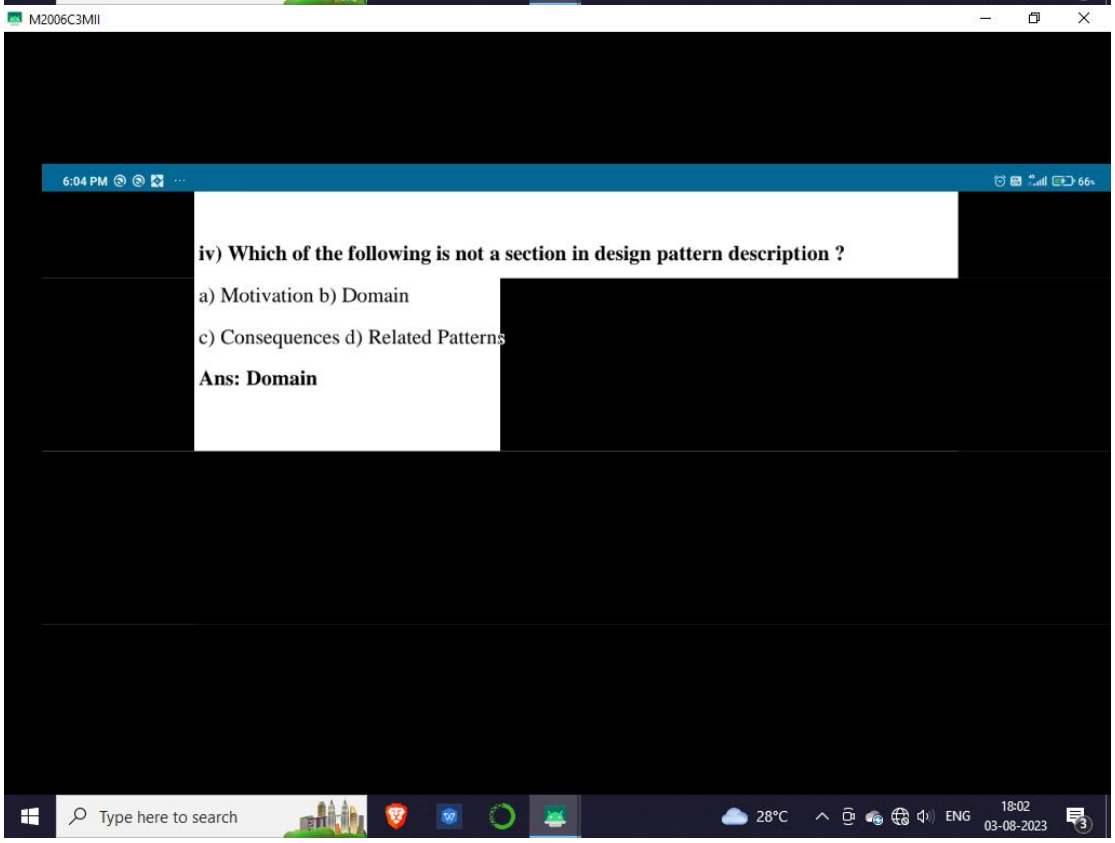
ii) Design patterns are classified on the basis of -

a) Purpose b) Scope

c) Both A and B d) None of these

Type here to search 28°C ENG 18:02 03-08-2023


M2006C3MII
6:04 PM
ii) Design patterns are classified on the basis of -
a) Purpose b) Scope
c) Both A and B d) None of these
Ans: c)
iii) The scope of inheritance is limited to ---------- and is defined ----------.
a) Object, dynamically b) Class, dynamically
c) Object, statistically d) Class, statistically
Ans: d)
iv) Which of the following is not a section in design pattern description ?
Type here to search
28°C
ENG
18:02
03-08-2023


M2006C3MII
6:04 PM
iv) Which of the following is not a section in design pattern description ?
a) Motivation b) Domain
c) Consequences d) Related Patterns
Ans: Domain
Type here to search
28°C
ENG
18:02
03-08-2023

M2006C3MII

6:04 PM

Que: 12 List the common causes of redesign of an existing system

Ans:

- **Creating a class by specifying a class explicitly** : If need be use creational patterns instead
- **Dependence on specific methods** : If need be use Chain of Responsibility or Command pattern instead
- **Dependence on Hardware and/or software platforms** : Consider using Abstract Factory or Bridge pattern instead
- **Dependence on Object Representation or Implementation**: Must program to interface and not to implementation. Abstract Factory, Bridge, Memento, Proxy etc patterns can help.
- **Dependence on specific Algorithms** : Consider using Strategy or Visitor pattern
- **Extending functionality by sub classing** : Prefer composition over inheritance
- **Inability to alter classes conveniently**: esp. with third party libraries. Adapter, Decorator and Visitor can help.

Type here to search 28°C ENG 18:02 03-08-2023

M2006C3MII

6:04 PM

- **Tight coupling:** Tight coupling between classes or object

Que 13: Explain the application of design patters

Ans: 1) Design pattern is used to develop games for mobile

2) It is used to obtain modular extensible and reusable design for power system.

3) Pattern is used in virtually all websites that consist of software that reacts to user input events, as opposed to static sites that simply display HTML-coded information.

4) Design pattern is also used for product design.

5) Take some points from uses & advantages of design pattern

Que 14: Explain the concept of software reusability used in design patters

Type here to search 28°C ENG 18:02 03-08-2023

M2006C3MII

6:04 PM

Que 14: Explain the concept of software reusability used in design patters

Ans:

- In software development similar problems are solved again and again.
- The basic principal of reuse is not to repeat solving of what has already been solved.
- Designing object-oriented software is hard, and designing *reusable* object-oriented software is even harder. You must find pertinent objects, factor them into classes at the right granularity, define class interfaces and inheritance hierarchies, and establish key relationships among them.

Type here to search 28°C ENG 18:02 03-08-2023

M2006C3MII

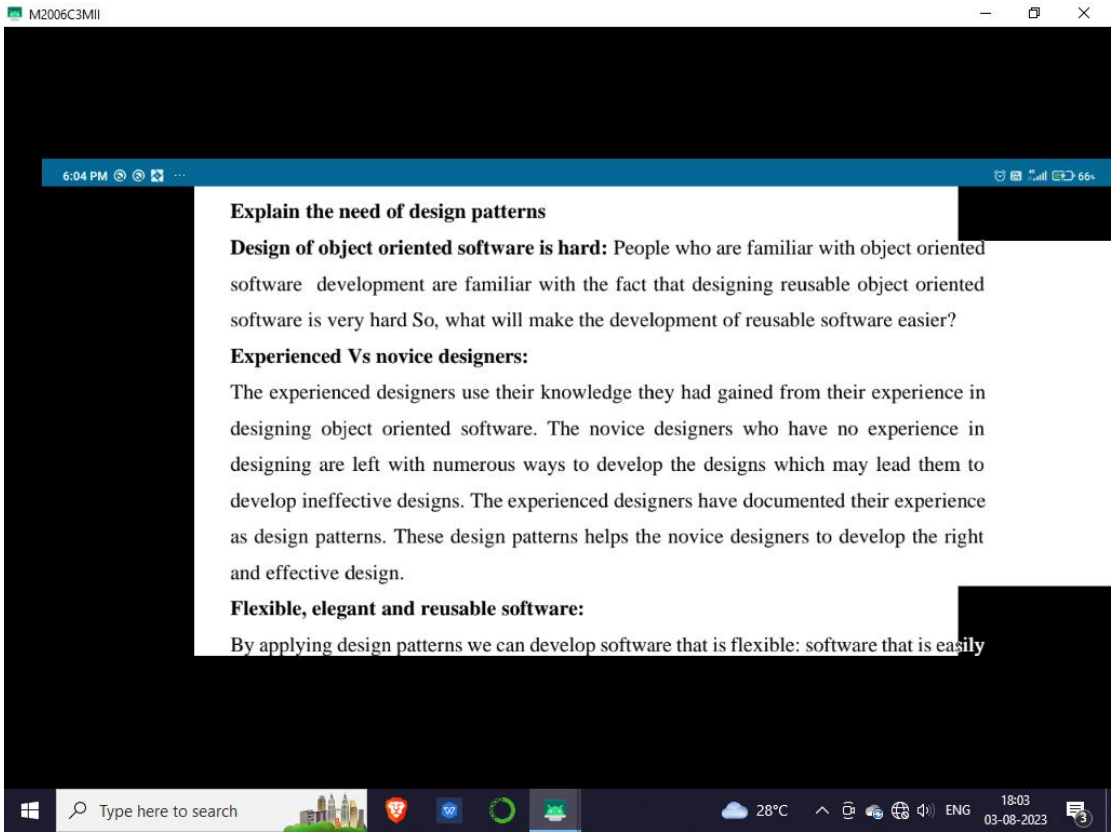
6:04 PM

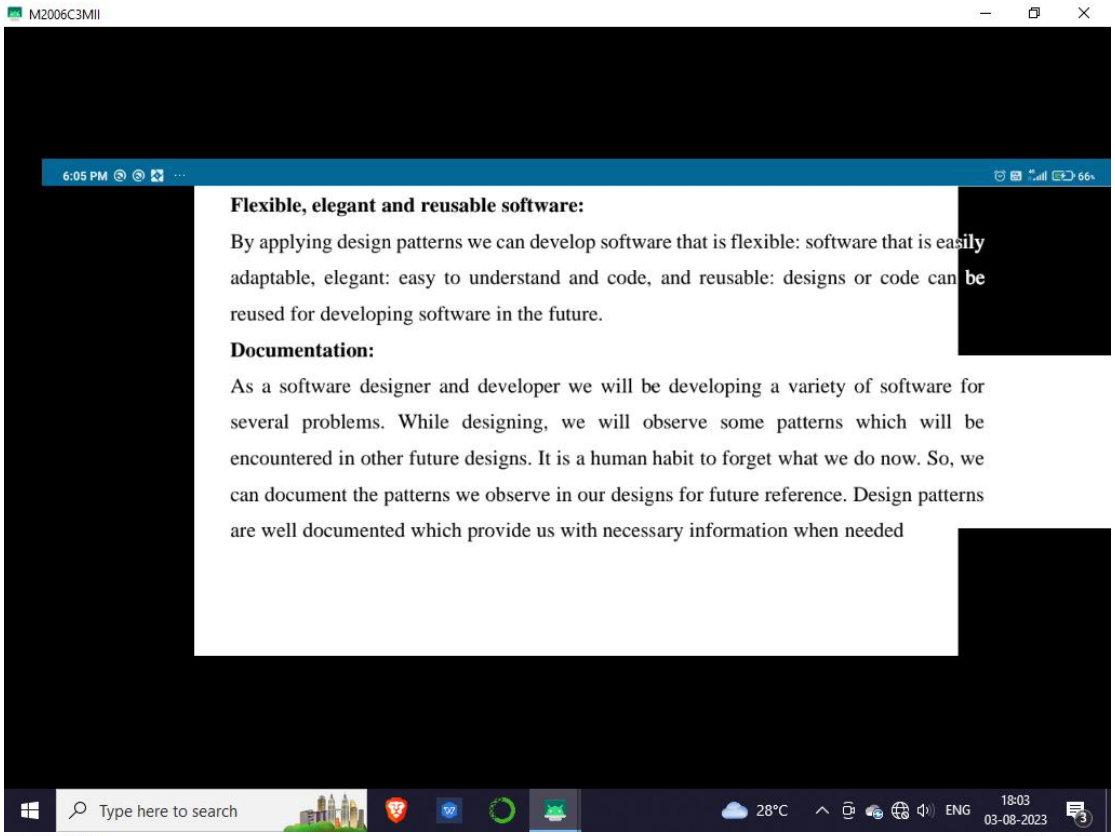
- Software design should be specific to the problem at hand but also general enough to address future problems and requirements.
- Software should designed in such a manner that it avoid redesign, or at least minimize it.
- Experienced object oriented designers will help to designed a reusable and flexible software design
- Design patterns make it easier to reuse successful designs and architectures. Expressing proven techniques as design patterns makes them more accessible to developers of new systems.
- Design patterns help you choose design alternatives that make a system reusable and avoid alternatives that compromise reusability

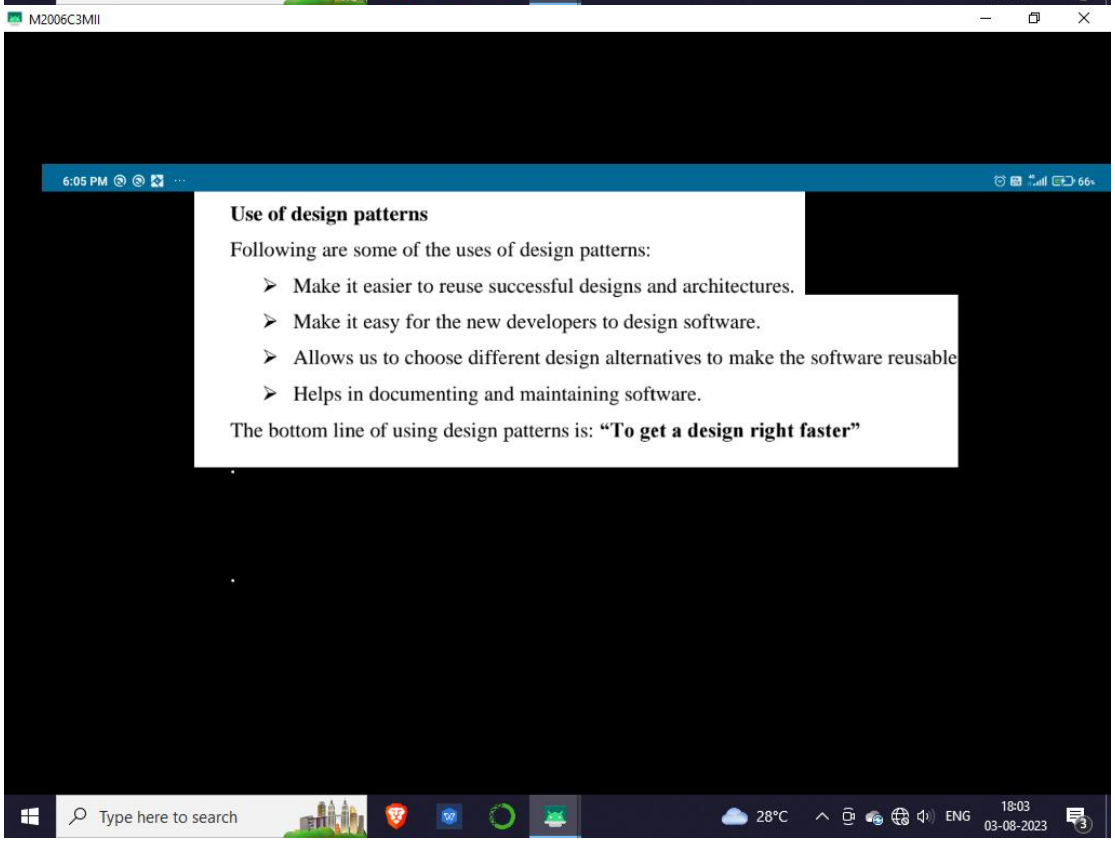
Explain the need of design patterns

Design of object oriented software is hard: People who are familiar with object oriented

Type here to search 28°C ENG 18:02 03-08-2023


M2006C3MII
6:04 PM
Explain the need of design patterns
Design of object oriented software is hard: People who are familiar with object oriented software development are familiar with the fact that designing reusable object oriented software is very hard So, what will make the development of reusable software easier?
Experienced Vs novice designers:
The experienced designers use their knowledge they had gained from their experience in designing object oriented software. The novice designers who have no experience in designing are left with numerous ways to develop the designs which may lead them to develop ineffective designs. The experienced designers have documented their experience as design patterns. These design patterns helps the novice designers to develop the right and effective design.
Flexible, elegant and reusable software:
By applying design patterns we can develop software that is flexible: software that is easily
Type here to search
28°C
ENG
18:03
03-08-2023


M2006C3MII
6:05 PM
Flexible, elegant and reusable software:
By applying design patterns we can develop software that is flexible: software that is easily adaptable, elegant: easy to understand and code, and reusable: designs or code can be reused for developing software in the future.
Documentation:
As a software designer and developer we will be developing a variety of software for several problems. While designing, we will observe some patterns which will be encountered in other future designs. It is a human habit to forget what we do now. So, we can document the patterns we observe in our designs for future reference. Design patterns are well documented which provide us with necessary information when needed
Type here to search
28°C
ENG
18:03
03-08-2023


M2006C3MII
6:05 PM
Use of design patterns
Following are some of the uses of design patterns:
Make it easier to reuse successful designs and architectures.
Make it easy for the new developers to design software.
Allows us to choose different design alternatives to make the software reusable
Helps in documenting and maintaining software.
The bottom line of using design patterns is: "To get a design right faster"
Type here to search
28°C
ENG
18:03
03-08-2023