IAT-1 Solution
Software Architecture & Design Patterns (15IS72)
September 2019

1 a) Define design pattern according to 'Christopher Alexander'. List the uses of design patterns.

* Def. from Christopher Alexander :- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

uses of design patterns :

(i) They make your Job easier
(ii) They ~~provide~~ help you analyze the more abstract areas of a program by providing concrete, well-tested solutions.
(iii) They encourage code reuse and accommodate change by supplying well-tested mechanisms for delegation & composition, and other non-inheritance based reuse technique.
(iv) They encourage more liable & maintainable code by following well-understood paths.
(v) They provide a common language & Jargon for progra-mmers.

b) Give the classification of design patterns in a neat tabular form and explain.

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight (195) Observer State Strategy Visitor

* Design patterns are classified by two criteria:
  (1) purpose - Reflects what a pattern does.
  - They can be:
    a) creational - concern the process of object creation.
    b) Structural - Deal with the composition of classes or objects.
    c) Behavioral - Characterize the ways in which classes or objects interact and distribute responsibility
  (2) Scope - Specifies whether the pattern applies primarily to classes or to objects
  - class patterns - deal with relationships between classes and their subclasses

which are established through inheritance; so they are static – fixed at compile time.

- Object patterns - deal with object relationships, which can be changed at run-time and are more dynamic.

* Creational class pattern & object pattern –
  Creational class patterns defer some part of object creation to subclasses, while creational object patterns ~~use inheritance to compose~~ defer it to another object.

* Structural class patterns & object patterns -
  Structural class patterns use inheritance to compose classes, while the structural object patterns describe ways to assemble objects.

2) What are the pitfalls, hints (or) techniques should a designer be aware, when implementing the design pattern? Explain.

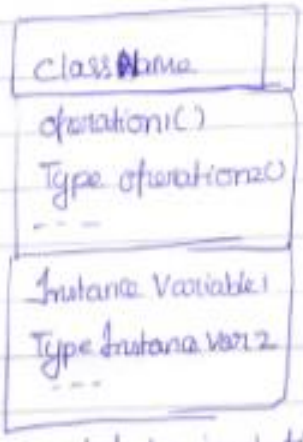
- Design patterns solve many of the day-to-day problems object-oriented designers face, and in many different ways.
- problems & solutions using design patterns are as follows:
  1) Finding Appropriate Objects:-
     - Object-oriented programs (OOP) are made up of objects — which package both data and the procedures that operate on the data
     - Decomposing a system into objects is the hard part in Object-oriented design.
     - Many objects in a design come from the analysis model. But, Object-oriented designs often end up with classes that have no counterparts in the real world.
     - Strict modeling of the real world leads to a system that reflects today's realities but not necessarily tomorrow's
     - The abstractions that emerge during design are key to making a design flexible
     - Design patterns identify less-obvious abstractions.
  2) Determining Object Granularity:-
     - Objects can vary tremendously in size and number.
     - Design patterns address this issue
       eg:- Facade pattern — describes how to represent subsystems as objects
       Flyweight pattern — describes how to support huge numbers of objects at the finest granularities

3) Specifying Object Interfaces:-

* Signature :- Every operation declared by an object specifies the operation's name, its parameters and return value. This is known as operation's signature.
* Interface :- The set of all signatures defined by an object's operations is called the interface to the object.
* 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.
* Type :- Name used to denote a particular interface.
  Subtype & Supertype - A subtype interface contains the interface of its supertype.
* Dynamic binding :- The run-time association of a request to an object and one of its operations is known as dynamic binding.
* polymorphism - It simplifies the definitions of clients, decouples objects from each other, and lets them vary their relationships to each other at run-time.
* Design patterns for interfaces -
  - They define interfaces by identifying their key elements and the kinds of data that get sent across an interface.
  - They are specify relationships b/w interfaces.
  - Eg:- (1) Memento pattern - describes how to encapsulate & save the internal state of an object so that the object can be restored to it' that state later.

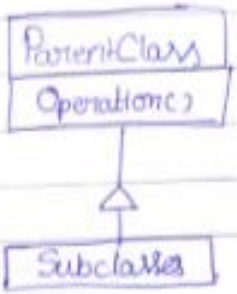
4) Specifying Object Implementation:

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



- Objects are created by instantiating a class i.e, an object is an instance of the class. This allocates storage for the objects internal data and associates the operations with these data.

- Class inheritance — Defining new classes (Subclass) in terms of existing classes (Parent class).
  - This includes the definitions of all the data & operations that the parent class defines.
  - objects that are instances of the subclasses will contain all data defined by the subclass and its parent classes and also perform all operations defined by this subclass and its parents.



* Class Vs Interface Inheritance:
  - Class inheritance defines an object's implementation in terms of another object's implementation
  - Interface inheritance describes when an object can be used in place of another.
  - Many of the design patterns depend on this distinction
  - Benefits:
    > Clients remain unaware of the specific types of objects they use
    > clients remain unaware of the classes that implement these objects.

* Programming to an Interface, not an Implementation

- There are two benefits to manipulating objects solely in terms of the interface defined by abstract class:
  1) Clients remain unaware of the specific types of objects, they use, as long as the objects adhere to the interface that clients expect
  2) clients remain unaware of the classes that implement these objects. Clients only know about the abstract class(es) defining the interface.
- These benefits reduces implementation dependencies between subsystems that leads to the following principle of reusable object-oriented design:
  "Program to an interface, not an implementation"

5) Putting Reuse Mechanisms to work:-

* Inheritance Vs Composition:-
  - Inheritance is often referred to as "white-box", as the internals of the parent classes are often visible to subclasses.

- Composition is refered to as "black-box" reuse because no internal details of objects are visible.
- Advantages & disadvantages:
- Inheritance:

Adv:- (i) Defined statically at compile time
(ii) Makes easier to modify the implementation being used.
(iii) Can override the operations

disadv:- (i) Can't change implementation dynamically at run-time.
(ii) It breaks encapsulation as it exposes a subclass to details of its parent's implementation.

- Composition:

Adv:- (i) Any object can be replaced at run-time by another as long as it has the same type.
(ii) Defined dynamically at run-time
(iii) Encapsulation is not broken
(iv) An object's implementation will be written in terms of object interfaces i.e., Very few implementation dependencies

disadv:- (i) class & class hierarchies will remain small
(ii) will have more objects in fewer classes

- Conclusion:-

"Favour object composition over class inheritance".

3) Explain how Frameworks are different from design patterns in software architecture.

a.

Design Patterns in framework

- A framework is a set of cooperating classes that make up a reusable design for a specific class of software.
- framework can be customized to a particular application by creating application-specific subclasses of abstract classes from the framework
- Framework dictates the architecture of your application
- It emphasize design reuse over code reuse
- In toolkit, we write the main body of the application and call the code we want to reuse. But in framework, we reuse the main body and write the code it calls.
- Adv:- Build an application faster, easier to maintain and more consistent to their users.
- Mature frameworks usually incorporate several design patterns
- Differences b/w framework & design pattern
  a) Design patterns are more abstract than frameworks
  b) " " smaller architectural elements than framework
  c) " " are less specialized than frameworks

Design patterns are most abstract than frameworks : Frameworks can be embodied in code, but only examples of patterns can be embodied in code. A strength of

frameworks is that they can be written down in programming languages and not only studied but executed and reused direcrtly. In contrast, the design pattern can be implemented, each time they are used. Design patterns also explain the intent, tradeoffs, and consequences of the design.

b. Design patterns are smaller architectural elements than framework : A typical framework contains several design patterns but the reverse is never true.
c. Design patterns are less specialized than frameworks : frameworks always have a particular application domain. A graphical editor framework might be used in factory simulation, but it won't be mistaken for a simulation framework. In contrast, the design patterns in this catalog, can be used in nearly any kind of application.

Frameworks are becoming increasingly common and important. They are the way Object-Oriented Systems achieve the most reuse.Larger Object-oriented applications will end up consisting of layers of frameworks that cooperate with each other. Most of the design and code in the application will come from or influednced by the frameworks it uses.

4 a) How is OOD related to software system? Explain.

Object-Oriented development is a software system with a collection of objects of various types that interact with each other through well-defined interfaces. This methodology provides us with following benefits:

- Faster Development: OOD leads to faster development of software design.
- Reuse of Previous work: OOD produces software modules that can be plugged into one another, which allows creation of new programs by reusing.
- Increased Quality: Increases in quality are largely a by-product of this program reuse.
- Modular Architecture: Object-oriented systems have a natural structure for modular design: objects, subsystems, framework, and so on. Thus, OOD systems are easier to modify.
- Client/Server Applications: the object-message paradigm of OOD meshes well with the physical and conceptual architecture of client/server applications.
- Better Mapping to the Problem Domain: This is a clear winner for OOD, particularly when the project maps to the real world. Whether objects represent customers, machinery, banks, sensors or pieces of paper, they can provide a clean, self-contained implication which fits naturally into human thought processes.

4 b) List and explain the key concepts of OOD.

Key Concepts of Object-orient Design (OOD)

- OOD paradigm, defines a software system as a collection of objects of various types that interact with each other through well-defined interfaces.
- The central role of objects:-
  - The notion of an object is centered around data & methods that could be used to modify it.
  - This provides abstraction that is very stable, independent of the changing requirements of the application.
  - The execution of each process relies on the objects to store and provide necessary operations on data.
- The notion of a class:
  - Classes allows objects to be represented as different types of entities
  - They help in defining hierarchies, and engage in specialisation & generalisation of objects.
- Abstract specification of functionality.
  - Specifying the properties of objects that are needed by a system, is abstract — doesn't place any restrictions on how the functionality is achieved.
  - This specification, called an interface or an abstract class, is like a contract for the implementer.

* A language to define the system:-

- The Unified Modelling language (UML) is the standard tool for describing the end product of the design activities.
- The documents generated in this language can be universally understood and are thus analogous to the 'blueprints' used in other engg. disciplines.

* Standard solutions:-

- The existence of an object structure facilitates the documenting of standard solutions, called design patterns - provide common form of reuse of solutions.

* An analysis process to model a system:-

- Object - oriented provides a systematic way to translate a functional specification to a conceptual design.
- This design describes the system in terms of Conceptual classes from which, the implementation classes, that constitute finished software is generated

* The notion of extenability & adaptability

- It helps us to modify existing entities in small ways to create new ones.
- Inheritance - creates new descendent class that modifies the features of an existing class
- Composition - which uses objects belonging to existing classes as elements to constitute a new class.

5) Describe the benefits to manipulating objects solely in terms of the interface defines by the abstract class

Benefits and drawbacks of the paradigm:

* Adv:-
  1) objects often reflect entities in application systems, which makes it easier for a designer to come up with classes in the design.
  2) Object-oriented helps increase productivity through reuse of existing Software.
  3) It is easier to accomodate changes.
  4) The ability to isolate changes, encapsulate data, and employ modularity reduces the risks involved in system development.

* Disadv:-
  1) The object-oriented development process introduces many layers of Software, which increases overheads
  2) Objects tend to have complex associations, which can result in non-locality, leading to poor memory access times.
  3) Programmers and designers schooled in other paradigms, find it difficult to learn and use object-oriented principles

6) Define the following w.r.t OOD

a) Modular design b) Encapsulation c) Cohesion
e) Coupling f) Modifiability g) Testability

• Modular design and encapsulation:

- Modularity: Refers to putting a large system by developing a no. of distinct components independently and then, integrating them to provide required functionality
  - The system's functionality must be provided by a number of well-designed, cooperating modules, with an interface that defines how they interact with the module.
- Encapsulation - Refers to the module that hides the details of its implementation from external agents
  - Eg:- abstract data type (ADT).

* cohesion and coupling

- cohesion - It is a measure of how well the entities within a module work together to provide functionality
  - Highly cohesive modules, tend to be more reliable, reusable and understandable
- coupling - Coupling refers to how dependent modules are on each other.
  - how coupling allows us to modify a module without worrying about the ramifications of the changes on the rest of the system.

- high coupling means that changes in one module would necessiate changes in other module, which may have a domino effect and also make it harder to understand the code.

- Modifiability & Testability :-

Modifiability :- Modification to a Software component can be done to change both functionality and design.
- object-orientation have set higher standards for adaptability

Testability :- Refers to both falsifiability & practical feasibility.
- It is the ease with which we can find bugs in a software system and the extent to which the structure of the system facilitates the detection of bugs.