11.Class Time Table

12.Individual Time Table

13. Lecture schedule with methodology being used/adopted along with tutorial, assignment and NPTEL class schedule

Micro Plan with dates and closure report

SNO	Unit No	Date	Topic Covered	No of Periods	Teaching aids used LCD/OHP/BB
1.	UNIT-1		What is a Design Pattern?	1	BB
2.			Describing Design Patterns,	1	BB
3.			Organizing The Design Catalog	1	BB
4.			How Design Patterns Solve Design Problems	1	BB
5.			How To Select A Design Pattern,	1	BB
6.			How To Use A Design Pattern	1	BB
7.			Design Patterns in Smalltalk MVC	1	BB

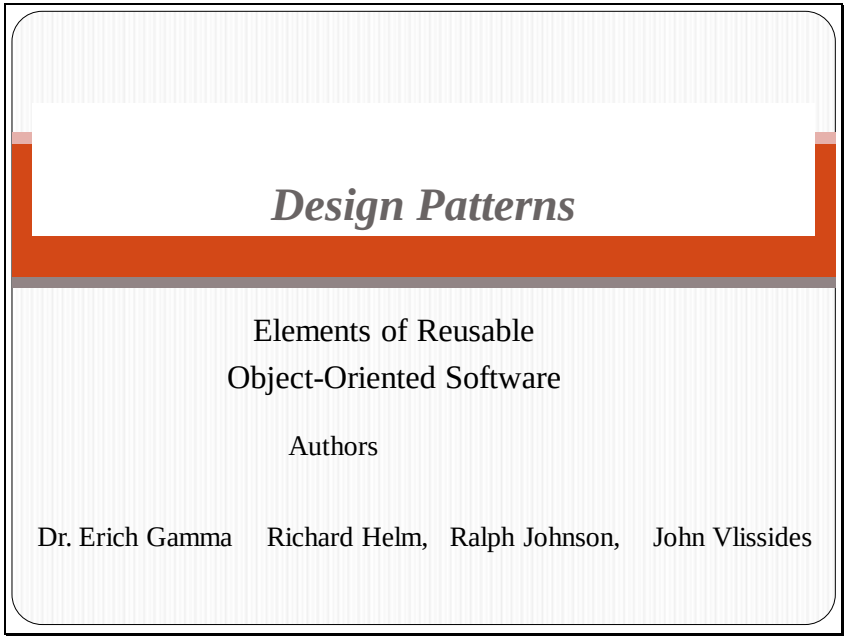
8.			The Catalog Of Design Patterns	1	BB/OHP
9.			Tutorial class /(NPTEL CLASS)	1	BB
10.			Assignment test	1	
11.	**UNIT-2**		Designing a Document Editor	1	BB
12.			Design problems, Document Structure	1	BB
13.			Formatting , Embellishing the User Interface	1	BB
14.			Supporting , Multiple Look and Feel Standards	1	BB/OHP
15.			Supporting Multiple Window Systems	1	BB
16.			User Operations Spell Checking And Hyphenation , Summary	1	BB
17.			Tutorial class/(NPTEL CLASS)	1	BB
18.			Assignment test	1	
19.	**UNIT-3**		Creational patterns	1	BB
20.			Abstract Factory	1	BB/OHP
21.			Builder, Factory Method	1	BB
22.			Prototype	1	BB
23.			Singleton	1	BB
24.			Discussion Of Creational Patterns	1	BB
25.			Tutorial class/(NPTEL CLASS)	1	BB
26.	**UNIT-5**		Structural pattern part-I	1	BB
27.			Adapter	1	BB
28.			Bridge	1	BB
29.			Composite	1	BB/OHP
30.			Tutorial class/(NPTEL CLASS)	1	BB
31.			Assignment test	1	

32.	**UNIT-5**		Structural pattern part-II	1	BB
33.			Decorator	1	BB
34.			Acade	1	BB
35.			Flyweight	1	BB
36.			Proxy.	1	BB/OHP
37.			Tutorial class/(NPTEL CLASS)	1	BB
38.			Assignment test	1	
39.	**UNIT-6**		Behavioral patterns part-I	1	BB
40.			Chain Of Responsibility	1	BB
41.			Command,	1	BB
42.			Interpreter	1	BB
43.			Iterator	1	BB/OHP
44.			Tutorial class/(NPTEL CLASS)	1	BB
45.			Assignment test	1	
46.	**UNIT-7**		Behavioral patterns part-II	1	BB
47.			Mediator	1	BB
48.			Memento	1	BB
49.			State , Strategy	1	BB
50.			Template, Method	1	BB
51.			Visitor,	1	BB
52.			Discussion of Behavioral patterns	1	BB
53.			Tutorial class/(NPTEL CLASS)	1	BB
54.			Assignment test	1	
55.	**UNIT-8**		What To Expect From Design Patterns	1	BB
56.			A Brief History	1	BB
57.			The Pattern Community	1	BB

58.			An Invitation	1	BB
59.			A parting Thought	1	BB/OHP
60.			Tutorial class/(NPTEL CLASS)	1	BB
61.			Assignment test	1	BB
62.			Solve University questions	1	BB
			Total number of Classes required	62	

15.Detailed Notes

Slide 1



Slide 2

UNIT----------1

INTRODUCTION

Slide 3

Design patterns describes simple and elegant solutions to specific problems in object-oriented software design.

Slide 4

Authors definition of **Design Patterns**

"The design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context".

OR

- **A Design Pattern is essentially a description of a commonly occurring object-oriented design problem and how to solve it**

Slide 5

Four essential elements of a pattern

- **Pattern Name**
  - It is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.
  - It gives higher level abstraction.
  - Finding good names has been one of the hardest parts of developing out catalog.
- **Problem**
  - It describes when to apply the pattern.
  - It explains the problem and context.
  - It describes how to represent algorithms as objects

Slide 6

- **Solution**
  - It describes the elements that make up the design, the relationships, responsibilities and collaborations.
  - 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
  - Pattern provides a general arrangement of elements to solve it
- **Consequences**
  - These are the results and trade –offs of applying the pattern
  - When describing design decisions, these are critical for evaluating design alternatives and for understanding the costs and benefits of applying the pattern.

Slide 7

A design pattern represents a widely accepted solution to a recurring design problem in OOP

Each design pattern focuses on a particular object-oriented design problem

Slide 8

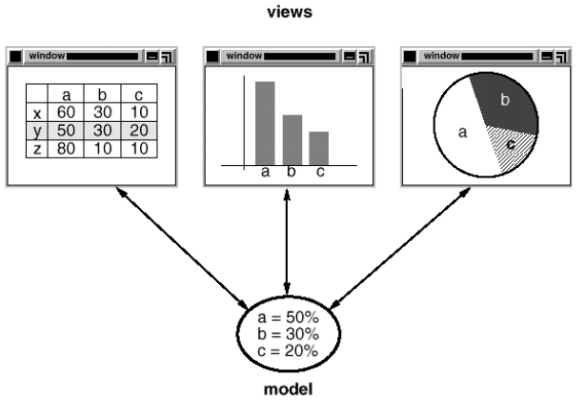
Design Patterns in Smalltalk MVC

The Model/View/Controller (MVC) triad of classes is used to build user interfaces in Smalltalk-80.

MVC consists of three kinds of objects. The **Model** is the application object, the **View** is its screen presentation, and the **Controller** defines the way the user interface reacts to user input

Slide 9

The example reflects a design that decouples views from models



P-9

Slide 10

Describing Design patterns

Each pattern is divided into sections according to the following template

Pattern Name and Classification

The pattern's name conveys the essence of the pattern succinctly.

A good name is vital because it becomes the design vocabulary

Intent

A short statement that answers the following questions

- What does the design pattern do?
- What is its rationale and intent?
- What particular design issue or problem does it address?

P-10

Slide 11

Also known as

Other names for the pattern, if any

Motivation

A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem

Applicability

What are the situations in which the design pattern can be applied? What are examples of poor designs that the pattern can address? How can you recognize these situations?

Slide 12

Structure

A graphical representation of the classes in the pattern using a notation based on the Object Modeling Technique (OMT).

We use interaction diagrams to illustrate sequences of requests and collaborations between objects.

Participants

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

Slide 13

Collaborations

How the participants collaborate to carry out their responsibilities

Consequences

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

Implementation

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

Slide 14

Sample Code

Code fragments that illustrate how you might implement the pattern in C++ or Smalltalk

Known Uses

Examples of the pattern found in real systems. At least 2 examples from different domains

Related patterns

What design patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?

Slide 15

Patterns

- This book defined 23 patterns, classified into three categories.
  - ***Creational patterns***, which deal with the process of object creation.
  - ***Structural patterns***, which deal primarily with the static composition and structure of classes and objects.
  - ***Behavioral patterns***, which deal primarily with dynamic interaction among classes and objects.

Slide 16

Catalog of Design Patterns

- *Creational Patterns*
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- *Structural Patterns*
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- *Behavioral Patterns*
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

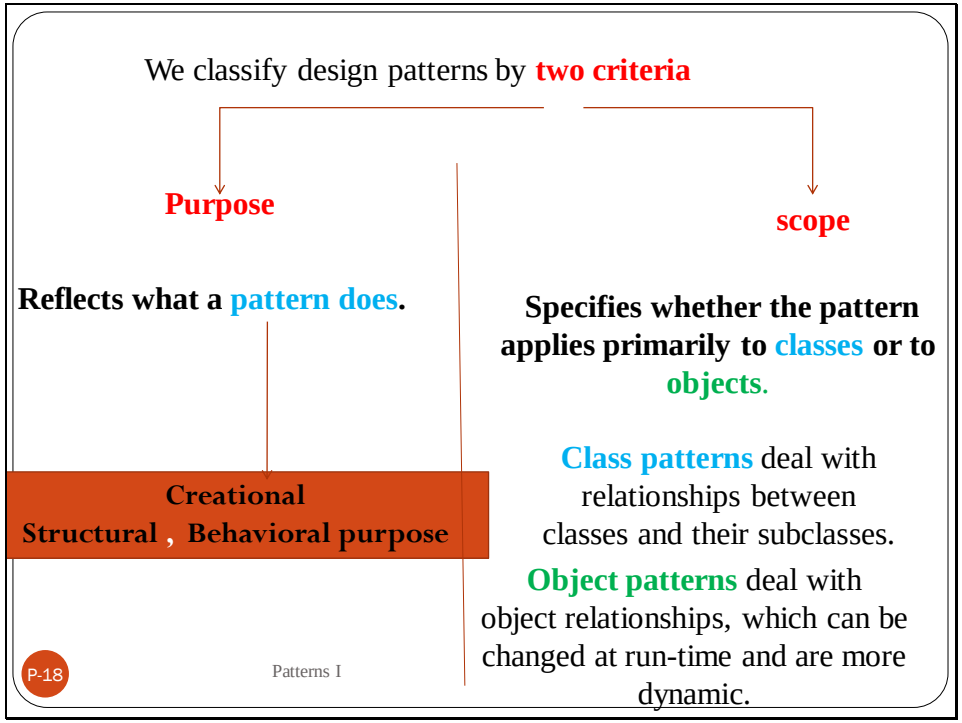
Slide 17

Organizing the catalog

Scope	Purpose		
	Creational	Structural	Behavioral
Class	Factory Method	Adapter(Class)	Interpreter Template Method
Object	Abstract Factory Builder Prototype Singleton	Adapter (Object) Bridge Composite Decorator Façade Flyweight proxy	Chain of responsibility Command Iterator Mediator Memento Observer State Strategy visitor

P-17

Slide 18


We classify design patterns by two criteria
Purpose
scope
Reflects what a pattern does.
Specifies whether the pattern applies primarily to classes or to objects.
Creational
Structural , Behavioral purpose
Class patterns deal with relationships between classes and their subclasses.
Object patterns deal with object relationships, which can be changed at run-time and are more dynamic.
P-18
Patterns I

Slide 19

How design patterns solve design problems

- **1) Finding Appropriate Objects**
  - An object packages both data and the procedures(methods or operations) that operate on the data
  - An object performs an operation when it receives a request from a client
  - These requests causes an object to execute an operation, operations are the only way to change an object's internal data, then object internal state is said to be **encapsulated** (i.e. object's representation is invisible from the outside).

P-19

Slide 20

- How to Create an object
  write problem statement find out nouns and verbs and create corresponding classes and operations .

- Design patterns help you identify less-obvious abstractions and the objects that can capture them.

The **State (305) pattern** represents each state of an entity as an object.

P-20

Slide 21

- **2) Determining object Granularity**
  - How do we decide what should be an object?
  - Subsystems can be represented as objects (Facade)
  - Objects of finest granularity (Flyweight)
  - Objects responsibilities can create other objects (Abstract Factory, Builder)
- **3) Specifying object interfaces**
  - An operation can be specified as operation name, parameters and its return value is known as *signature*
  - The set of all signatures defined by an object's operations is called the *interface* to the object.

Slide 22

- A *type* is a name used to denote a particular interface

Eg. Window obj1;

- A *type* is a *subtype* of another if its interface contains the interface of its *supertype*.
- The runtime association of a request to an object and one of its operations is known as *dynamic binding*.
- Dynamic binding lets us to substitute objects that have identical interfaces for each other at runtime is called as *Polymorphism*.

Design patterns help you define interfaces by identifying their key elements and the kinds of data that get sent across an interface.

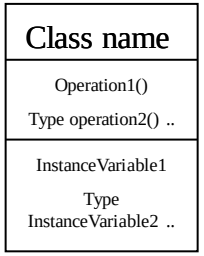
A design pattern might also tell you what *not to put in the interface*.

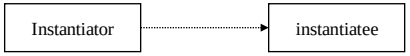
P-22

Slide 23

4) Specifying Object Implementations

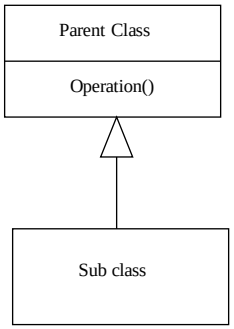
- An object's implementation is defined by its *class*. The class specifies the object's internal data and representation and defines the operations the object can perform
- Class can be rendered as





Slide 24

- Class inheritance



P-24

Patterns I

Slide 25

- An *abstract class* is one whose main purpose is to define a common interface for its subclass(it defers in implementation to operations defined in subclass).
  - It cannot be instantiated
- The operations that an abstract class declares but doesn't implement are called *abstract operations*.

  (the names in italic indicates a class or operation as abstract)
- Classes that are not abstract are called *concrete classes*.
- A class may *override* an operation defined by its parent class
- A *Mixin class* is a class that's intended to provide an optional interface or functionality to other classes(it is an abstract class that can be instantiated).

Slide 26

- Class Vs Interface inheritance
  - The difference between object's class and type is class defines the object's internal state and the implementation of its operations, in contrast type refers to its interface.
  - Class inheritance defines an object's implementation in terms of another object's implementation(mechanism for code and representation sharing), in contrast interface inheritance describes when an object can be used in place of another.
- Programming to an interface, not an Implementation
  - It is the principle of reusable OOD.

Slide 27

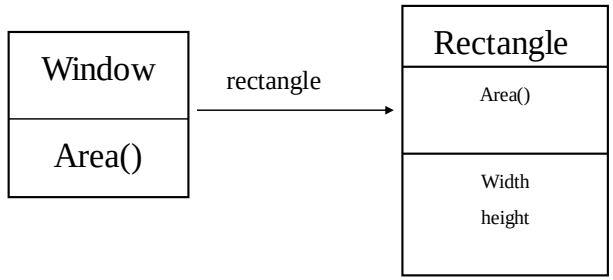
5) Putting reuse mechanisms to work

- It is easy to know class, interface, object etc., but applying them to build flexible, reusable s/w and Design Patterns can show how
- Inheritance versus Composition
  - White-box reuse – with inheritance, the internals of parent classes are often visible to subclasses (visibility).
  - Black-box reuse – object composing, i.e., objects that composed have well defined interfaces
- Disadvantages with class inheritance
  - Change the implementations inherited from parent classes at run-time is not possible.
  - Parent classes often define at least part of their subclasses physical representation.

P-27

Slide 28

- Delegation
- It is a way of making composition as powerful for reuse as inheritance
- The following diagram depicts the Window class delegating its Area operation to a Rectangle instance



Slide 29

- Advantage of delegation – it makes it easy to compose behaviors at run-time and to change the way they are composed
- Disadvantage – it shares with other techniques that make software more flexible through object composition (I.e., understanding s/w, runtime inefficiencies)

- Several design patterns use delegation.
- The **State (338), Strategy (349),** and **Visitor (366)** patterns depend on it.

Slide 30

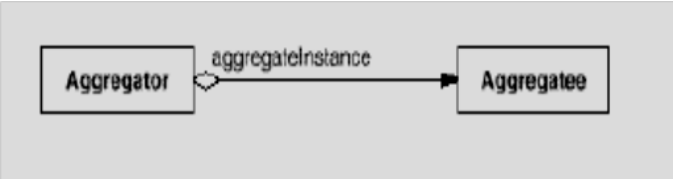
6) Relating run-time and compile time structures

- compile time structure is static and runtime structure is dynamic.
  - **Acquaintance or Association**
  - **Acquaintance implies that an object merely knows of another object.**
- **Aggregation**
- Aggregation implies that one object owns or is responsible for another object
- :

P-30

Slide 31

In our diagrams, a plain arrowhead line denotes **acquaintance**. An arrowhead line with a diamond at its base denotes **aggregation**



Many design patterns capture the distinction between compile-time and run-time structures explicitly.
Composite(183) and Decorator (196) are especially useful for building complex run-time structures.
Observer (326) involves run-time structures that are often hard to understand unless you know the pattern

P-31

Slide 32

7) Designing for change

- A design that doesn't take change into account risks major redesign in the future.
- We must consider how the system might need to change over its lifetime
- These changes leads to class redefinition and reimplementation, client modification, and retesting
- Some common causes of redesign
  - Creating an object by specifying a class explicitly. (Abstract Factory, Factory Method, Prototype)
  - Dependence on hardware and software platform. (Chain of responsibility, Command)
  - Dependence on object representations or implementations. (Abstract Factory, Bridge)
  - Algorithmic dependencies
  - Tight coupling. etc.,

P-32

Slide 33

Role played by Design patterns

- Application Programs
  - When building application program such as a document editor or spreadsheet then internal reuse, maintainability, and extensions are high priorities.

    design patterns makes it easy to have all these without any side effects.
- Toolkits
  - A toolkit is a set of related and reusable classes designed to provide useful, general-purpose functionality.
- Frameworks
  - A framework is a set of cooperating classes that makeup a reusable design for a specific class of software.

Slide 34

How to select a Design Pattern

- Consider how design patterns solve design problems.
- Scan intent sections.
- Study how patterns interrelate
- Study patterns of like purpose
- Examine a cause of redesign
- Consider what should be variable in your design.

Slide 35

How to use a Design Pattern

- Read the pattern once through for an overview
- Go back and study the structure, participants, and collaborations sections.
- Look at the sample code section to see a concrete example of the pattern in code
- Choose names for pattern participants that are meaningful in the application context
- Define the classes
- Define application- specific names for operations in the pattern.
- Implement the operations to carry out the responsibilities and collaborations in the pattern