# UNIT - I





# UNIT - I
# INTRODUCTION TO DESIGN PATTERNS AND OBSERVER PATTERN

## SYLLABUS

Basics of Design patterns, Description of design patterns, Catalog and organization of catalog, design patterns to solve design problems, selection of design pattern, Use of design patterns.

### INTRODUCTION

The concept of a pattern is used in software architecture is borrowed from the field of (building) architecture, in particular from the writings of architect Christopher Alexander.

A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts and presents a well-proven generic scheme for its solution.

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

### INTRODUCTION TO DESIGN PATTERNS AND OBSERVER PATTERN

**Q.1. What is a design pattern?**

**Ans. Design pattern :**

- Patterns are the solutions to a problem in a context. It is the way of describing higher level perspective on problem and on the process of design and object orientation.
- When building more complex computer systems, problems of construction rather than problems of analysis.
- The problems of construction are solved by designing programming solutions for such problems in the context of the computer application we are trying to build.
- Some constructional problems occur over and over again across a wide range of different computer applications.
- We can design a generic solution to such repeating problems, and then try to adjust such a generic solution to the specific need of the application we are currently building. Such generic solutions are usually referred to as design patterns.
- Christopher Alexander first introduced patterns to encode knowledge and experience in designing buildings. According to 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 we can use this solution many times over, without ever doing it the same way twice".
- Thus, a design pattern is the core of a solution to a problem in context. The solution can be applied in different situations, and has to be adapted to fit the needs of the specific situation.

**Q.2. What is a objective of design pattern?**

**Ans.** The objectives of design pattern are as follows :

**(i)** Solutions are reusable. There are recurring patterns of classes and communicating objects that solve specific design problems and make these designs more flexible and reusable.

**(ii)** A design pattern systematically names, explains and evaluates an important and recurring design in object-oriented systems.

**(iii)** It is possible to collect these patterns into a catalog for designers to use to select and evaluate alternatives.

**(iv)** It establishes common terminology to improve communications within teams.

**(v)** Improve individual learning and team learning.

**(vi)** Improve the modifiability of code.

**(vii)** It facilitates adoption of improved design alternatives, even when patterns are not used explicitly.

**(viii)** It discovers alternatives to large inheritance hierarchies.

**Q.3. What are the characteristics of pattern?**

**Ans.** The characteristics of pattern are as follows :

(1) A pattern describes a solution to a recurring problem that arises in specific design situations.

(2) Patterns are not invented; they are distilled from practical experience.

(3) Patterns describe a group of components (classes or objects), how the components interact, and the responsibilities of each component. That is, they are higher-level abstractions than classes or objects.

(4) Patterns provide a vocabulary for communication among designers. The choice of a name for a pattern is very important.

(5) Patterns help document the architectural vision of a design. If the vision is clearly understood, it will less likely be violated when the system is modified.

(6) Patterns provide a conceptual skeleton for a solution to a design problem and, hence, encourage the construction of software with well-defined properties.

(7) Patterns are building blocks for the construction of more complex designs.

(8) Patterns help designers manage the complexity of the software. When a recurring pattern is identified, the corresponding general solution can be implemented productively to provide a reliable software system.

**Q.4. What is object-oriented design pattern?**

**Ans. Object-oriented design pattern :**

- The object-oriented paradigm is centered on the concept of the object.
- Objects are defined by their responsibilities. Objects simplify the tasks of programs that use than by being responsible for themselves.
- Design patterns in programming were first introduced by the Gang of Four (Gamma, Helm, Johnson, Vlissides). They referred to design patterns always as to object-oriented design patterns, i.e., design patterns that occur in building object-oriented computer systems.
- Thus, object-oriented design patterns might be defined as descriptions of communicating objects and classes that are customized to solve a general (object-oriented) design problem in a particular context.
- An object-oriented design pattern is not a primitive building block, such as linked lists, or hash-tables that can be encoded in classes and reused in a wide range of applications.
- An object-oriented design pattern is not a complex, domain specific design for an entire application or subsystem.
- An object-oriented design pattern just describes a recurring object-oriented design structure.
- Object-oriented design pattern is responsible to name, abstract, and identifies the key aspect (classes, objects and their relations) of a common design structure. Object-oriented design pattern is responsible to creates a reusable object-oriented design.
- Object-oriented design pattern is responsible to focuses on a particular object-oriented design by describing when it applies, whether it can be applies in view of other design constraints, and the consequences and trade-offs of its use.
- Object-oriented design pattern is responsible to implements the design idea in an object-oriented programming language.

**Q.5. What are the basic elements of design pattern?**

**Ans.** The basic elements of a design pattern are as follows :

**(1)** Pattern Name.

**(2)** Problem.

**(3)** Solution.

**(4)** Consequences.

**(1) Pattern Name :**

- The pattern name is used to describe a design problem, its solutions and consequences in a word or two.
- Naming a pattern immediately increases the design vocabulary of programmers.
- Having a vocabulary for patterns enables to talk about patterns with other programmers, in the documentation, etc.

**(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.
- It might describe class or object structures that are symptomatic of an inflexible design.
- Sometimes the problem includes also a list of conditions that must be met before it makes sense to apply the pattern.







**(3) Solution :**

- The solution describes the elements that make up the design, their 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.
- Instead, the pattern provides an abstract description of a design problem and how a general arrangement of elements (classes and objects) solves it.

**(4) Consequences :**

- The consequences are the results and trade-offs of applying the pattern. They may address language and implementation issues as well.
- Since reuse is often a factor in object-oriented design, the consequences of a pattern include its impact on a system's flexibility, extensibility, or portability.

## BASICS AND DESCRIPTION OF DESIGN PATTERN

**Q.6. What are the basic template of design pattern?**

**Ans. Basic template of design pattern :**

- The Gang of Four used a consistent format to describe patterns. They developed a template for describing a design pattern.
- The template lent a uniform structure to the information and made design patterns easier to learn, compare and use. This template describes a design pattern with the following :

**(1) Pattern Name and Classification :**

It is important because it becomes part of our design vocabulary.

**(2) Intent :**

It describes what does the design pattern do?, What is its rationale and intent? and What particular design issue or problem does it address?

**(3) Motivation :**

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

**(4) Applicability :** Applicability describes where we should use the pattern and how can we recognize these situations.

**(5) Structure :** A graphical representation of the classes in the pattern using a notation such as Object Modeling Technique (OMT) or UML to illustrate sequences of requests and collaborations.

**(6) Participants :**

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

**(7) Collaborations :**

It states, how the participants collaborate to carry out their responsibilities.

**(8) Consequences :**

How does the pattern support its objectives, What are the trade-offs and results of using the pattern and what aspect of system structure does it let us vary independently describes in consequences of the design pattern.

**(9) Implementation :**

What pitfalls, hints, or techniques should one be aware of when implementing the pattern and are there any language-specific issue arise, while designing a pattern.

**Q.7. What are the classifications of design pattern?**

**Ans.** The classifications of design pattern are as follows :

**(i)** Creational patterns concern the process of object creation.

**(ii)** Structural patterns deal with the composition of classes and objects.

**(iii)** Behavioural patterns characterize the ways in which classes and objects interact and distribute responsibility.

- Scope, 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. Patterns can have creational, structural or behavioural purpose :

| Purpose | Design Patterns | Scope |
|---|---|---|
| Creational | Abstract Factory | Object |
| | Builder | Object |
| | Factory Method | Class |
| | Prototype | Object |
| | Singleton | Object |

| Purpose | Design Patterns | Scope |
|---|---|---|
| Structural | Adapter | Class |
| | Bridge | Object |
| | Composite | Object |
| | Decorator | Object |
| | Facade | Object |
| | Flyweight | Object |
| | Proxy | Object |
| 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 |

## CATALOG AND ORGANIZATION OF CATALOG

**Q.8. Explain in details the catalog/classification or categories of the design patterns.**

**Ans.**

- Design pattern's vary in their granularity and level of abstraction because there are many design patterns we need a way to organize them.
- It classifies design patterns so that we can refer to families of related pattern.
- The classification helps us to learn the patterns in the catalog faster and it can direct efforts to find new pattern as well.

**(1) Creational Patterns :**

It has two types :

**(a)** Class Creational.

(b) Object Creational.

**(a) Class Creational Patterns :**

It has one category factory method.

**(i) Factory Method :**

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

Varying subclass of object that is instantiated.

**(b) Object Creational Patterns :**

It has four categories :

**(i) Abstract Factory :** Provide an interface for creating families of related or dependent objects without specifying their concrete classes. Varying: Families of product objects.

**(ii) Builder :** Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Varying : How a composite object gets created.

**(iii) Prototype :** Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Varying : Class of object that is instantiated.

**(iv) Singleton :** Ensure a class only has one instance, and provide a global point of access to it. Varying : The sole instance of a class.

**(2) Structural patterns :**

It has two types :

**(a)** Class Structural.

(b) Object Structural.

**(a) Class Structural Patterns :**

It has one category :

**(i) Adapter (class) :**

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

Varying : Interface to an object.

**(b) Object Structural Patterns :**

It has following categories :





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

Varying : Interface to an object.

**(ii) Bridge :** Decouple an abstraction from its implementation so that the two can vary independently.

Varying : Implementation of an object.

**(iii) Composite :** Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Varying : Structure and composition of an object.

**(vi) Decorator :** Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Varying : Responsibilities of an object without subclassing.

**(v) Façade :** Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to user.

Varying : Interface to a subsystem.

**(vi) Flyweight :** Use sharing to support large numbers of fine-grained objects efficiently.

Varying : Storage costs of objects.

**(vii) Proxy :** Provides a surrogate or placeholder for another object to control access to it.

Varying : How an object is accessed; its location.

**(3) Behavioral patterns :**

It has two types :

**(a)** Class Behavioral.

**(b)** Object Behavioral.

**(a) Class Behavioral Patterns :**

It has two categories :

**(i) Interpreter :**

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

Varying : Grammar and interpretation of a language.

**(i) 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 algorithms structure.

Varying : Steps of an algorithm.

**(b) Object Behavioral Patterns :** It has following categories :

**(i) 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.

Varying : Object that can fulfill a request.

**(ii) Command:** Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Varying : When and how a request is fulfilled.

**(iii) Iterator :** Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Varying : How an aggregate's elements are accessed, traversed.

**(vi) 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 let us vary their interaction independently.

Varying : How and which objects interact with each other.

**(v) Memento :** Without violating encapsulation, capture and externalize an objects internal state so that the object can be restored to this state later.

Varying : What private information is stored outside an object, and when.

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

Varying : Number of objects that depend on another object; how the dependent objects stay up to date.

**(vii) State :** Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

Varying : States of an object.

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

Varying : An algorithm.

**(ix) 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 operate.

Varying : Operations that can be applied to object(s) without changing their class(es).

| | | Purpose | | |
|---|---|---|---|---|
| | | Creational | Structural | Behavioral |
| Scope | Class | Factory Method | Adapter | Interpreter , Template Method |
| | Object | Abstract Factory, Builder Prototype Singleton | Adapter, Bridge Composite Decorator Façade, Proxy | Chain or Responsibility Command Iterator Mediator Memento Flyweight Observer, State Strategy, Visitor. |

Fig. (a) Classification of design pattern

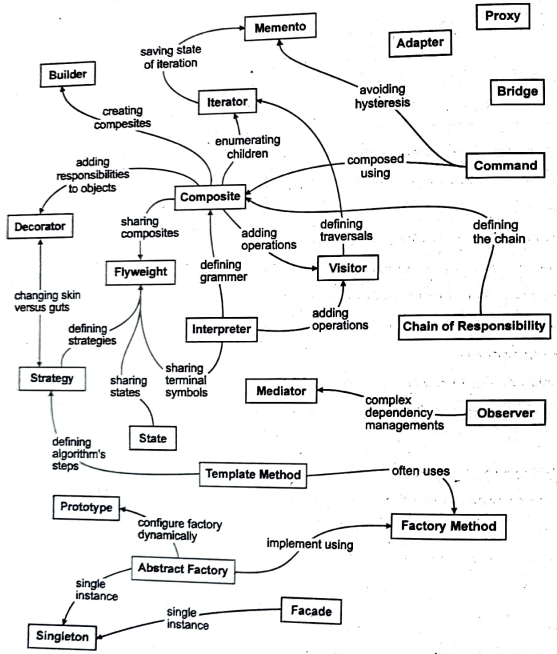


Fig. (b) Relationship between catalog methods





## DESIGN PATTERNS TO SOLVE DESIGN PROBLEMS

**Q.9. Explain in details design problems with in design patterns.**

**Ans.** Design problems with in design patterns can be solved in four steps :

**(1) Finding appropriate objects :**

(a) Object-oriented programs are made up of objects.

(b) The hard part about object-oriented design is decomposing a system into objects.

(c) 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 for example.

**(2) Determining object granularity :**

(a) Usually, objects vary in size and number. They can represent everything down to the hardware or all the way up to entire applications.

(b) 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.

**(3) Specifying object interfaces :**

(a) Design patterns help programmers to define interfaces by identifying their key elements and the kind of data that get sent across an interface.

(b) 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) Designing for Change :**

(a) Designing a system that is robust to changes is a rather hard task to do.

(b) Design patterns can ensure that a system can change in specific ways.

(c) 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.

- 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 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.

**Q.10. State and explain advantages of the design pattern.**

**Ans.** Some advantages of design patterns are as follows :

**(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.
- Design patterns make a system less complex, since we can talk about it in the terms form different design patterns rather then in terms of programming languages.

**(2) Documentation aid :**

- Describing a system by applying a common design vocabulary makes this system easier to understand.
- Having such a common vocabulary means you don't have to describe the whole design pattern; programmers just can name it and expect that the reader already knows what a specific pattern means.
- Therefore, writing the documentation can become much more interesting and easier.

## SELECTION OF DESIGN PATTERN

**Q.11. Explain in details the selection criteria for design pattern.**

**Ans.** The following criteria is responsible for the selection of appropriate design pattern:

(1) Consider how design patterns solve design problems :

(i) Find appropriate objects.

(ii) It determine object granularity.

(iii) It specify object interfaces.

(2) Scan intent sections.

(3) Study how patterns interrelated.

(4) Study patterns of like purpose.

(i) Study similarities and differences of the patterns.

(5) Examine a cause of redesign.

(i) Consider what might force a change to our design.

(6) Consider what should be variable in our design.

(i) Consider what we want to be able to change without redesign - encapsulate the concept that varies.

## USE OF DESIGN PATTERNS

**Q.12. How to use a design pattern?**

**Ans.** The steps of using a design pattern can be summarized as follows :

(1) Identify a problem.

(2) Find a design pattern.

(3) Understand the design pattern.

(4) Look at the example.

(5) Fit the pattern.

(6) Implement the pattern.

**(1) Identity a problem :**

- It is very easy to design software in an ad-hoc fashion by adding functions and classes whenever the need arises. The problem with this is that it doesn't work well for large projects or when multiple develop on the same project.
- The way to remedy this is to think about how we want to do something before we actually do it. By doing this, we can look at the trade-offs and shortcomings whenever we make a decision about the structure of our code.

**(2) Find a design pattern :**

- If the problem has been accurately understood, the next thing to do is to find a design pattern that will either solve the problem or lessen the impact of it.
- This requires knowing some design patterns or at least knowing the categories that they fall in.
- When we know that the problem we are solving deals with object instances and instantiating them effectively, it would help to look at creational design patterns.

**(3) Understand the design pattern :**

- Once we have selected the design pattern that will help us. It is well worth our time to understand it fully.

Each design pattern on this site has following types :

(i) A definition.

(ii) List of alternative names (if any).

(iii) A diagram showing a UML representation of the pattern.

(iv) A common problem the pattern solves.

(v) A wrong solution.

(vi) A correct solution using the pattern.

(vii) Consequences of using the pattern.

(viii) Example code.

- After understanding it, if it doesn't actually solve our problem, we may have to go to the previous step of finding one that might solve it.

**(4) Look at the example :**

- The example will provide a self contained use case of the pattern in action.
- It will show how to structure the classes/ interfaces and it will show the functions/methods that will need to interact together.
- The example can be also used as a template for what we wish to do. The next step will look what might happen if our problem requires a lot more effort to solve. We may wish to just use it as a reference and make further modifications based on our specific use case.

**(5) Fit the pattern :**

- The problem that we have might not fit the given problem exactly. For whatever the reason may be, it may be required to make certain changes to either our structure or to the design pattern.





- It may be the case that we are already using another design pattern as well. We may have to make them work together and do other things to improve the design of our software.

**(6) Implement the pattern :**

- The implementation of a design pattern will vary greatly depending on the language, package layout, and other factors of the software environment.
- However, the core idea does not change. An example of this would be implementing the Observer pattern in Rust as compared to Java.
- In Java, we could use polymorphism or an interface when creating your observer pattern.

**Q.13. Explain benefits of design pattern.**

**Ans.** The benefits of design pattern are as follows :

(1) Capture expertise and make it accessible to non-experts in a standard form.

(2) Facilitate communication among developers by providing a common language.

(3) Make it easier to reuse successful designs and avoid alternatives that diminish reusability.

(4) Facilitate design modifications.

(5) Improve design documentation.

(6) Improve design understandability.

**Q.14. Explain the facade design pattern with example.**

**Ans.** **Façade design pattern :**

- When designing good programs, programmers usually attempt to avoid excess coupling between module/classes. It decouple client from complex system
- It makes the task of accessing a large number of modules much simpler by providing an additional interface layer.
- Using this pattern helps to simplify much of the interfacing that makes large amounts of coupling complex to use and difficult to understand: It hides the complexities of the system and provides an interface to the client from where the client can access the system.
- For example, the Java interface JDBC can be called a façade.
- Façade, a single class to access a collection of classes.

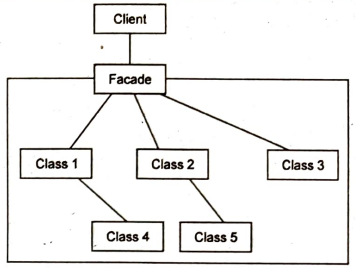


**Fig. (a) Facade design pattern**

**Classes :**

(i) One client class.

(ii) The facade class : It has a little of code to call lower layers most of the time.

(iii) The classes underneath the facade.

**Advantages/Disadvantages :**

- Decouple the interfacing between many modules or classes.
- One possible disadvantage to this pattern is that you may lose some functionality contained in the lower level of classes, but this depends on how the facade was designed.
- For example, consumers encounter a facade when ordering from a catalog. The consumer calls one number and speaks with a real or virtual customer service representative.
- The customer service representative acts as a Facade, providing an interface to the order fulfilment department, the billing department, and the shipping department.

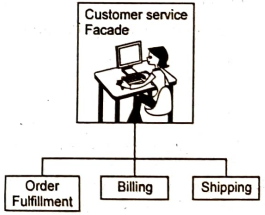


**Fig. (b) Customer service facade**

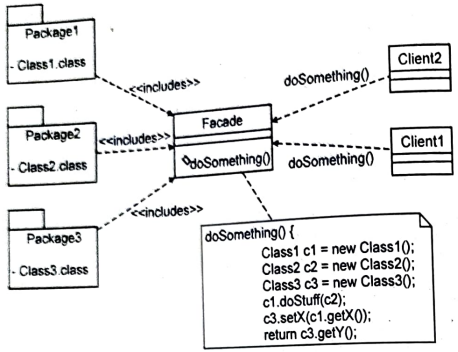


**Fig. (c) Interact packages**

- Facade : Interacts packages 1, 2, and 3 with the rest of the application.
- Clients : The objects using the facade pattern to access resources from the packages.
- Packages : Software library / API collection accessed through the facade Class.

**Example :**

The following example hides the parts of a complicated calendar API behind a more user friendly facade.

```
import java.text.*;
import java.util.*;
/** "Facade" **/ class
UserfriendlyDate
{
    Calendar cal = Calendar.getInstance();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    public UserfriendlyDate (String isodate_ymd) throws ParseException
    {
        Date date = sdf.parse(isodate_ymd);
        cal.setTime(date);
    }
    public void addDays (int days) {
        cal.add (Calendar. DAY_OF_MONTH, days);
    }
    public String toString() {
        return sdf.format(cal.getTime());
    }
}
```





```
/** "Client" **/
class FacadePattern
{
    public static void main(String[ ] args) throws ParseException
    {
        UserfriendlyDate d = new UserfriendlyDate("1980-08-20");
        System.out.println ("Date: " + d.toString( ));
        d.addDays(20);
        System.out.println ("20 days after: " + d.toString( ));
    }
}
```

## *POINTS TO REMEMBER :*

*(1) The problems of construction are solved by designing programming solutions for such problems in the context of the computer application we are trying to build.*

*(2) 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 we can use this solution a million times over, without ever doing it the same way twice.*

*(3) The pattern name is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.*

*(4) The problem describes when to apply the pattern. It explains the problem and its context.*

*(5) The solution describes the elements that make up the design, their relationships, responsibilities, and collaborations.*

*(6) Some advantages of design patterns are common design vocabulary and documentation aid.*