CBCS (15IS72)

Seventh Semester B.E. Degree Examination,
Dec.2018/Jan.2019

Software Architecture and Design Patterns

Time: 3 hrs. **Max. Marks: 80**

MODULE 1

1a. What is design pattern? How patterns and frame works are different? (6Marks)

Design pattern:

"A proven solution to a common problem in a specified context"

Example: We can light a candle if light goes out at night

Essential Elements:

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

The **problem** describes when to apply the pattern.

The **solution** describes the elements that make up the design, their relationships, responsibilities, and collaborations. The pattern provides an abstract description of a design problem and how a general arrangement of classes and objects solves it.

The **consequences** are the results and trade-offs of applying the pattern.

Example Pattern:

Pattern Name – Iterator

Problem – How to serve Patients at a Doctor's Clinic

Solution – Front-desk manages the order for patients to be called

- By Appointment
- By Order of Arrival
- By Extending Gratitude
- By Exception

Consequences

- Patient Satisfaction
- Clinic's Efficiency
- Doctor's Productivity

Framework:

A **framework** is a set of cooperating classes that makeup a reusable design for a specific class of software.

For example, a framework can be geared toward building graphical editors for different domains like artistic drawing, music composition, and mechanical.

Differences between framework and design pattern

Patterns and frameworks differ in three ways

1. ***Design patterns are more 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 ca n be written down in programming languages and not only studied but executed and reused directly.
- ✓ Design patterns also explain the intent, trade-offs, and consequences of a design.

2. ***Design patterns are smaller architectural elements than frameworks***

A typical framework contains several design patterns, but the reverse is never true.

3. ***Design patterns are less specialized than frameworks***

- ✓ Frameworks always have a particular application domain.
- ✓ In contrast, the design patterns in this catalo g can be used in nearly any kind of application.

1 b. Explain Object-Oriented Development. (4Marks)

- **First computers** are developed mainly to automate a well-defined process (i.e., an algorithm) for numerical computation, as systems became more complex; its effectiveness in developing solutions became suspect.
- software applications developed in later years had two differentiating characteristics:
  - ✓ Behavior that was hard to characterize as a process
  - ✓ Requirements of reliability, performance, and cost that the original developers did not face
- The ‘process-centred’ approach to software development used what is called top down functional decomposition.
  - ✓ The first step in such a design was to recognize what the process had to deliver which was followed by decomposition of the process into functional modules.

- ✓ Structures to store data were defined and the computation was carried out by invoking the modules, which performed some computation on the stored data elements.
- ✓ The life of a process-centred design was short because changes to the process specification required a change in the entire program.
- ✓ This resulted in an inability to reuse existing code without considerable overhead

- Thus engineering disciplines started soon after, and the disciplines of 'software design' and 'software engineering' came into existence.
- The reasons for this success are easy to see:
  - ✓ Easily understandable designs
  - ✓ Similar (standard) solutions for a host of problems
  - ✓ An easily accessible and well-defined 'library' of 'building-blocks'
  - ✓ Interchangeability of components across systems,
  - ✓ A software component is also capable of storing data,
  - ✓ The components can also communicate with each other as needed to complete the process

1 c. What are the key concepts of object oriented design? **(6Marks)**

1. The Central Role of Objects
2. The notion of a Class
3. Abstract specification of functionality
4. A language to define the System
5. Standard Solutions
6. An analysis process to model a system
7. The notions of extendibility and adaptability

2a. What does the design pattern do? How to select a design patterns. (6Marks)

"A 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"

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 how design patterns solve design problems

Determine object granularity; specify object interfaces, and several other ways in which design patterns solve design problems.

Scan Intent sections

Read through each pattern's intent (purpose) to find one or more that should relevant to your problem.

Study how patterns interrelate

Studying these relationships can help direct you to the right pattern or group of patterns.

Study patterns of like purpose

Study only those patterns which are of specific purposes (creational patterns, structural patterns, and behavioural patterns).

Examine a cause of redesign

Look at the patterns that help you avoid the causes of redesign

Consider what should be variable in your design

Consider what you want to be able to change without redesign.

2 b What pitfalls, hints. or techniques should be aware of, when implementing the pattern? (6Marks)

The ***first criterion,*** called **purpose,** reflects what a pattern does.

1 **Creational patterns** concern the process of object creation.
2 **Structural patterns** deal with the composition of classes or objects.
3 **Behavioral patterns** characterize the ways in which classes or objects interact and distribute responsibility.

The ***second criterion***, called **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— fixed at compile-time.
- Object patterns deal with object relationships, which can be changed at run-time and are more dynamic.

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method (107)	Adapter (class) (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (object) (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Flyweight (195) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Observer (293) State (305) Strategy (315) Visitor (331)

Table 1.1: Design pattern space

2c. Describe the benefits to manipulating object solely in terms of the interface defined by the abstract classes. (4 Marks)

- The class defines the object's internal state and the implementation of its operations.
- In contrast, an object's type only refers to its interface—the set of requests to which it can respond.
- An object can have many types, and objects of different classes can have the same type.
- An object is an instance of a class; we imply that the object supports the interface defined by the class.
- Class inheritance defines an object's implementation in terms of another object's implementation.
- Interface inheritance (or subtyping) describes when an object can be used in place of another.

Examples: Chain of Responsibility, Composite pattern, Command, Observer, State, and Strategy.

Module 2

3 a "The analysts neéd to learn the existing system and the requirements" justify. (05 marks)

The major role of the analysts is to address the basic question: what should the system do? Requirements are often simple and any clarifications can be had via questions in the classroom, e- mail messages, etc. However, as in the case of the classroom assignment, there are still two parties: the user community, which needs some system to be built and the development people, who are assigned to do the work.

Requirements can be classified into two categories:

Functional requirements: These describe the interaction between the system and its users, and between the system and any other systems, which may interact with the system by supplying or receiving data.

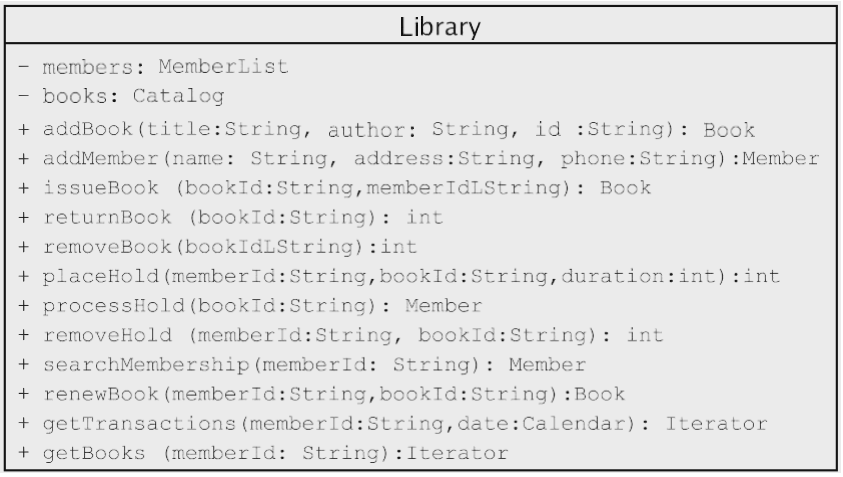
Non-functional requirements: Any requirement that does not fall in the above category is a non-functional requirement. Such requirements include response time, usability and accuracy. Sometimes, there may be considerations that place restrictions on system development; these may include the use of specific hardware and software and budget and time constraints.

3b. What are the Guidelines to remember when writing Use Cases? (04 Mark)

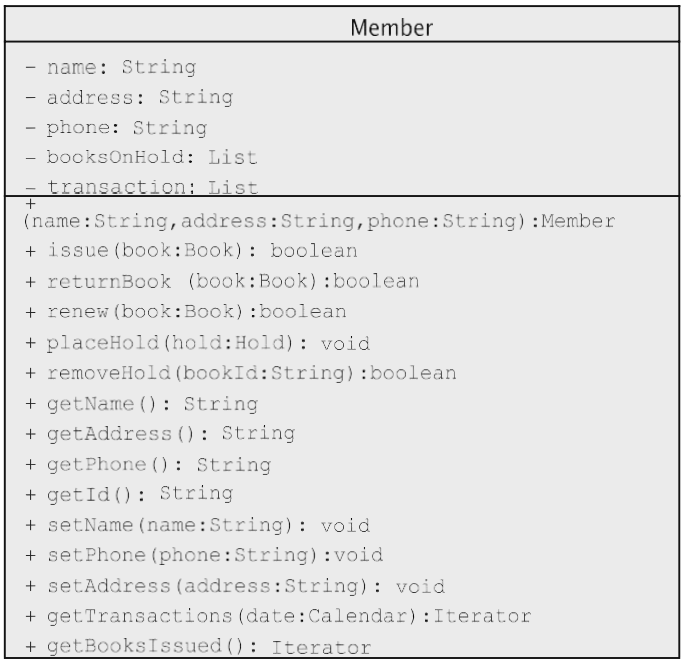
- A use case must provide something of value to an actor or to the business.
- Use case should be functionally cohesive, i.e., they encapsulate a single service that the system provides.
- Use case should be temporally cohesive. This notion applies to the time frame over which the use case occurs.
- If a system has multiple actors, each actor must be involved in at least one, and typically several use cases.
- The model that we construct is a set of use cases.
- Use cases are written from the point of view of the actor.
- A use case describes a scenario.
- Use cases change over the course of system analysis.

3 c Draw class diagram for library and class diagram for Member of Library (07 Marks)

1. Class diagram for Library



2. Class diagram for Member of Library



4a. Explain the major steps in analysis phase (6 Marks)

The process could be split into three activities:

- Gather the requirements: this involves interviews of the user community, reading of any available documentation, etc.
- Precisely document the functionality required of the system.
- Develop a conceptual model of the system, listing the conceptual classes and their relationships.

Gathering the Requirements

The purpose of *requirements analysis* is to define what the new system should do. Since the system will be built based on the information garnered in this step, any errors made in this stage will result in the implementation of a wrong system. Once the system is implemented, it is expensive to modify it to overcome the mistakes introduced in the analysis stage.

Imagine the scenario when you are asked to construct software for an application. The client may not always be clear in his/her mind as to what should be constructed.

First reason for this is that it is difficult to imagine the workings of a system that is not yet built.

Second reason Incompleteness and errors in specifications can also occur because the client does not have the technical skills to fully realize what technology can and cannot deliver

Third reason for omissions is that it is all too common to have a client who knows the system very well and consequently either assumes a lot of knowledge on the part of the analyst or simply skips over the 'obvious details'.

4 b Compare functional requirements versus non-functional requirements. (4Marks)

ID	*Functional Requirement*	*ID*	*Non-functional Requirement*
R.1.	Improving P2P application performance while reducing the network traffic	R.10.	Easy deployment
R.2.	Incentive-compatibility for all player involved	R.11.	Extensibility for new overlay applications
R.3.	Support of different overlay applications	R.12.	Scalability in terms of large end-user population.
R.4.	Interface supporting various optimization schemes	R.13.	Efficiency of the SIS operation
R.5.	QoS support	R.14.	Robustness of the SIS against malicious behavior and against dynamic behavior
R.6.	Different operations: user anonymity mode (free services), user aware mode (premium services)	R.15.	Security: Secure communication between SIS entities and between SIS and overlay application
R.7.	Inter-domain support	R.16.	Standard compliance: The SIS shall use and based on standard protocols where applicable.
R.8.	OAM (Operation and Management) support	R.17.	Transparency: The SIS shall not apply Deep Packet Inspection (DPI).
R.9.	Mobile network support		

4 c Describe conceptual Software and implantation classes (6Marks)

A conceptual model is a representation of a system that uses concepts and ideas to form said representation. Conceptual modeling is used across many fields, ranging from the sciences to social economics to software development

When using a conceptual model to represent abstract ideas, it's important to distinguish between a model of a concept versus a model that is conceptual. That is to say, a model is intrinsically a thing unto itself, but that model also contains a concept of what that model represents — what a model is, as opposed to what a model represents.

An implementation class is a class that may have attributes, associations, operations, and methods. An implementation class defines the physical implementation of objects of a class. For example, if you were to implement our classes in a database management system, the Worker class might be implemented as an employee table, the WorkProduct class might be implemented as an artifact table, and the UnitOfWork class might be implemented as work order table. An implementation class is shown as a class marked with the implementationClass keyword. The three implementation classes just mentioned are shown in following Figure.

Fig: Implementation classes

Module 3

5 a How classes and objects are composed to form larger structure? (4 Marks)

- ✓ Structural patterns are concerned with how classes and objects are composed to form larger structures.
- ✓ Structural class patterns use inheritance to compose interfaces or implementations.

Example: Multiple inheritances mix two or more classes into one.

Popular structural design patterns include**:**

1. Adaptor
2. Bridge
3. Composite
4. Decorator
5. Façade
6. Flyweight
7. Proxy

5b. Explain intent, Motivation, Applicability, Structure, Participants, Collaboration, Consequences and Implementation of Decorator Pattern. (8 Marks)

Intent: To convert the interface of one class into another interface that the client expects. Adapter pattern allows two incompatible classes to communicate with one another.

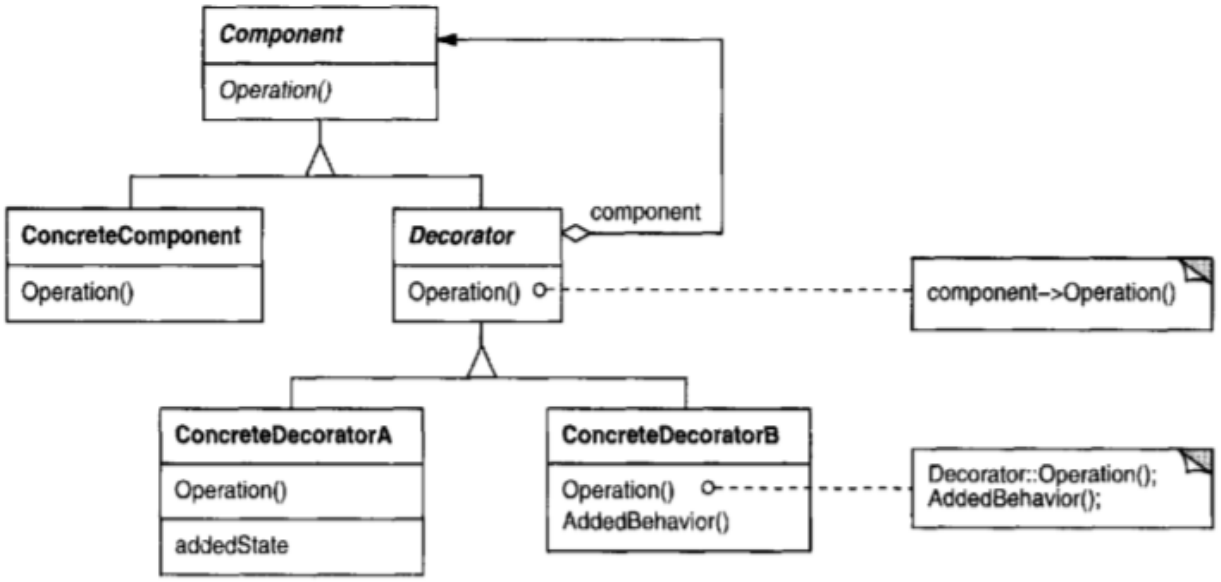
Motivation:

- ✓ Sometimes we want to add responsibilities to individual objects , not to an entire class .
- ✓ A graphical user interface toolkit, for example, should let you add properties like borders or behaviors like scrolling to any user interface component.
- ✓ One way to add responsibilities is with inheritance.
- ✓ Inheriting a border from another class puts a border around every subclass instance,this is inflexible. A client can't control how and when to decorate the component with a border.
- ✓ A more flexible approach is to enclose the component in another object that adds the border. The enclosing object is called a decorator.
- ✓ The decorator forwards requests to the component and may perform additional actions (such as drawing a border) before or after forwarding.

Applicability:

1. To add additional responsibilities to individual objects dynamically and transparently, that is, without affecting other objects.
2. For responsibilities that can be withdrawn.
3. When extension by sub classing is impractical.

Structure:



Participants

- Component (VisualComponent) - defines the interface for objects that can have responsibilities added to them dynamically.
- ConcreteComponent (TextView) - defines an object to whic h additional responsibilitie s can be attached.
- Decorator - maintains a reference to a Component object and defines an interface that conforms to Component's interface.
- ConcreteDecorator (BorderDecorator, ScrollDecorator) - Adds responsibilities to the component

Collaborations: Decorator forwards requests to its component object. It may optionally perform additional operations before and after forwarding the request.

Consequences: The decorator pattern has atleast two key benefits and two liabilities:

1. More flexibility than static inheritance: The Decorator pattern providesa more flexible way to add responsibilities to objects than can be had withstatic (multiple) inheritance.

2. Avoids feature-laden classes high up in the hierarchy: Decorator offersa pay-as-you-go approach to adding responsibilities. Instead of trying tosupport all foreseeable features in a complex, customizable class, you candefine a simple class and add functionality incrementally with Decoratorobjects.

3. A decorator and its component aren't identical: A decorator acts as atransparent enclosure. But from an object identity point of view, adecorated component is not identical to the component itself.

4. Lots of little objects: A design that uses Decorator often results in systems composed of lots of little objects that all look alike. The objects differ only in the way they are interconnected, not in their class or in the value of their variables.

Implementation: Following issues should be considered when applying the decorator pattern:

1. Interface conformance: A decorator object's interface must conform to the interface of the component it decorates. Concrete Decorator classes must therefore inherit from a common class.

2. Omitting the abstract Decorator class: There's no need to define an abstract Decorator class when you only need to add one responsibility.

3. Keeping Component classes lightweight: To ensure a conforming interface, components and decorators must descend from a common Component class. It's important to keep this common class lightweight; that is, it should focus on defining an interface, not on storing data.

4. Changing the skin of an object versus changing its guts: We can think of a decorator as a skin over an object that changes its behavior. An alternative is to change the object's guts. The Strategy (349) pattern is a good example of a pattern for changing the guts.

5c. Mention few common situations in which the Proxy pattern is applicable. (04 Marks)

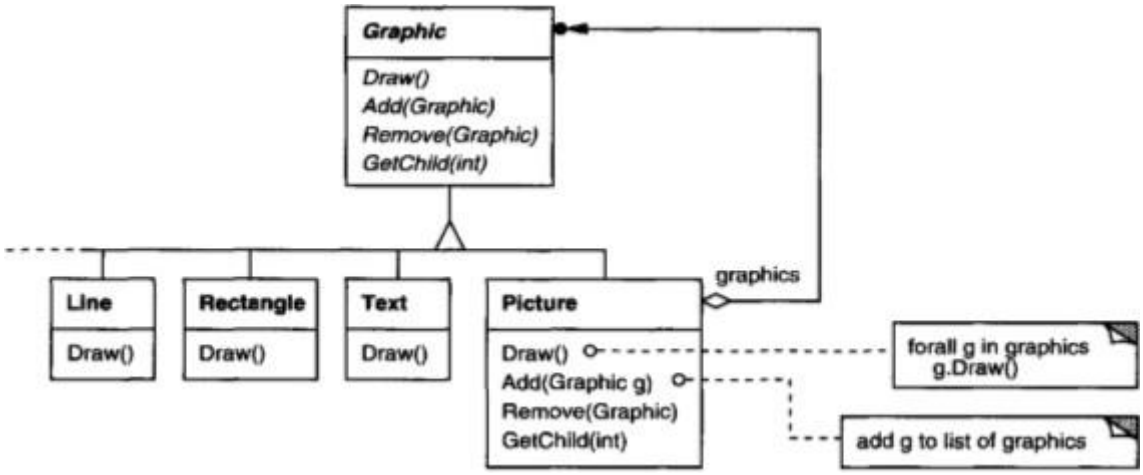
- A remote proxy provides a local representative for an object in a different address space
- A virtual proxy creates expensive objects on demand.
- A protection proxy controls access to the original object
- A smart reference is a replacement for a bare pointer that performs additional actions when an object is accessed

6a. What do you mean by-part-whole hierarchies? Explain with suitable- example

(05 Mark)

Composite pattern lets client treat individual objects and compositions of objects uniformly in order to compose objects into tree structures to represent part-whole hierarchies.

- ✓ Graphics applications like drawing editors and schematic capture systems let users build complex diagrams out of simple components.
- ✓ The user can group components to form larger components, which in turn can be grouped to form still larger components.
- ✓ A simple implementation could define classes for graphical primitives such as Text and Lines plus other classes that act as containers for these primitives.



- ✓ Graphic declares operations like Draw that are specific to graphical objects. It also declares operations that all composite objects share, such as operations for accessing and managing its children.
- ✓ The subclasses Line, Rectangle, and Text define primitive graphical objects. These classes implement draw to draw lines, rectangles, and text, respectively.

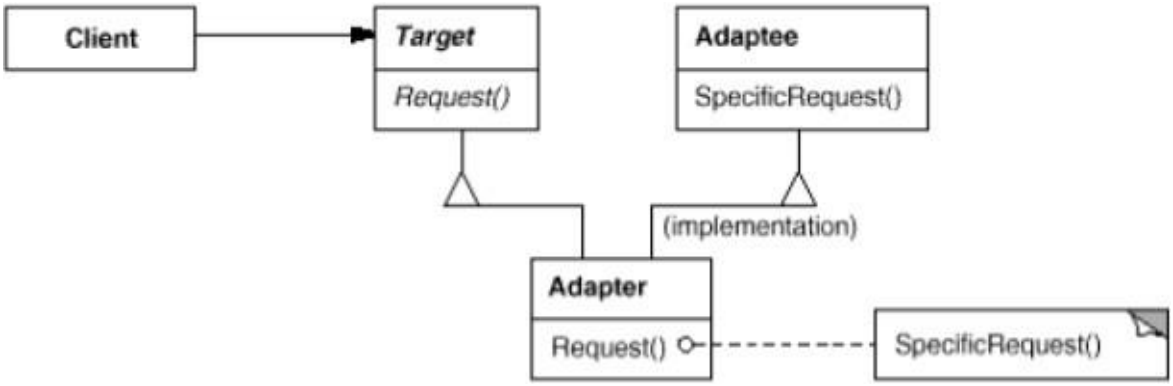
Applicability: Use composite pattern when:

1. You-want to represent part-whole hierarchies of objects.
2. You want clients to be able to ignore the difference between compositions of objects and individual objects.

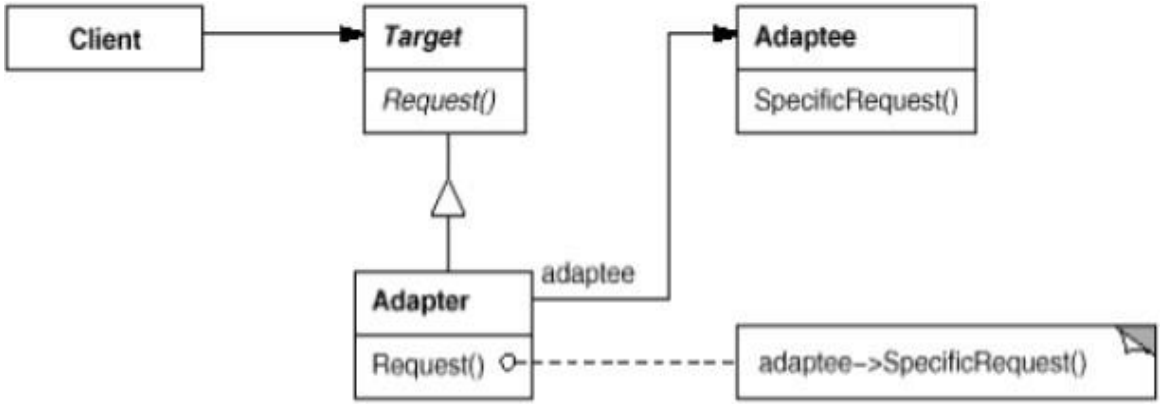
6 b Explain an object adapter and a class adapter. (0 5Marks)

Structure:

A class adapter uses multiple inheritances to adapt one interface to another. The structure of class adapter is shown below:



An object adapter relies on object composition. The structure of an object adapter is as shown below:



Participants:

- Target (shape): Defines the domain specific interface the client uses.
- Client (Drawing Editor): Collaborates with the objects conforming to the Target interface.
- Adaptee (TextView): Defines an existing interface that needs to be adapted.
- Adapter (TextShape): Adapts the interface of the Adaptee to the Traget interface.

Collaborations:

Clients call operations on an Adapter instance. In turn, the adapter calls Adaptec operations that carry out the request.

Consequences: Class and object adapters have different trade-offs.

A class adapter:

- ✓ Adapts Adaptee to Target by committing to a concrete Adapter class. As a consequence, a class adapter won't work when we want to adapt a class and its subclasses.
- ✓ Let Adapter to override some of the behavior of the Adaptee since it is a subclass of Adaptee.
- ✓ Introduces only one object, and no additional pointer indirection is needed to get to the Adaptee.

An object adapter:

- ✓ Let's a single Adapter work with many Adaptees i.e the Adaptee itself and all of its subclasses. The Adapter can also add functionality to all Adaptees at once.
- ✓ Makes it harder to override Adaptee behavior

6c. What are the issues to consider when implementing the composite pattern? (06Marks)

Following are the issues to consider when implementing composite pattern:

1. Explicit parent references: Maintaining references from child components to their parent can simplify the traversal and management of a composite structure.
2. Sharing components: It's often useful to share components
3. Maximizing the Component interface: One of the goals of the Composite pattern is to make clients unaware of the specific Leaf or Composite classes they're using.
4. Declaring the child management operations: Although the Composite class implements the Add and Remove operations for managing children.
5. Should Component implement a list of Components: You might be tempted to define the set of children as an instance variable in the Component class where the child access and management operations are declared
6. Child ordering: Many designs specify an ordering on the children of Composite
7. Caching to improve performance: If you need to traverse or search compositions frequently, the Composite class can cache traversal or search information about its children.
8. Who should delete components: In languages without garbage collection, it's usually best to make a Composite responsible for deleting its children when it's destroyed.
9. 9. What's the best data structure for storing components: Composites may use a variety of data structures to store their children, including linked lists, trees, arrays, and hash tables.

MODULE 4

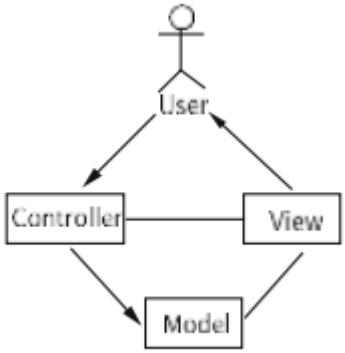
7a. Explain Model-View-Control1er pattern in detail? (6Marks)

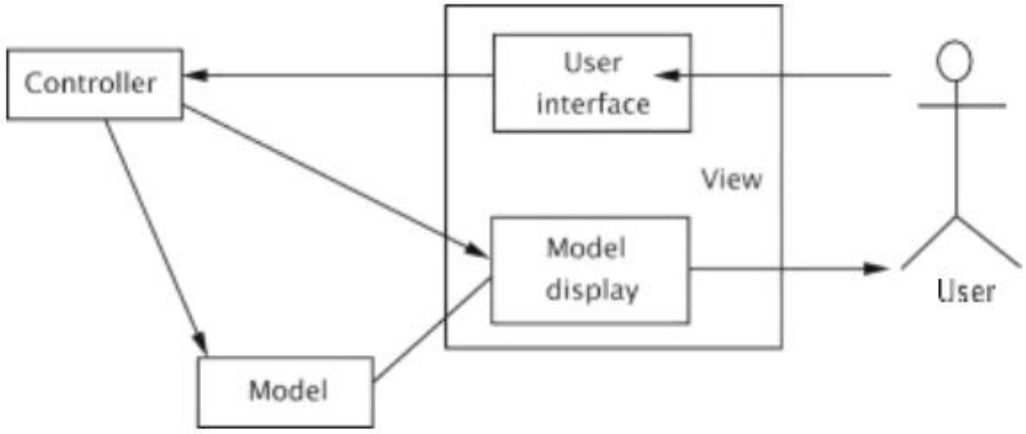
The pattern divides the application into three subsystems: model, view, and controller.

1: Model: The model, which is a relatively passive object, stores the data. object can play the role of model.

2: View: The view renders the model into a specified format, typically something that is suitable for interaction with the end user. For instance, if the model stores information about bank accounts, a certain view may display only the number of accounts and the total of the account balances.

3: Controller: The controller captures user input and when necessary, issues method calls on the model to modify the stored data. When the model changes, the view responds by appropriately modifying the display.





- The pattern separates the application object or the data, which is termed the Model, from the manner in which it is rendered to the end-user (View) and from the way in which the end-user manipulates it (Controller)
- The MVC pattern helps produce highly cohesive modules with a low degree of coupling
- This facilitates greater flexibility and reuse
- MVC also provides a powerful way to organize systems that support multiple presentations of the same information
- The model, which is a relatively passive object, stores the data
- Any object can play the role of model
- If the model stores information about bank accounts, a certain view may display only the number of accounts and the total of the account balances
- The controller captures user input and when necessary, issues method calls on the model to modify the stored data
- The model changes only when user input causes the controller to inform the model of the changes
- As with any software architecture, the designer needs to have a clear idea about how the

responsibilities are to be shared between the subsystems.

- This task can be simplified if the role of each subsystem is clearly defined.
  - The view is responsible for all the presentation issues.
  - The model holds the application object.
  - The controller takes care of the response strategy.

7 b Draw and explain sequence diagram for adding a line (5 Marks)

11.7 Implementing the Undo Operation 365

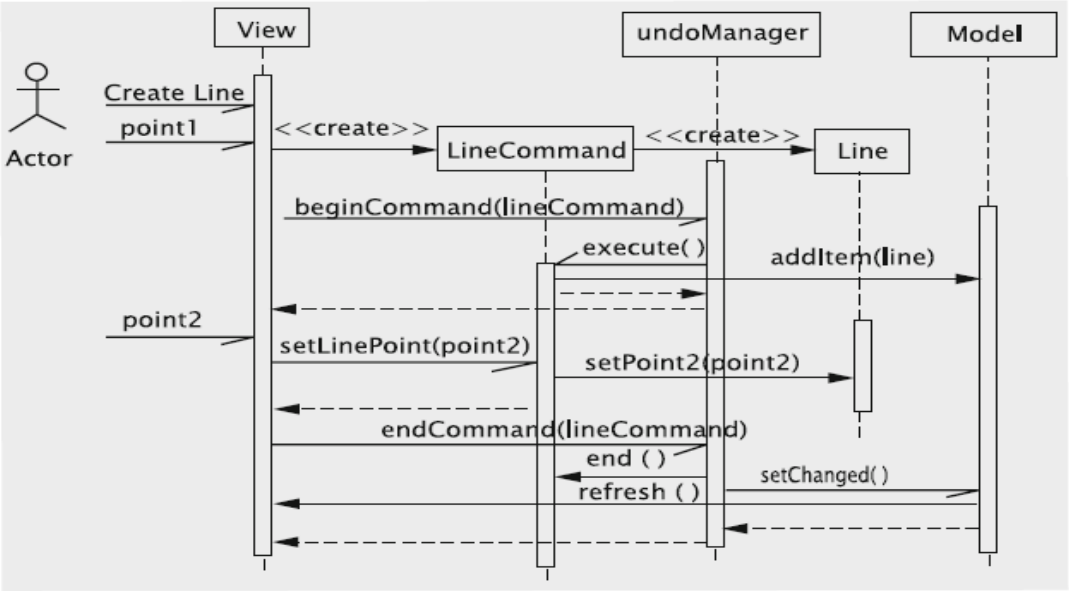


Fig. 11.17 Sequence diagram for adding a line

1 Incomplete items might be rendered differently from complete items. For instance, for a line, after the first click, the UI could track the mouse movement and draw a line between the first click point and the current mouse location; this line keeps shifting as the user moves the mouse. Likewise, if we were to extend the program to include triangles, which need three clicks, one side may be displayed after two clicks. Labels in construction must show the insertion point for the next character.

2 Some fields in an incomplete item might not have 'proper' values. Consequently, rendering an incomplete item could be trickier. An incomplete line, for instance, might have one of the endpoints null. In such cases, it is inefficient to use the same render method for both incomplete items and complete items because that method will need to check whether the fields are valid and take appropriate actions to handle these special cases. Since we ensure that there is at most one incomplete item, this is not a sound approach.

7c. Mention the characteristics of architectural patterns. (5 Marks)

Architectural patterns have the following characteristics:

- They have evolved over time. In the early years of software development, it was not very clear to the designers how systems should be laid out. Over time, some kind of categorization emerged, of the kinds software systems that are needed. In due course, it became clearer as to how these systems and the demands on them change over their lifetime. These enabled practitioners to figure out what kind of layout could alleviate some of the commonly encountered problems.

- A given pattern is usually applicable for a certain class of software system The MVC pattern for instance, is well-suited for interactive systems, but might be a poor fit for designing a payroll program that prints pay checks.
- The need for these is not obvious to the untrained eye when a designer first encounters a new class of software; it is not very obvious what the architecture should be. One reason for this is that the designer is not aware of how the requirements might change over time, or what kind of modifications is likely to be needed. It is therefore prudent to follow the dictates of the wisdom of past practitioners. This is somewhat different from design patterns, which we are able to 'derive' by applying some of the well-established 'axioms' of object-oriented analysis and design. (In case of our MVC example, we did justify the choice of the architecture, but this was done by demonstrating that it would be easier to add new operations to the system. Such an understanding is usually something that is acquired over the lifetime of a system.)

8a. What are the benefits of design of the subsystems? (4 Marks)

1. Cohesive modules: Instead of putting unrelated code (display and data) in the same module, we separate the functionality so that each module is cohesive.
2. Flexibility: The model is unaware of the exact nature of the view or controller it is working with. It is simply an observable. This adds flexibility.
3. Low coupling: Modularity of the design improves the chances that components can be swapped in and out as the user or programmer desires. This also promotes parallel development, easier debugging, and maintenance.
4. Adaptable modules: Components can be changed with less interference to the rest of the system.
5. Distributed systems: Since the modules are separated, it is possible that the three subsystems are geographically separated.

8 b Explain the issues need to be highlighted when implementing the UNDO operation? (6Marks)

In the context of implementing the undo operation, a few issues need to be highlighted.

- **Single-level undo versus multiple-level undo** A simple form of undo is when only one operation (i.e., the most recent one) can be undone. This is relatively easy, since we can afford to simply clone the model before each operation and restore the clone to undo.
- **Undo and redo are unlike the other operations** If an undo operation is treated the same as any other operation, then two successive undo operations cancel each other out, since the second undo reverses the effect of the first undo and is thus a redo. The undo (and redo) operations must therefore have a special status as meta-operations if several operations must be undone.
- **Not all things are undoable** This can happen for two reasons. Some operations like 'print file' are irreversible, and hence undoable. Other operations like 'save to disk' may not be worth the trouble to undo, due to the overheads involved.
- **Blocking further undo/redo operations** It is easy to see that uncontrolled undo and redo can

result in meaningless requests. In general, it is safer to block redo whenever a new command is executed. Consider a situation where we have the sequence: Select(a), undo, Select(a), redo. The redo tries to mark a as selected, and this could result in an exception depending on how things are implemented. A more severe problem arises with Create

- **Rectangle(r), Colour Rectangle(r, blue), undo, Delete(r), redo**. Here, the redo will attempt to colour a rectangle that does not exist anymore.
- **Solution should be efficient** This constraint rules out naive solutions like saving the model to disk after each operation.

8c. Describe implementation of view class with example. (6 Marks)

The view maintains two panels: one for the buttons and the other for drawing the items.

```
public class View extends JFrame implements Observer { private JPanel
  drawingPanel;

  private JPanel buttonPanel;

  // JButton references for buttons such as draw line, delete, etc. private class
  DrawingPanel extends JPanel {

    // code to redraw the drawing and manage the listeners

  }

  public View() {

    // code to create the buttons and panels and put them in the JFrame

  }

  public void update(Observable model, Object dummy) {
    drawingPanel.repaint();

  }

}
```

The code to set up the panels and buttons is quite straightforward, so we do not dwell upon that.

The DrawingPanel class overrides the paintComponentmethod, which is called by the system whenever the screen is to be updated. The method displays all unselected items by first obtaining an enumeration of unselected items from the model and calling the render method on each. Then it changes the colour to red and draws the selected items.

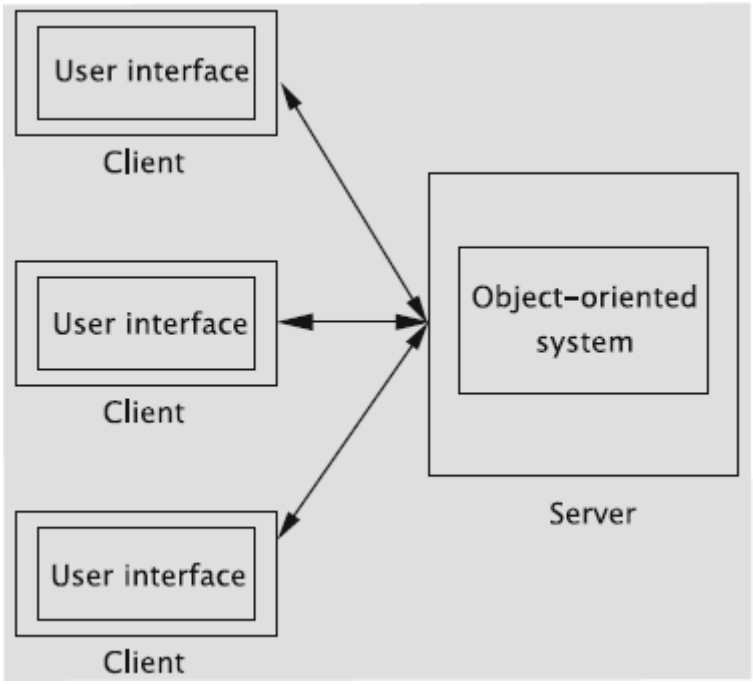
```
public void paintComponent(Graphics g) {
  model.setUI(NewSwingUI.getInstance()); super.paintComponent(g);
  (NewSwingUI.getInstance()).setGraphics(g); g.setColor(Color.BLUE);

  Enumeration enumeration = model.getItems(); while
  (enumeration.hasMoreElements()) {

    ((Item) enumeration.nextElement()).render();
```

```
    }

    g.setColor(Color.RED);

    enumeration = model.getSelectedItems(); while
    (enumeration.hasMoreElements()) {

      ((Item) enumeration.nextElement()).render();

    }

}
```

MODULE 5

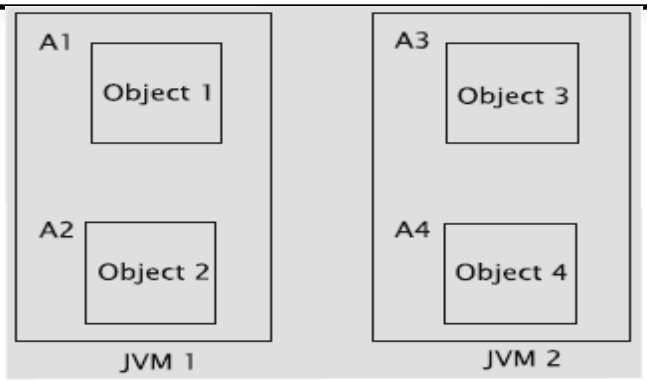
9a. Explain the performance of client/server systems? (6Marks)

- Each client runs a program that provides a user interface, which may or not be a GUI.
- The server hosts an object-oriented system.
- Like any other client/server system, clients send requests to the server, these requests are processed by the object-oriented system at the server, and the results are returned.
- The results are then shown to end-users via the user interface at the clients.



There is a basic difficulty in accessing objects running in a different Java Virtual Machine (JVM). Let us consider two JVMs hosting objects as in Fig. below.

- A single JVM has an address space part of which is allocated to objects living in it. For example,



- Objects object 1 and object 2 are created in JVM 1 and are allocated at addresses A1 and A2 respectively. Similarly, objects object 3 and object 4 live in JVM 2 and are respectively allocated addresses A3 and A4.
- Code within Object 2 can access fields and methods in object 1 using address A1. However, addresses A3 and A4 that give the addresses of objects object 3 and object 4 in JVM 2 are meaningless within JVM 1.

9 b How the Library Syslem can be deployed on the World-Wide-Web? (5 Marks)

Developing User Requirements

First task is to determine the system requirements: **Example**: **Library system**

1. The user must be able to type in a URL in the browser and connect to the library system.
2. Users are classified into two categories:
   a. super users :
      - Superusers are essentially designated library employees, and ordinary members are the general public who borrow library books. superusers can execute any command when logged in from a terminal in the library.
   b. Ordinary members.
      - Ordinary members are the general public who borrow library books.
      - Ordinary members cannot access some ‘privileged commands’.

   In particular, the division is as follows:
      a. Only superusers can issue the following commands: add a member, add a book, return a book, remove a book, process holds, save data to disk, and retrieve data from disk.
      b. Ordinary members and super users may invoke the following commands: issue and renew books, place and remove holds, and print transactions.
      c. Every user eventually issues the exit command to terminate his/her session.
3. Some commands can be issued from the library only. These include all of the commands that only the superuser has access to and the command to issue books.
4. A superuser cannot issue any commands from outside of the library. They can log in, but the only command choice will be to exit the system.
5. Superusers have special user ids and corresponding password. For regular members, their library member id will be their user id and their phone number will be the

password.

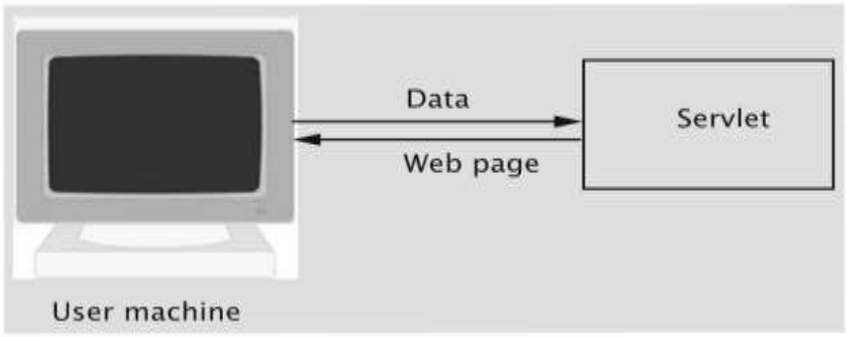


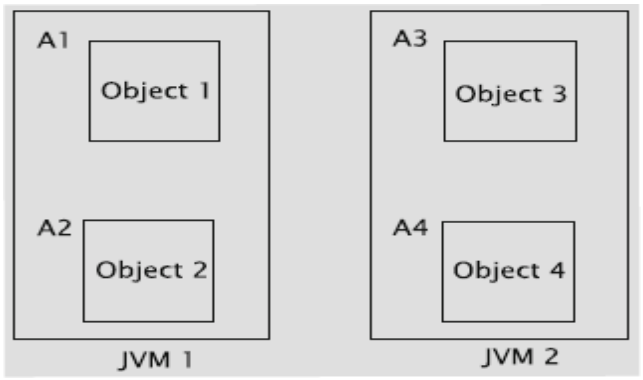
Figure 5.1: How servlets and HTML cooperate to serve web pages

9 c Describe the difficulties in accessing objects in a different JVM. (6 Marks)

This difficulty can be handled in one of two ways:

1. By using object-oriented support software: The software solves the problem by the use of proxies that receive method calls on 'remote' objects, ship these calls, and then collect and return the results to the object that invoked the call. The client could have a custom-built piece of software that interacts with the server software. This approach is the basis of Java Remote Method Invocation.

2. By avoiding direct use of remote objects by using the Hyper Text Transfer Protocol (HTTP): The system sends requests and collects responses via encoded text messages. The object(s) to be used to accomplish the task, the parameters, etc., are all transmitted via these messages. This approach has the client employ an Internet browser, which is, of course, a piece of general-purpose software for accessing documents on the world-wide web.



10a. Explain how to implement object-oriented system on the web? (5 Marks)

HTML and Java Servlets

- System displays web pages via a browser has to create HTML code.
- HTML code displays text, graphics such as images, links that users can click to move to other web pages, and forms for the user to enter data.
- An HTML program can be thought of as containing a header, a body, and a trailer.

The header contains code like the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta content="text/html; charset=ISO-8859-1"
   http-equiv="content-type">
  <title>A Web Page</title>
</head>
```

- The first four lines are usually written as given for any HTML file.
- observe words such as html and head that are enclosed between angled brackets (< and >). They are called tags.
- HTML tags usually occur in pairs: start tag that begins an entry and end tag that signals the entry's end. For example, the tag <head> begins the header and is ended by </head>.
- The text between the start and end tags is the element content.
- In the fifth line we see the tag title, which defines the string that is displayed in the title bar.

As a sample body, let us consider the following.

```
<body>
<h1>
    <span style="color: rgb(0, 0, 255);">
    <span style="font-family: lucida bright;">
    <span style="font-style: italic;">
    <span style="font-weight: bold;">
       An Application
    </span>
    </span>
    </span>
    </span>
</h1>
</body>
```

- The body contains code that determines what gets displayed in the browser's window.
- Some tags may have attributes, which provide additional information.

10b. List and explain for hosting distributed applications? (5 Marks)

Businesses usually install multiple computer systems that are interconnected by communication links, and applications run across a network of computers rather than on a single machine. Such systems are called **distributed systems.**

Advantages

a. It is more economical and efficient to process data at the point of origin.
b. Distributed systems make it easier for users to access and share resources.
c. They also offer higher reliability and availability: failure of a single computer does not cripple the system as a whole.
d. It is also more cost effective to add more computing power.

Drawbacks

a. The software for implementing them is complex.
   Distributed systems must coordinate actions between a number of possibly heterogeneous computer systems; if data is replicated, the copies must be made mutually consistent.
b. Data access may be slow because information may have to be transferred across communication links.
c. Securing the data is a challenge.
   As data is distributed over multiple systems and transported over communication links, care must be taken to guarantee that it is not lost, corrupted, or stolen.

10 c Write short note on (6 Marks)

i. **Marshalling and Demarshalling**
ii. **GET or POST**

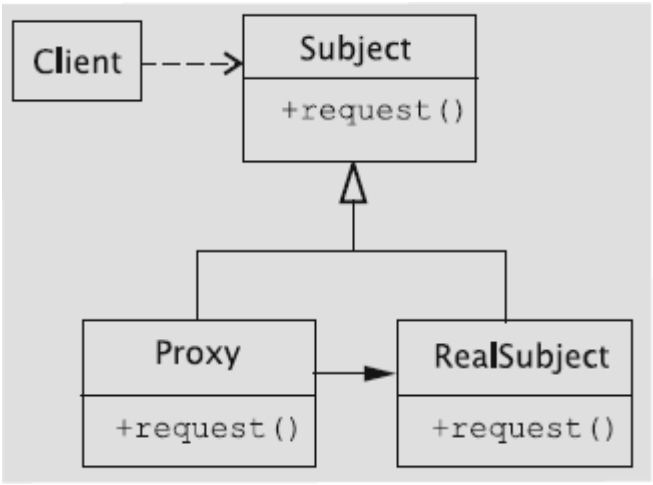
i. Marshalling and Demarshalling

Java RMI employs proxies to stand in for remote objects. All operations exported to remote sites (remote operations) are implemented by the proxy. Proxies are termed *stubs* in Java RMI. These stubs are created by the RMI compiler

- When the client calls a remote method, the corresponding method of the proxy object is invoked. The proxy object then assembles a message that contains the remote object's identity, method name, and parameters. This assembly is called **marshalling**.
- In this process, the method call must be represented with enough information so that the remote site knows the object to be used, the method to be invoked, and the parameters to be supplied.
- When the message is received by it, the server performs **demarshalling**, whereby the process is reversed.

ii. GET or POST

GET method is used for requesting the URL from a web server to fetch the HTML



documents. It is a conventional method for browsers to deliver the information which counted as a part of the HTTP protocol. The GET method represented in the form of URL, so that it can be bookmarked. GET is extensively used in search engines. After the submission of a query by the user to the search engine, the engine executes the query and gives the resulting page. The query results can be set as a link (bookmarked).

GET method enables the generation of anchors, which helps in accessing the CGI program with the query de voiding the usage of form. The query is constructed into a link, so when the link is visited the CGI program will retrieve the suitable information from the database.

POST method is suitable in the condition where a significant amount of information can pass through. When a server receives the request by a form employing POST, it continues to "listens" for the left information. In simple words, the method transfers all the relevant information of the form input instantly after the request to the URL is made.

The POST method needs to establish two contacts with the web server whereas GET just makes one. The requests in the POST are managed in the same way as it is managed in the GET method where the spaces are represented in the plus (+) sign and rest characters are encoded in the URL pattern. It can also send the items of a file.