Experiment-1

Aim : Presentation of the Project: Weather Mapping System:

Summary of the Requirements for the Weather Mapping System (WMS) System

Overview

A weather mapping system (WMS) is required to generate weather maps on a regular basis using data collected from remote, unattended weather stations and other data sources such as weather observers, balloons and satellite. Weather stations transmit their data to the area computer in response to a request from that machine.

The area computer system validates the collected data and integrates the data from different sources. The integrated data is archived and, using data from this archive and a digitized map database, a set of local weather maps is created. Maps may be printed for distribution on a special-purpose map printer or may be displayed in a number of different formats.

Typically, the weather data collection system establishes a modem link with the weather station and requests transmission of the data.

Then, the weather station sends a summary of the weather data that has been collected from the instruments (e.g. anemometer, barometer, ground thermometer) in the collection. The data sent to the weather data collection system are the maximum, minimum and average ground temperatures, the maximum, minimum and average air pressures, the maximum, minimum, and average wind speeds. Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and be modified in the future. Weather stations are also automatically monitored by the system, for start-up, shutdown, and for instrument testing and calibration.

Another function carried out by weather stations is pollution monitoring. More specifically, there is an air quality meter that computes the amount of various pollutants in the atmosphere collected from pollution monitoring instruments (e.g. No meter, Smoke meter and Benzene meter). The pollution readings are transmitted at the same time as the weather data. The pollution data collected are NO, smoke and benzene rates.

Product Features

WMS provides a range of map processing operations (e.g. two-dimensional and three- dimensional mappings), weather data storage for the maps, and network access to the maps for remote viewers. Its key marketing features are the following:

- A user-friendly operator environment

- The throughput of weather data acquisition is 50% higher than previous product

- Maps display can be as fast as the maximum hardware speed
- Is designed for easy upgrade to new platforms
- Has open platform connectivity
- Can be connected to peripherals, including printers and digital imagers.

A single control panel is provided with each WMS unit. Additional independent viewing stations are also available.

The Future of WMS

WMS must be designed to be extensible, maintainable, and portable. Its design must be flexible enough to accommodate certain expected changes. The product requirements may change somewhat during development and certainly will change over the lifetime of the product. The physical characteristics of the weather data acquisition instruments may change as new models are introduced. As the processing power of the system increases, more and more work that was the responsibility of the users will shift to the system.

Other product features will also likely change. The product needs to remain compatible with new or evolving standards for file formats and communication of weather information.
In addition, the technology affecting the software components is likely to change over the lifetime of the system. WMS may have to be improved to handle upgrades to commercial components that are part of the product, and the target software environment is likely to change as upgrades are introduced.

The Company

The software company in charge of the development of the WMS has faced significant turnover of their skilled personnel in the last few years, mainly due to the competitive job market. Facing at the same time tight deadlines set by the customers to deliver useful business functionality; the company has decided to make use of significant third-party products and packages as an integral part of their application. However they will also maintain development and maintenance control of the mission-critical components of their applications.

Experiment-2

Aim: Design of the Use Case View, Risk Analysis

Weather Mapping System (WMS)-Use Case View

In this lab session you'll start the analysis of the Weather Mapping System (WMS). The requirements of WMS are summarized in a separate document.

A. Use Case Modeling

1. Identify the actors involved in this problem, create them and provide appropriate documentation using Rational Rose. The documentation for an actor is a brief description that should identify the role the actor plays while interacting with the system. (20%)
2. Evaluate the needs that must be addressed by the system, and based on this information, identify the use cases for this problem. Create the use cases and provide appropriate documentation using Rational Rose. The documentation for a use case is a brief description that states the purpose of the use case in few sentences, and gives high-level definition of the functionality provided by the use case. (20%)
3. Create the main use case diagram in Rational Rose, by putting together the use cases and actors identified previously. Complete the diagram by defining and creating appropriate relationships among the actors and usecases involved. (10%)

B. Risk Analysis

1. One of the most important risks that the development of the weather mapping system is facing is Change management; failure to handle properly for instance changes in the product requirements may lead to the overall project failure. Identify using a cause-effect (sub) tree the risks hierarchy that may lead to the materialization of this risk. Describe these risks and corresponding reduction measures by providing a risk registry. (30%)
2. Not all use cases are architecturally significant. Only critical use cases that carry the most important risks, or the quality requirements or key functionalities are eligible for the architecture definition. One such use case encompasses functionality for collecting data from remote sources: we call it Collect data.
3. Provide the flow of events for Collect data use case. The flow of events is a description of the events needed to accomplish the required behavior of the use case. The flow of events is written in terms of what the system should do, not how the system does it. Link the flow of events document to the use cases in your Rose model. (20%)

Exp 2 Solution:

Implementation of Use case View:

Use Case Modeling

1.The following actors may be identified from the requirements: Instrument, Weather Instrument, Pollution Instrument, Operator, Digitized map database and Consumer.

Actor documentation:

- Operator: a person who is responsible for the maintenance of the weather mapping system.
- Consumer: a person who uses or needs the weather information produced by the system.
- Weather Instrument: weather sensors related to weather stations used to collect raw weather data.
- Pollution Instrument: air data sensors used to collect air data for pollution monitoring.
- Instrument: generalization of weather and pollution sensors.
- Digitized map database: an existing database containing mapping information.

2. The following needs must be addressed by the system:

- The Consumer actor needs to use the system to view or print weather maps.
- Weather sources collect weather data and send them on request to the area computer.
- The area computer validates, integrates and archives the weather data collected, and then processes weather maps.
- The digitized map database provides the mapping information needed to process local weather maps.
- The Operator actor manages and maintains the system.
- The area computer monitors the weather sources, for start-up, shutdown, and for instrument testing and calibration.

Needs:

Based on these needs, the following use cases may be identified:

- Startup
- Shutdown
- Collect data
- Calibrate

Test:

- Validate
- Integrate
- Archive
- Process map
- Display
- Print

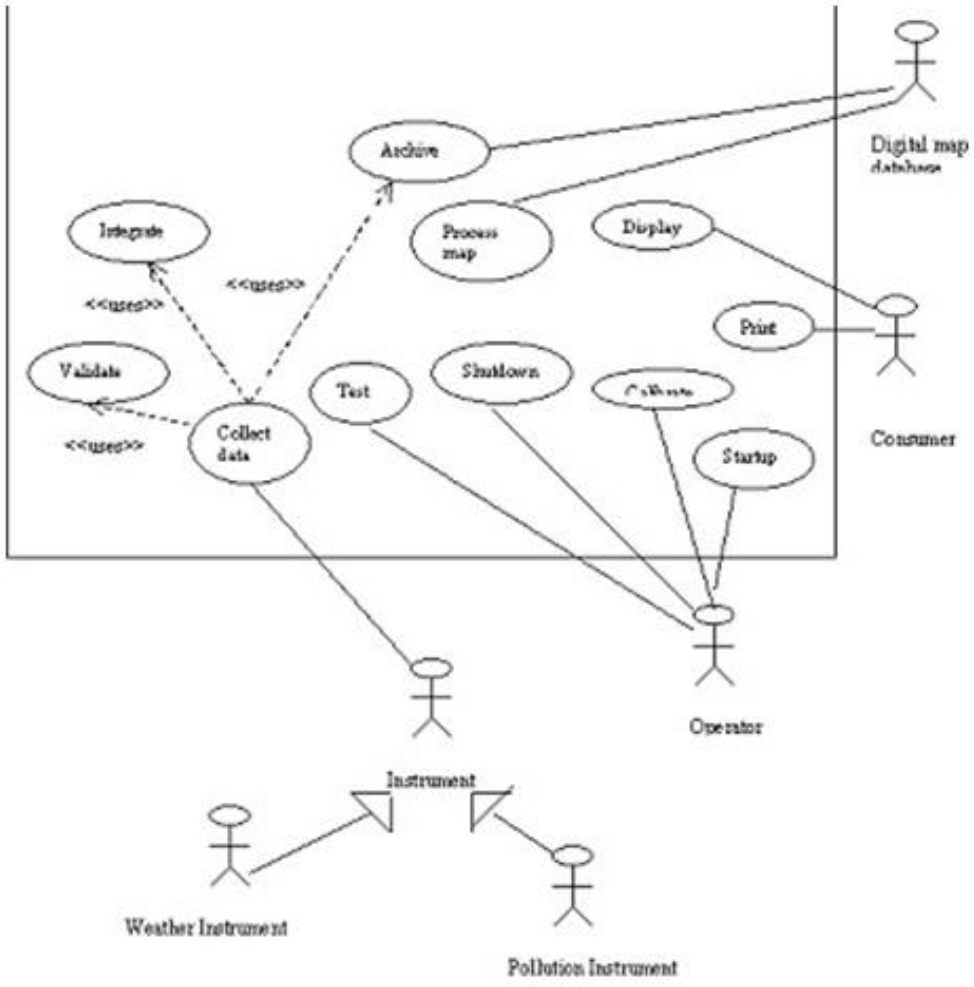
You may identify different use cases (or the same use cases with different names). In all cases the sum of the use cases identified must cover all the key functionalities mentioned in the needs. That should appear through the documentation that you will provide.

Use Case Documentation:

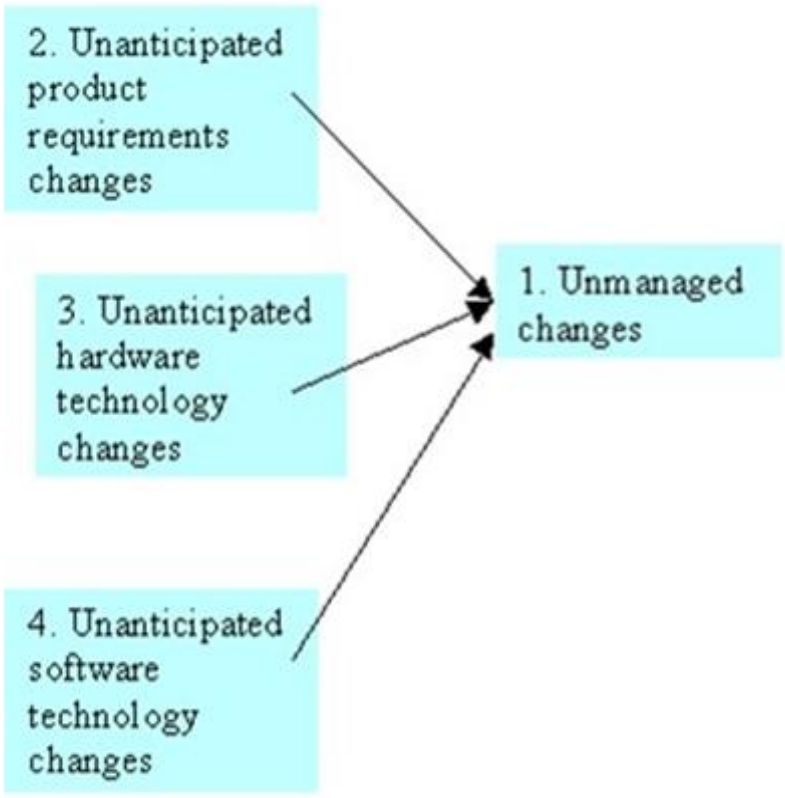
The documentation provided for a use case consists of a brief description of the purpose of the use case in a few sentences. For instance, a brief description of the Collect data use case may be:
The operator or the system starts this use case. It consists of sending a request for data collection to the weather sources, which collect and summarize weather data from the instruments and send them to the area computer.

Experiment-3:

Aim: Main Use Case Diagram for WMS



C. Risk Analysis



Risk	Description	Causes	Source of uncertainty	Nature	Probability	Impact	Reduction Measures
1	Unmanaged changes: failure to cope with the changes occurring in the system development or operation; excessive cost or effort required modifying, updating, evolving, or repairing the system during its operation.	Project fails	Event	Some of the components may remain stable and other may change over time.	L	E	S1:Identify, and Isolates functionalities and components, which are likely to change. S2: Make it easy to add or remove components (modular design, separation of policy and implementation, separation of interface and implementation Etc.) S3: Portable design (use open standards and technologies; use platform independent technologies and standards such as Java and CORBA)
2	Unanticipated product requirements changes: product requirements such as the frequency or	1	Event	Complex design; Design lack of flexibility.	L	E	S1, S2

	procedures of weather data reporting by weather stations may change in the future.						
3	Unanticipated hardware technology changes: physical characteristics of the weather data acquisition instruments may change in the future.	**1**	Event	Complexity, lack of portability.	L	**E**	S1, S3
4	Unanticipated software technology changes: upgrade of COTS components, standard changes (e.g. file formats, communication of weather information etc.)	**1**	Event	Complex design; design lack of flexibility	L	**E**	S1, S2, S3

2. The collect data use case is related to the change management risk in several respects:
Weather data acquisition instruments change, weather information communication procedure and standard change etc.

Flow of Events Description

The flow of events should include:

- When and how the use case starts and ends
- What interaction the use case has with the actors
- What data is needed by the use case
- The normal sequence of events for the use case
- The description of any alternate or exceptional flows

The following description may be given for the Collect data use case:

1.0 Flow of Events for the Collect Data Use Case 1.1Preconditions

The main flow of the Startup use case needs to complete before this use can start.

1.2 Main Flow

This use case begins when the weather data collection system establishes a modem link with the weather station and requests transmission of the data. The weather station sends an acknowledgment to the collection system (E-1), and then collects and summarizes the data from the weather (S-1) and air (S-2) data instruments. The summarized data is sent to the weather data collection system.

1.3 Sub flows

S-1: Report weather data

Weather data readings are collected and reported from weather data instruments. The data sent are the maximum, minimum and average ground and air temperatures, the maximum, minimum and average air pressures, the maximum, minimum, and average wind speeds.

S-2: Report air data

The pollution readings are collected and transmitted at the same time as the weather data. The pollution data collected are NO, smoke and benzene rates

1.4 Alternative Flows

E-1: The collection system hasn't received any acknowledgment after a certain deadline. The main flow of Test use case is started. The use case is terminated and an error message is displayed.

Exp 3: Logical View

Design of the Logical View of the Weather Mapping System (WMS).

A. Use Case Realization

The realization of a use case is called collaboration. A use case is a collection of scenarios, each describing specific aspect of the use case. We consider two scenarios of the Collect data use case called Report Weather Readings and Report Air Quality:

- Report Weather Readings: encompasses functionality for reading weather data from remote sources.
- More specifically weather stations transmit the data collected from weather instruments to the area

computer when they receive a request from that machine.

- Report Air Quality Readings: the pollution readings are automatically collected when requested by a weather station and transmitted at the same time as the weather data.

1. The flow of events for a use case is captured in text, whereas scenarios are captured in interaction diagrams. There are two types of interaction diagrams: sequence diagrams that show object interactions arranged in time sequence, and collaboration diagrams, which show object interactions organized around the objects and their links to each other. Both represent an alternate way of describing a scenario.

a. Give using Rational Rose the sequence diagram corresponding to the Report Weather Readings and the Report Air Quality scenarios. (30%)

b. Give an alternate representation for the same scenarios using collaboration diagrams. (10%) 2.

a. Identify the boundary, entity and control classes involved in these scenarios. Create these classes using Rational Rose and add if applicable appropriate stereotypes. (10%)

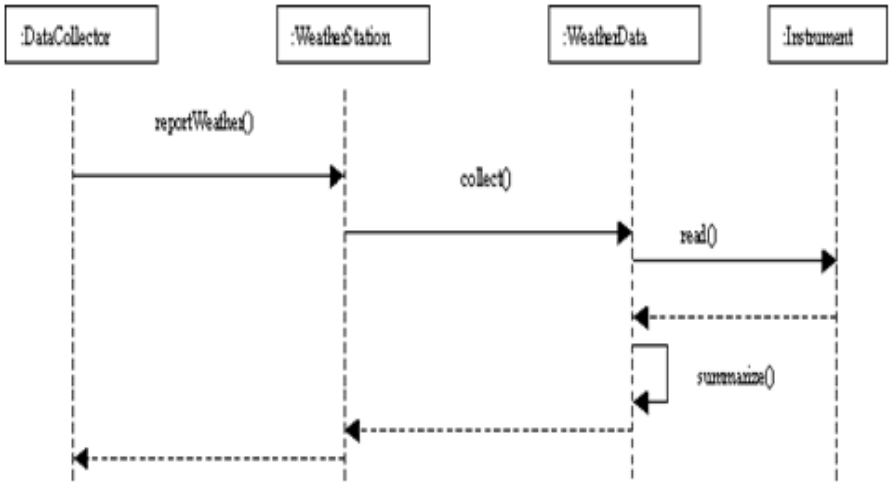
b. Class diagrams may also be attached to use cases and contain a view of the classes participating in the use case. Create using Rose a view of participating classes to the Collect data use case. (10%)

c. Identify the relationships among the classes involved in the scenarios, and create them in the corresponding class diagram using Rose. (10%)

3. Create the attributes and operations involved in the classes participating in the Collect data use case and document them using Rose. The documentation for an operation should state briefly the functionality performed by the operation. It should also state any input values needed by the operation along with the return value of the operation. This information may not be known initially, in which case it should be added later when more is known about the class. The documentation for an attribute should state concisely the purpose of the attribute, not the structure of the attribute. (30%)

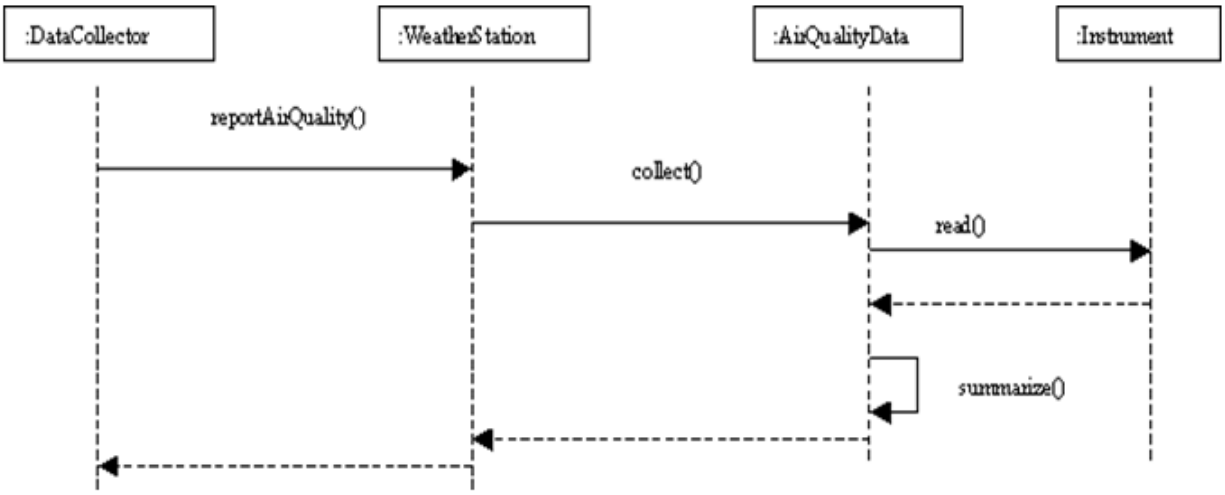
Exp 3: Solution

- Report Weather Readings Scenario



- Report Air Quality readings Scenario

Both diagrams are similar: you may describe both scenarios using one diagram, in which case you should describe the concurrency aspects.



a. Entity class models information and associated behavior that is generally long lived. This type of class is typically independent of their surroundings; that is they are not sensitive to how the

b. Surroundings communicate with the system. Boundary classes handle the communication between the system surroundings and the inside of the system. They can provide the interface to a user or another system. Control classes model sequencing behavior specific to one or more use cases: they represent the dynamics of the use case.

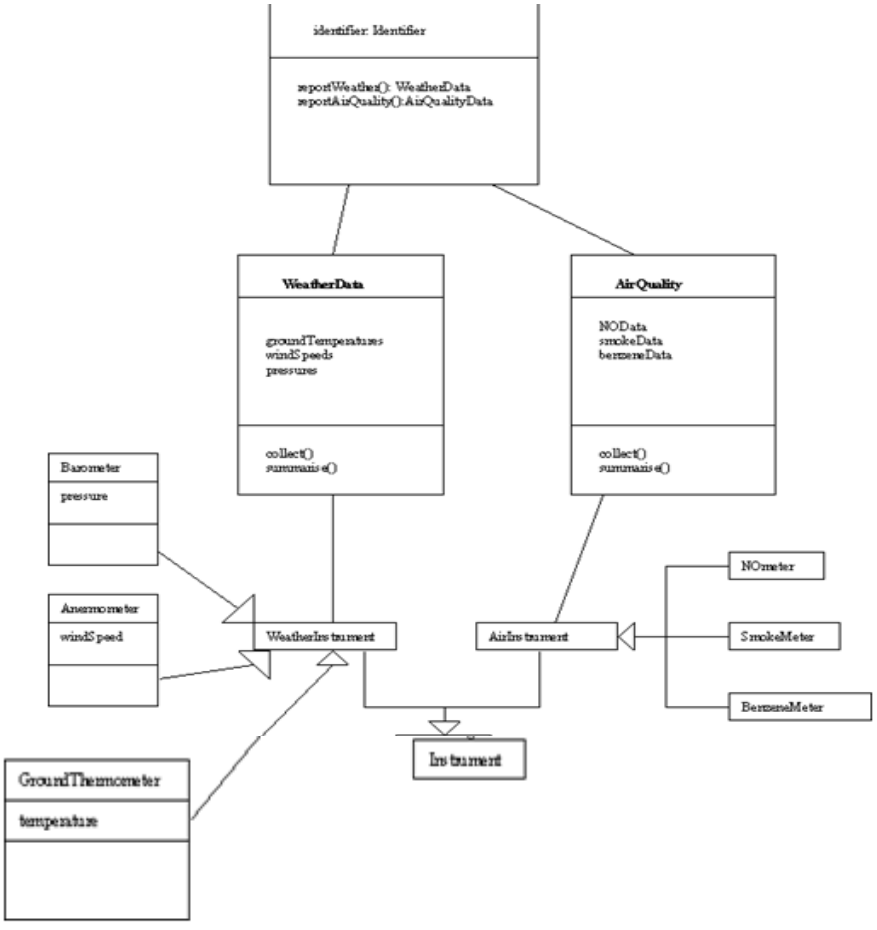
- **Boundary classes**: this use case interacts only with the data collection instruments. Hence, the

boundary classes would be the classes encapsulating the instruments (weather and air data collection instruments: Ground Thermometer, Anemometer, Barometer, Nometer, Smoke meter and Benzenmeter).

- **Entity classes**: this scenario deals with weather and air data collection. We can identify two entity classes: WeatherData and AirQualityData. AirQualityData class represents the air quality data.

- **Control classes**: there are two control classes that handle the flow of events for the use case: Weather Station and Data Collector.

c. We derive the classes participating to the use case from the interaction diagrams. We shall find in the class diagram the same relationships (correspond to the message flows) and the same operations.



Operations:

Operations may be identified using interaction diagrams: operations are associated to the messages exchanged.

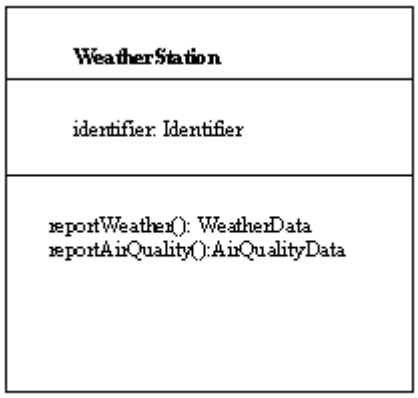
Documentation for the collect () operation of the WeatherData class:

Collect weather information from the weather instruments. Inputs: request from the weather data collection subsystem

Outputs: summarized weather data sent to the weather collection subsystem.

Attributes:

Example: attributes and operations for Weather Station class:



Documentation for the identifier attribute of the WeatherSation class:

The unique reference used to distinguish any weather station in the WMS system.

The documentation should state briefly the purpose of the attribute, not the structure of the attribute. For instance a bad definition for the identifier attribute would be “a character string of length 15”.

Experiment-4

Aim: Integrating Patterns in the Architecture

Integration of selected architectural and design patterns in the logical view obtained previously

Figure 1 depicts a class diagram describing the static structure of the classes participating to the Report weather data scenario of the Collect data use case. The diagram consists of eight classes. Classes Weather Station, WeatherData, Instrument, ground Thermometer, Barometer, and Anemometer describe the data collection functions carried by the weather stations and associated instruments. Classes Data Collector and Archived Data describe data collection functions carried by the area computer.

- Weather Station: provides the basic interface of a weather station with its environment.
- WeatherData: encapsulates the summarized data from the different instruments in a weather station.
- Instrument, ground Thermometer, Barometer, and Anemometer: represent corresponding weather instruments in the system.
- Data Collector: sends data collection requests to Weather stations, and stores the received data.
- Archived Data: represents the data collected by the area computer from a weather station; it encapsulates the collected data, the collection date and time, and the identifier of the source station.
- Identifier, Date, Time, and Readings are user-defined types: Identifier is a user- defined type consisting of a sequence of 4 digits; Readings is a record type whose fields correspond to maximum, minimum, and average weather readings (e.g. temperature, pressure etc); Date (day,month,year) and Time (hour,minute,second) are also defined as record types.

Some of the patterns selected by the architecture team as structuring mechanisms for the software architecture include the pipes-and-filters architectural pattern and the factory design pattern.

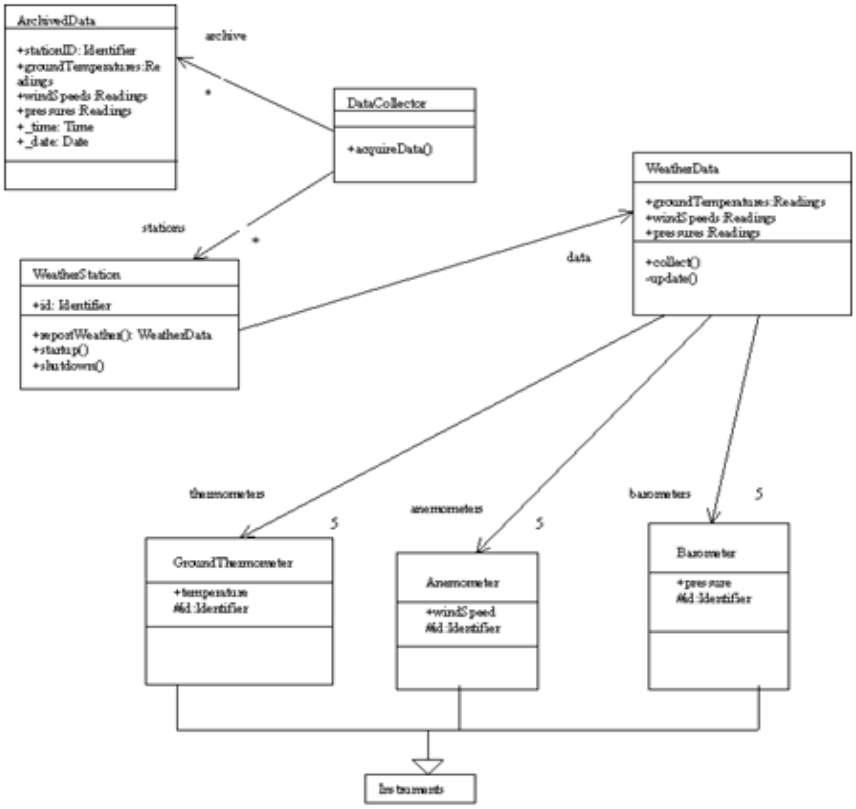
1. Based on the risk factors identified during the risk analysis (conducted in Lab1) explain (briefly) why these patterns are appropriate for handling some of the design issues underlying the development of the weather mapping system. (20%)
2. Re-organize the class diagram given in Figure 1 around the Pipes-and-filter architectural pattern: specify the filters, the data source and sink, and the model of pipes used for interconnection. Precisely, you must provide (in your report) the following information: (50%)
    a. Filters: indicate for each filter whether it is active or passive; specify corresponding

classes (a filter may match one or several classes).

b. Data sink and source: indicate whether they are active or passive; specify corresponding classes.

c. Pipes: specify for each pair of filters (including data source and subsequent filter, or data sink and preceding filter) the model of pipes used for interconnection, either as direct call or synchronization pipeline. For synchronization pipelines, specify whether they use a push, a pull, or a mix model.

3. Use the Factory design pattern to re-design the class structure of the various classes corresponding to weather instruments, and update the class diagram accordingly. (10%)
4. Reusability, extensibility and maintainability are some the main quality factors that characterize the weather mapping system. Organizing the system into loosely coupled and highly cohesive subsystems (e.g. modules) is one of the strategies used to achieve these quality goals. Decomposition in three subsystems named Sensors, Station, and Data Collection is considered. Two possible grouping strategies are proposed in Tables 1 and 2. Select the most suitable one, and motivate your choice by computing a coupling metric such as CBO (Coupling Between Object Classes). Create the subsystems using Rational Rose, and relocate the classes in the Rose Browser. (20%)



Classes	Subsystems
Instrument, GroundThermometerBarometer, Anemometer	Sensor
WeatherStation, WeatherData	Station
Data Collector, Archived Data	DataCollection

Exp 4: Solution

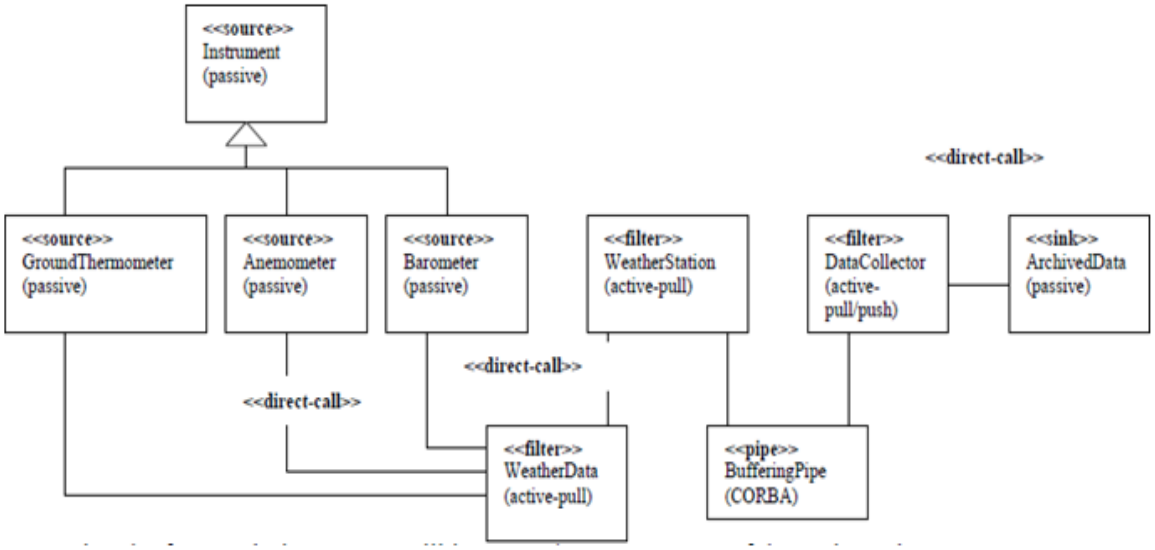
1. Extensibility, maintainability, and portability are among key requirements of the system. It is expected that the product requirements may change during development or operation. The physical characteristics, the kind and the number of the instruments involved may change.

An important design issue in that case consists of making easy removal and addition of components. That can be achieved using the pipes-and-filter pattern.

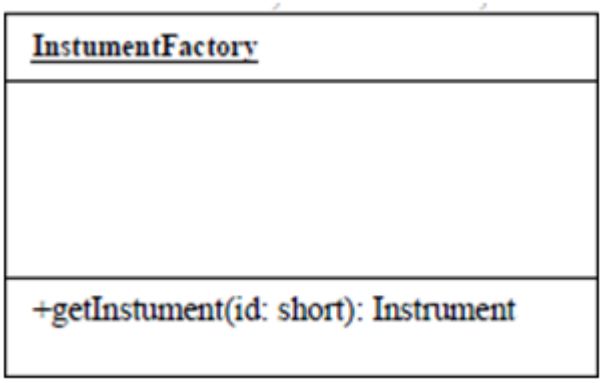
In order to handle the changes in the instruments, the factory design pattern is a mechanism that can introduce the desired level of decoupling.

2.The pipes-and-filter pattern can be applied to the given class structure in various ways. The instruments must be designed as data source, active or passive. WeatherData class may be designed either as a filter or as a pipe. WeatherStation and DataCollector must represent active filters. ArchivedData must be designed as a passive data sink.

According to the requirements, the area computer, through DataCollector class, polls weather stations. So Data Collector is an active filter based on push/pull mechanism. Since WeatherStation is also an active filter, a synchronization buffer may be included here; the buffer may be implemented later using an Interprocess Communication Mechanism such as CORBA. The DataCollector may also pull data directly from weather



3. Using the factory design pattern will improve the management of the various classes corresponding to weather instruments and to anticipate changes in the class structure. The factory design pattern is applied by integrating a factory class named InstrumentFactory in the class diagram that will be used to create instances of instruments. It provides the getInstrument () method, which is in charge of calling the appropriate constructors (GroundThermometer, Barometer, Anemometer).



Experiment-5

Aim: Using UML design Strategy Design pattern

Solution:

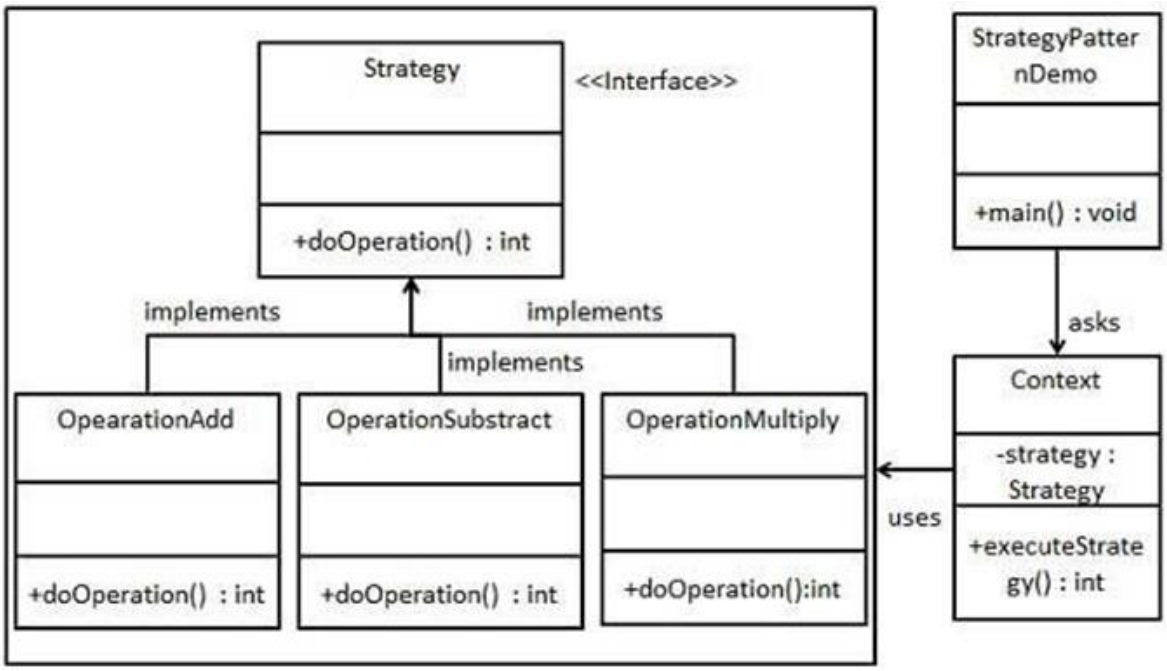
In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.

In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

Implementation:

We are going to create a Strategy interface defining an action and concrete strategy classes implementing the Strategy interface. Context is a class which uses a Strategy.

Strategy Pattern Demo, our demo class, will use Context and strategy objects to demonstrate change in Context behavior based on strategy it deploys or uses.



Experiment-6

Aim: Using UML design Builder Design pattern

Solution:

Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

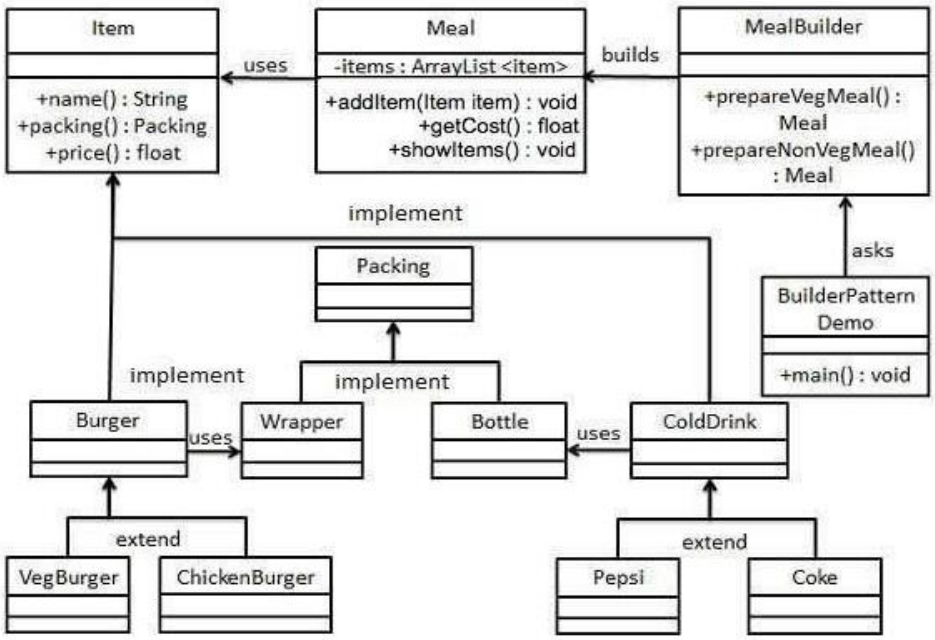
A Builder class builds the final object step by step. This builder is independent of other objects.

Implementation:

We have considered a business case of fast-food restaurant where a typical meal could be a burger and a cold drink. Burger could be either a Veg Burger or Chicken Burger and will be packed by a wrapper. Cold drink could be either a coke or pepsi and will be packed in a bottle.

We are going to create an Item interface representing food items such as burgers and cold drinks and concrete classes implementing the Item interface and a Packing interface representing packaging of food items and concrete classes implementing the Packing interface as burger would be packed in wrapper and cold drink would be packed as bottle.

We then create a Meal class having ArrayList of Item and a MealBuilder to build different types of Meal objects by combining Item. BuilderPatternDemo, our demo class will use MealBuilder to build a Meal.



Experiment-7

Aim: Using UML design Bridge Design pattern

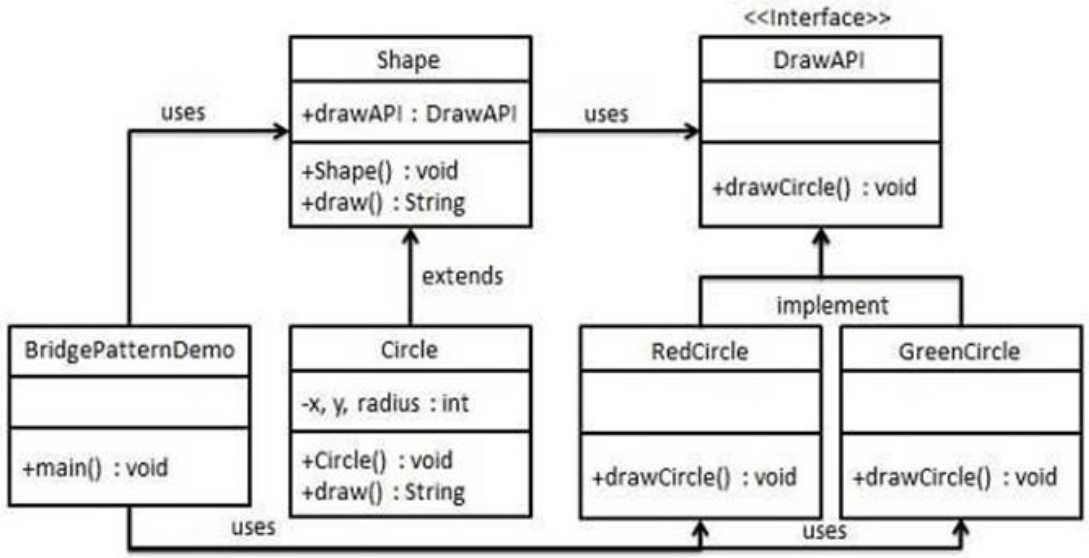
Solution:

Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under structural pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.
This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

We are demonstrating use of Bridge pattern via following example in which a circle can be drawn in different colors using same abstract class method but different bridge implementer classes.
Implementation

We have a DrawAPI interface which is acting as a bridge implementer and concrete classes RedCircle, GreenCircle implementing the DrawAPI interface. Shape is an abstract class and will use object of DrawAPI. BridgePatternDemo, our demo class will use Shape class to draw different colored circle.

Class Diagram:



Experiment-8

Aim: Using UML design Decorator Design pattern

Theory:

Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class. This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

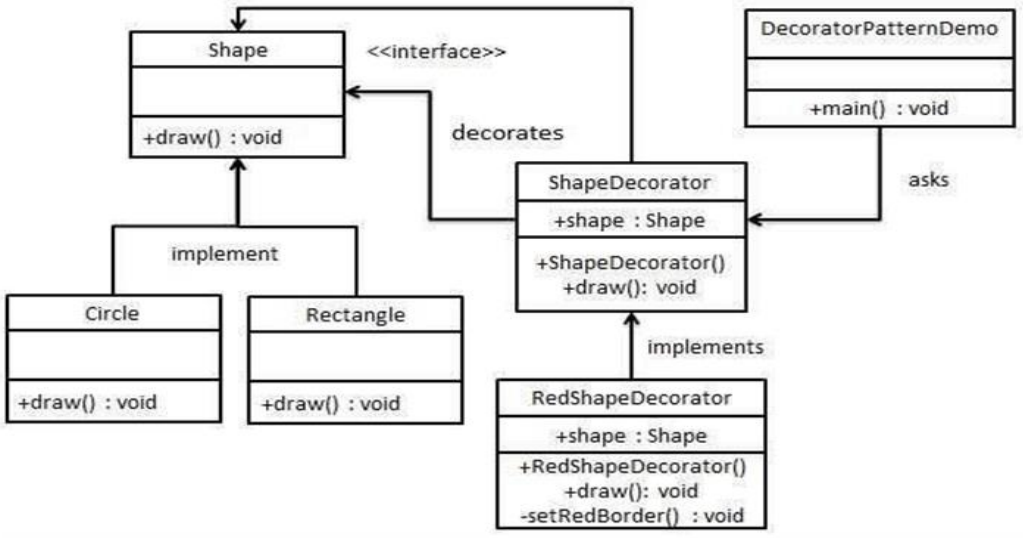
Intent

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub classing for extending functionality.

Also Known As Wrapper **Applicability** Use Decorator

- To add responsibilities to individual objects dynamically and transparently, that is, without affecting other objects.
- for responsibilities that can be withdrawn.
- when extension by sub classing is impractical. Sometimes a large number of independent extensions are possible and would produce an explosion of subclasses to support every combination. Or a class definition may be hidden or otherwise unavailable for sub classing.

Class Diagram:



Experiment-9

Aim: Write a Program to design chain of responsibility pattern.

Theory:

As the name suggests, the chain of responsibility pattern creates a chain of receiver objects for a request. This pattern decouples sender and receiver of a request based on type of request. This pattern comes under behavioral patterns.

In this pattern, normally each receiver contains reference to another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

Example: ATM (rupees of 1000,500,100 etc)

Intent

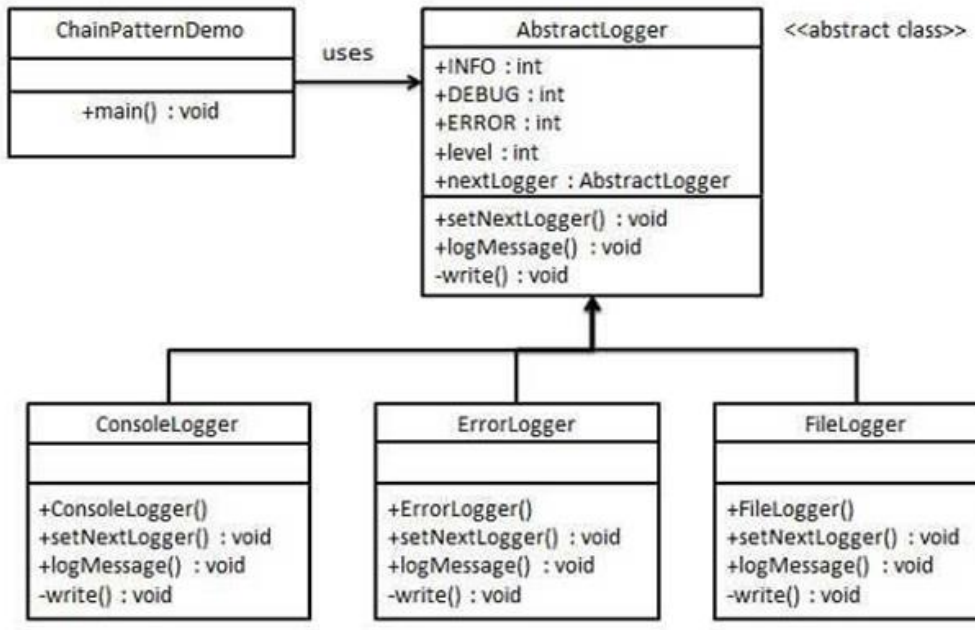
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Applicability

Use Chain of Responsibility when

- More than one object may handle a request, and the handler isn't known a priori. The handler should be ascertained automatically.
- You want to issue a request to one of several objects without specifying the receiver explicitly.
- The set of objects that can handle a request should be specified dynamically.

Class Diagram:



Experiment-10

Aim: Using UML implement Flyweight Design Pattern

Solution:

Flyweight pattern is one of the structural design patterns as this pattern provides ways to decrease object count thus improving application required objects structure. Flyweight pattern is used when we need to create a large number of similar objects (say 10^5). One important feature of flyweight objects is that they are immutable. This means that they cannot be modified once they have been constructed.
Why do we care for number of objects in our program?

- Less number of objects reduces the memory usage, and it manages to keep us away from errors related to memory like java.lang.OutOfMemoryError.
- Although creating an object in Java is really fast, we can still reduce the execution time of our program by sharing objects.

In Flyweight pattern we use a HashMap that stores reference to the object which have already been created, every object is associated with a key. Now when a client wants to create an object, he simply has to pass a key associated with it and if the object has already been created we simply get the reference to that object else it creates a new object and then returns it reference to the client.

To understand Intrinsic and Extrinsic state, let us consider an example.

Suppose in a text editor when we enter a character, an object of Character class is created, the attributes of the Character class are {name, font, size}. We do not need to create an object every time client enters a character since letter ‘B’ is no different from another ‘B’ . If client again types a ‘B’ we simply return the object which we have already created before. Now all these are intrinsic states (name, font, size), since they can be shared among the different objects as they are similar to each other.

Now we add to more attributes to the Character class, they are row and column. They specify the position of a character in the document. Now these attributes will not be similar even for same characters, since no two characters will have the same position in a document, these states are termed as extrinsic states, and they can’t be shared among objects.

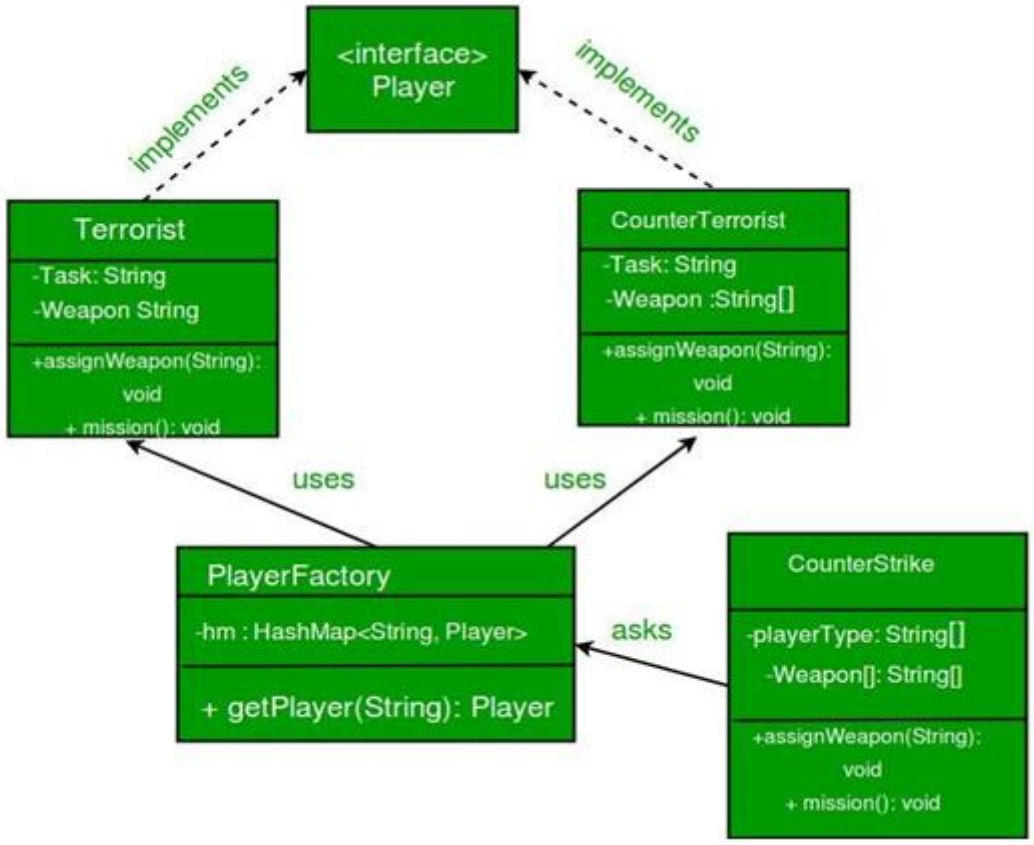
Implementation:

We implement the creation of Terrorists and Counter Terrorists In the game of Counter Strike. So we have 2 classes one for Terrorist (T) and other for Counter Terrorist (CT). Whenever a player asks for a weapon we assign him the asked weapon. In the mission, terrorist’s task is to plant a bomb while the counter terrorists have to diffuse the bomb.

Why to use Flyweight Design Pattern in this example? Here we use the Fly Weight design pattern, since here we need to reduce the object count for players. Now we have n number of players playing CS 1.6, if we do not follow the Fly Weight Design Pattern then we will have to create n number of objects, one for each player. But now we will only have to create 2 objects one for terrorists and other for counter terrorists, we will reuse then again and again whenever required.

Intrinsic State: Here 'task' is an intrinsic state for both types of players, since this is always same for T's/CT's. We can have some other states like their color or any other properties which are similar for all the Terrorists/Counter Terrorists in their respective Terrorists/Counter Terrorists class.
Extrinsic State: Weapon is an extrinsic state since each player can carry any weapon of his/her choice. Weapon need to be passed as a parameter by the client itself.

Class Diagram:



Experiment-11

Aim: Using UML implement FACADE PATTERN

Solution:

Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

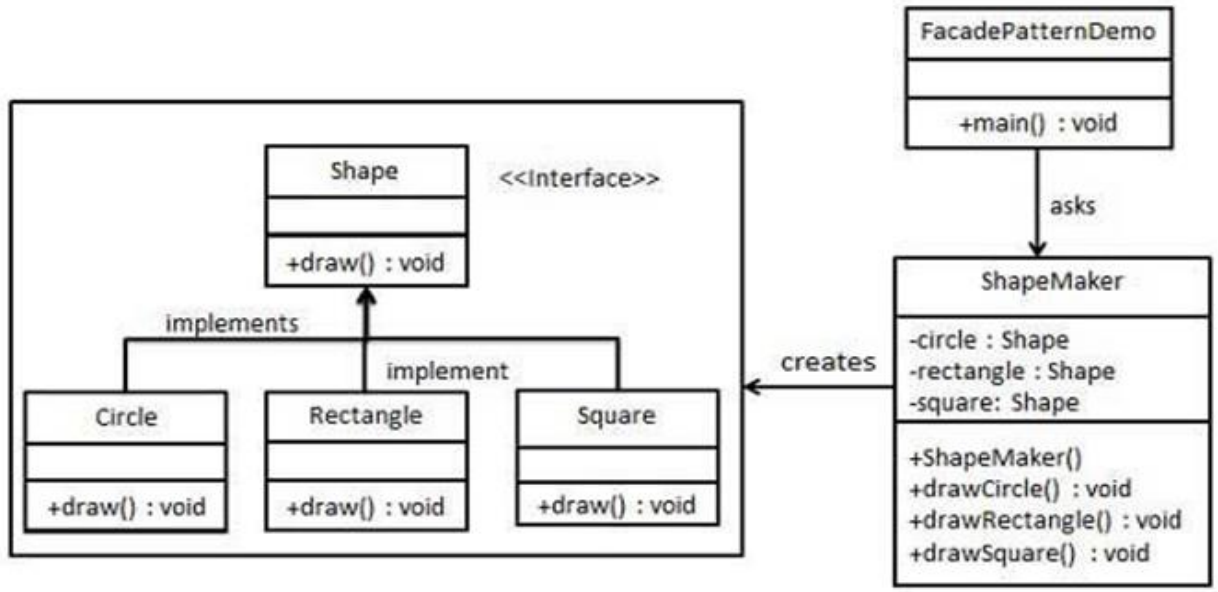
This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.

Implementation:

We are going to create a Shape interface and concrete classes implementing the Shape interface. A facade class ShapeMaker is defined as a next step.

Shape Maker class uses the concrete classes to delegate user calls to these classes. FacadePatternDemo, our demo class, will use ShapeMaker class to show the results.

Class Diagram:



Experiment-12:

Aim: Using UML design Iterator Design pattern

Theory:

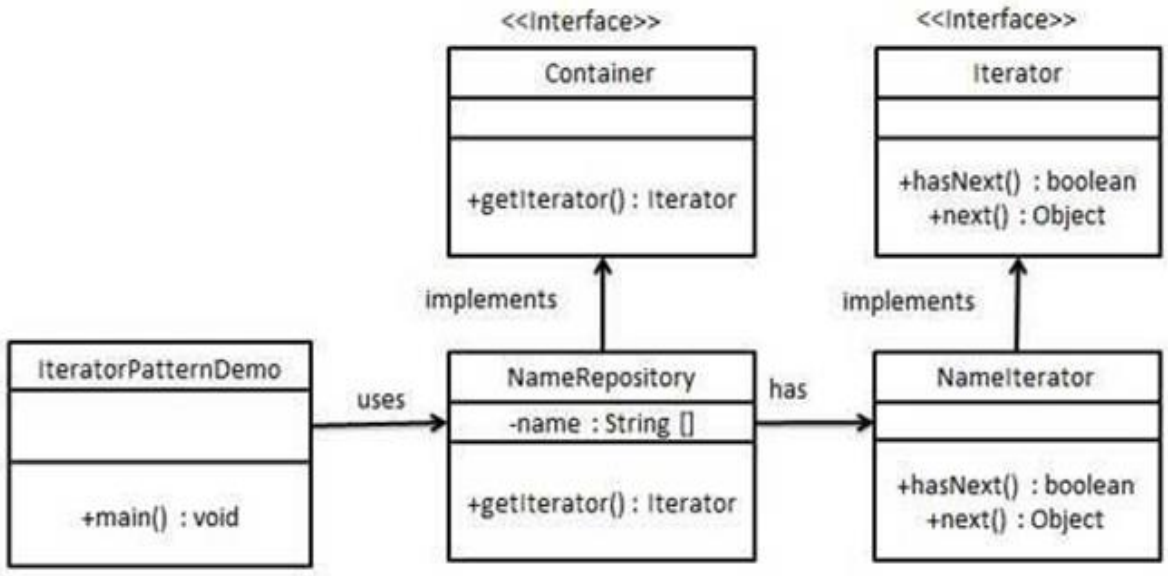
Iterator pattern is very commonly used design pattern in Java and .Net programming environment. This pattern is used to get a way to access the elements of a collection object in sequential manner without any need to know its underlying representation.

Intent:

Use the Iterator pattern:

a. to access an aggregate object's contents without exposing its internal representation.
b. to support multiple traversals of aggregate objects.
c. to provide a uniform interface for traversing different aggregate structures (that is, to support polymorphic iteration).

Class Diagram:



Experiment-13

Aim: Using UML design mediator Design pattern.

Theory:

- Mediator pattern is used to reduce communication complexity between multiple objects or classes. This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintainability of the code by loose coupling. Mediator pattern falls under behavioral pattern category

Intent:

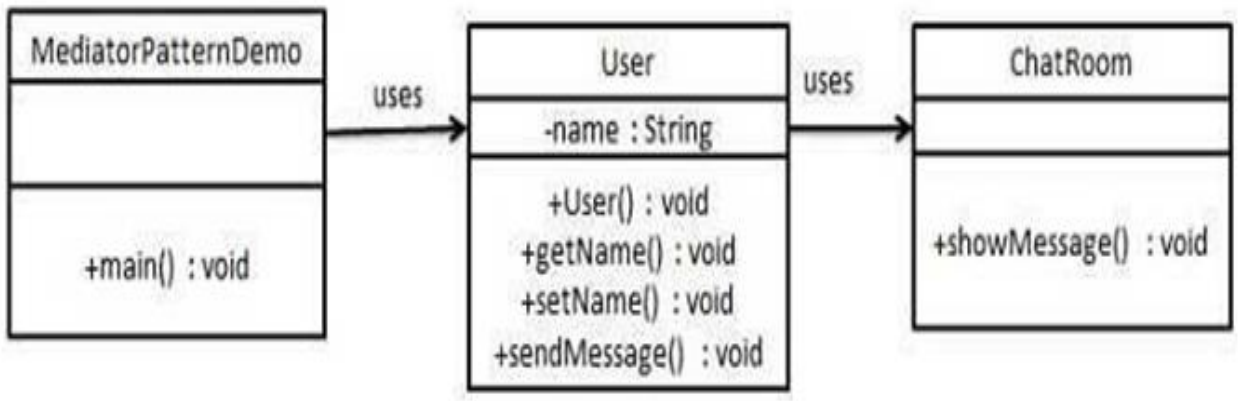
Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

- Applicability

  Use the Mediator pattern when

  - a set of objects communicate in well-defined but complex ways. The resulting interdependencies are unstructured and difficult to understand.
  - reusing an object is difficult because it refers to and communicates with many other objects.

Class Diagram:



Experiment-14

Aim: Using UML design proxy design pattern.

Theory:

- In Proxy pattern, a class represents functionality of another class. This type of designpattern comes under structural pattern.
- In Proxy pattern, we create object having original object to interface its functionality toouter world

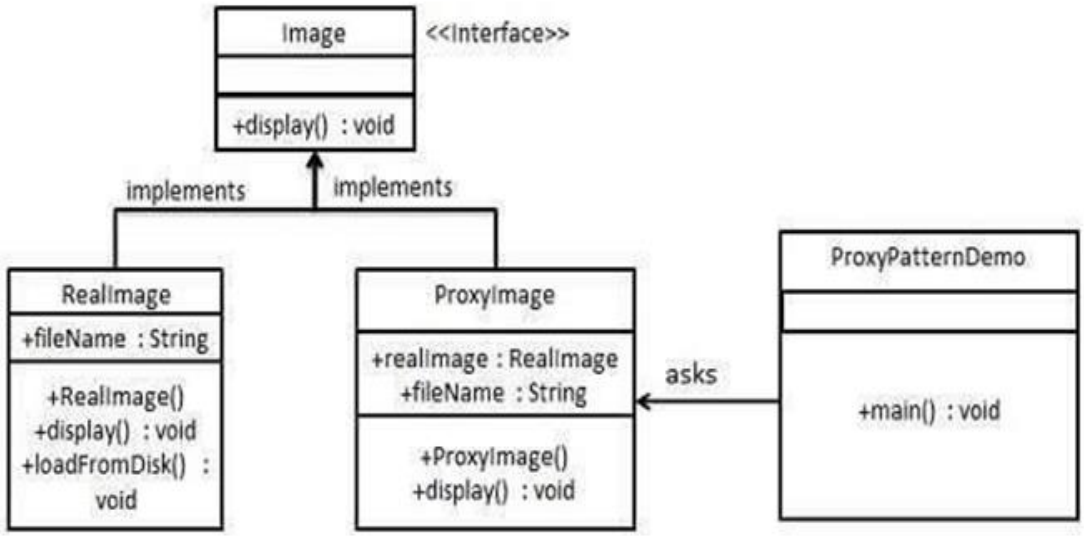
Intent

- Provide a surrogate or placeholder for another object to control access to it.
- Also Known As: Surrogate

Applicability:

- A remote proxy provides a local representative for an object in a different addressspace. NEXTSTEP [Add94] uses the class NXProxy for this purpose.
- virtual proxy creates expensive objects on demand. The ImageProxy described in the Motivation is an example of such a proxy.

Class Diagram:



Experiment-15

Aim: Using UML Design visitor Design pattern. Theory:

In Visitor pattern, we use a visitor class which changes the executing algorithm of an element class. By this way, execution algorithm of element can vary as and when visitor varies. This pattern comes under behavior pattern category. As per the pattern, element object has to accept the visitor object so that visitor object handles the operation on the element object.

Intent

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

Applicability

Use the Visitor pattern when

a. An object structure contains many classes of objects with differing interfaces, and you want to perform operations on these objects that depend on their concrete classes.

b. Many distinct and unrelated operations need to be performed on objects in an object structure, and you want to avoid "polluting" their classes with these operations. Visitor lets you keep related operations together by defining them in one class. When the object structure is shared by many applications, use Visitor to put operations in just those applications that need them.

c. The classes defining the object structure rarely change, but you often want to define new operations over the structure. Changing the object structure classes requires redefining the interface to all visitors, which is potentially costly. If the object structure classes change often, then it's probably better to define the operations in those classes.

Class Diagram: