OOAD – Pass Paper 2012 & Answer

Q1

01.What is meant by "Object Orientation"?

Object Orientation is meant by concepts as ‘object’ that have data fields (attributes that describe the object) and associated procedures known as methods.

02. Brieflydescribe the following fundamental concepts of object oriented world

1) Object:-An Object is a Real Word Thing Which performs a Specific Task and which has a set of Properties and Methods. Properties of Object are Also Called as Attributes of an Object and Method is also Known as the Function of the Object

2) Class: - A Class is that which contains the set of properties and Methods of an object in a single unit Like Animals is name of class which contains all the Properties and Methods of an Object of another Animals So if we wants to access any data from the Class then first we have to create the Object of a class Then we can use any data and method from class

3)Constructors

Constructor is used for Initializing the values to the data members of the Class.
2) Constructor is that whose name is same as name of class.
3) Constructor gets Automatically called when an object of class is created.
4) Constructors never have a Return Type even void.

4) Destructors- As we know that Constructor is that which is used for Assigning Some Values to data Members and For Assigning Some Values this May also used Some Memory so that to free up the Memory which is Allocated by Constructor, destructor is used which gets Automatically Called at the End of Program and we doesn’t have to Explicitly Call a Destructor and Destructor Cant be Parameterized or a Copy This can be only one Means Default Destructor which Have no Arguments.

3. Define the suitable class using c++ programming language and explain above mention four concepts there

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<process.h>

// statement of customer detail....

struct acc_data
{
char c_name[25];
int acc_no;
float balance;
char acc_type[2];

};

//statement of Class....

class bank
{
struct acc_data x;
```

```
public:
void input();
void withdrawal(int,float);
void deposit(int,float);
void display(int);

};

// Statement of customer information...

void bank:: input()
{
cout<<endl<<"   Enter the name of customer:- ";
gets(x.c_name);
cout<<" Enter account no:- ";
cin>>x.acc_no;
cout<<" Enter opening balance:- ";
cin>>x.balance;
cout<<" Enter acc_type (SA/CA):- ";
gets(x.acc_type);
}

//Statement of Withdrawal function...

void bank::withdrawal(int ac,float money)
{
if(x.acc_no==ac)
{
if(x.balance-1000>=money)
x.balance -= money;
else
cout<<"
Sorry, Insufficient Balance";
}
else
cout<<"
** Sorry, Account no. is not found ** ";
}

// Statement of deposite function...

void bank::deposit(int ac,float money)
{
if(ac==x.acc_no)
x.balance += money;
else
cout<<"
Sorry, Account no. is not found...";
}

// Statement of display Funtion...

void bank::display(int ac)
{
if(ac==x.acc_no)
{
cout<<"
Your acc_no is: "<<x.acc_no;
cout<<"
Your name is: ";
puts(x.c_name);
cout<<"Your current balance is: "<<x.balance;
cout<<"
YOur type of account is: "<<x.acc_type;
}
}

// statement of main Function....

main()
{

bank obj; //creating object for class bank
int ac;
float money;
int c;
clrscr();
do
{
clrscr();
cout<<endl<<"

Welcome to Banking System";
cout<<"
_____";
cout<<endl<<"
1:Open New Account of the customer.";
cout<<endl<<"
2:Withdrawal the money.";
cout<<endl<<"
3:Deposit the money.";
cout<<endl<<"
4:Display information about customer.";
cout<<endl<<"
5:quit";
cout<<endl<<"
what is your choice: ";
cin>>c;

switch(c)
{
case 1:
```

```
obj.input();
break;
case 2:
cout<<"
Enter account no: ";
cin>>ac;
cout<<"
enter the money to withdrawal: ";
cin>>money;
obj.withdrawal(ac,money);
break;
case 3:

cout<<"
enter acc no: ";
cin>>ac;
cout<<"
enter the money to deposit: ";
cin>>money;
obj.deposit(ac,money);
break;
case 4:
cout<<"
enter acc no: ";
cin>>ac;
obj.display(ac);
break;
case 5:
exit(0);
default:
cout<<"
Sorry, unable to process.. Try again later
";
}
cout<<"
Press Enter to continue...";
getch();

}while(c!=5);

return 0;
}
```

Posted by: Ravi Ranjan

Contact Ravi Ranjan

```
#include<iostream>
#include<string>
using namespace std;
class bank
{
string name;
int accNo;
string accountType;
int balance;
public:
bank()
{
name = "abc";
accNo = 123;
accountType ="Savings";
balance =0;
}
void deposit(int amount)
{
balance = balance + amount;
}
void withdrawl(int amount)
{
if(balance>amount)
{
balance = balance - amount;

}
else
{
cout<<"balance is less than specified amount.
Pls specify less amount.";
}
}
void displayAccountinfo()
{
cout<<"AccountNo: "<<accNo<<endl;
cout<<"Name: "<<name<<endl;
cout<<"AccountType: "<<accountType<<endl;
cout<<"Balance: "<<balance<<endl;
}
};

int main()
{
bank ob;
ob.deposit(1000);
ob.withdrawl(500);
ob.displayAccountinfo();
getchar();
}
```

Posted by: hanumanth

Contact hanumanth

```
//BankAccount.h
#ifndef BANK_ACCOUNT_H
#define BANK_ACCOUNT_H

enum ACCOUNT_TYPE
{
newHolder = 0,
Savings,
TermDeposit,
VPF
};

class BankAccount
{
private:
CString strAccHolderName;
CString strAccNumber;
ACCOUNT_TYPE accType;
double  lfAmount;
public:
BankAccount();
~BankAccount();
bool AccountOpen(CString, CString,
ACCOUNT_TYPE, double);
bool DepositAmount(double);
bool WithdrawAmount(double);
bool DisplayDetails();
};

#endif

//BankAccount.cpp
using namespace std;
#include"BankAccount.h"
#include<iostream.h>
BankAccount::BankAccount()
{
strAccHolderName = strAccNumber = "";
accType = newHolder;
lfAmount = 0;
}

BankAccount::~BankAccount()
{
}

bool BankAccount::AccountOpen(CString
szAccHolderName, CString czAccNumber,
ACCOUNT_TYPE eAccType, double amount)
{
strAccHolderName = czAccHolderName;
strAccNumber = czAccNumber;
accType = eAccType;
lfAmount = amount;
return true;
}

bool BankAccount::DepositAmount(double
amount)
{
if(amount > 0)
{
lfAmount += amount;
return true;
}
else
return false;
}

bool BankAccount::WithdrawAmount(double
amount)
{
if(lfAmount >= amount)
{
lfAmount -= amount;
return true;
}
else
return false;
}

bool BankAccount::DisplayDetails()
{
cout<<"
Account holder Name : "<<strAccHolderName;
cout<<"
Amount              : "<<lfAmount;
return true;
}

//BankAccountUser.cpp
#include"BankAccount.h"
void main()
{
BankAccount    objAccount;
objAccount.OpenAccount("Kalpana",
"0089A513", Savings, 5000);
```

```
objAccount.DisplayDetails();
objAccount.DepositAmount(3000);
objAccount.DisplayDetails();
objAccount.WithdrawAmount(2000);
objAccount.DisplayDetails();
return;
}

// Output to console
Account holder Name : Kalpana
Amount              : 5000.0
Account holder Name : Kalpana
Amount              : 8000.0
Account holder Name : Kalpana
Amount              : 6000.0
```

Posted by: Kalpana

Contact Kalpana

```
class bank
{
private:
string name;
double acc_no;
string acc_type;
double bal;

public:
bank(string Name="",double Acc_no=0,string
Acc_type="",double Bal=0)
{
name=Name;
acc_no=Acc_no;
acc_type=Acc_type;
bal=Bal;
}
void deposit(double dep)
{
bal=bal+dep;
}

void withdraw(double amt)
{
if(bal<amt)
cout<<"No Enough fund in your
account"<<endl;
else
bal=bal-amt;
}
void statement()
{
cout<<"Name:"<<name<<endl;
cout<<"ACCount Number:"<<acc_no<<endl;
cout<<"Account Type:"<<acc_type<<endl;
cout<<"Balance:"<<bal<<endl;

}
};

int main()
{
bank
cust1("Gates",10023304754,"SB",2000),cust2;
cust1.statement();
cust2.statement();
cust1.withdraw(1000);
cust2.withdraw(500);
cust1.deposit(500);
cust2.deposit(3000);
cust1.statement();
cust2.statement();
return 0;
}
```

Q2.

. 1. Explain the differences between Object Oriented and Traditional Analysis & Design Techniques.

Traditional Analysis & Design Techniques	Object Oriented
Used to develop the Traditional Projects that uses procedural programming.	Used to develop Object-oriented Projects that depends on Object- Oriented programming.
Uses common processes likes: analysis, design, implementation, and testing.	Uses UML notations likes: use case, class diagram, communication diagram, development diagram and sequence diagram.
Depends on the size of projects and type of projects.	Depends on the experience of the team and complexity of projects through the numbers of objects.
Needs to large duration sometimes to development the large projects.	Need to more time than Traditional approach and leads that to more cost.
The problem of Traditional approach using classical life cycle	The object-oriented software life cycle identifies the three traditional activities of analysis, design, and implementation

2. Briefly explain the differences between encapsulation and abstraction giving suitable example
   - Encapsulation : Wrapping up of data and methods into a single unit is Encapsulation (e.g. Class)
   - Abstraction : It is an act of representing only the essential things without including background details. (e.g. Interface)
3. Briefly explain what is "polymorphism"?
   - Polymorphism is often considered the most powerful feature of object-oriented programming. Greek for "many forms," polymorphism is the ability to hide alternative implementations behind a common interface.
   - This concept leverages inheritance and encapsulation among other OO concepts. Polymorphism is the ability for objects of different types to respond to messages of the same type.
4. State four (04) benefits of using object oriented programming?
   - 1. Testability/Increased Quality (automated testing can increase speed of testing and increase quality)
   - 2. Code re-use (Polymorphism, Generics, Interfaces)
   - 3. Code extensibility
   - 4. Catch errors at compile time rather than at runtime.
   - 5. Maintainability: If designed correctly, any tier of the application can be replaced by another that implements the correct interface(s), and the application will still work (can use multiple user interfaces, can swap out data providers, etc.).
   - 6. Reduces large problems to smaller, more manageable ones.
   - 7. Fits the way the real world works. It is easy to map a real world problem to a solution in OO code.

Q3.

Stages of software development process

1.Planning: Without the perfect plan, calculating the strengths and weaknesses of the project, development of software is meaningless. Planning kicks off a project flawlessly and affects its progress positively.

2. Analysis: This step is about analyzing the performance of the software at various stages and making notes on additional requirements. Analysis is very important to proceed further to the next step.

3. Design: Once the analysis is complete, the step of designing takes over, which is basically building the architecture of the project. This step helps remove possible flaws by setting a standard and attempting to stick to it.

4. Development & Implementation: The actual task of developing the software starts here with data recording going on in the background. Once the software is developed, the stage of implementation comes in where the product goes through a pilot study to see if it's functioning properly.

5. Testing: The testing stage assesses the software for errors and documents bugs if there are any.

6. Maintenance: Once the software passes through all the stages without any issues, it is to undergo a maintenance process wherein it will be maintained and upgraded from time to time to adapt to changes.

What is a model?

Modeling is an abstract representation Of some complex knowledge or design to help communicate Complex software designs that would be difficult for you to describe textually can readily be conveyed through diagram. Modeling provides three key benefits: Visualization, complexity management and clear communication.

Stages of USDP

- Inception: defining the scope and objectives of the project, as well as the business case
- Elaboration: capturing the crucial requirements, developing and validating the architecture of the software system, and planning the remaining phases of the project
- Construction: implementing the system in an iterative and incremental fashion based on the architecture developed in the previous phase
- Transition: testing and releasing the system

What are the best practice of USDP?

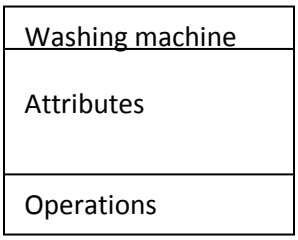
Q4

1.There are 4 kinds of things in the UML. Briefly explain them giving examples?

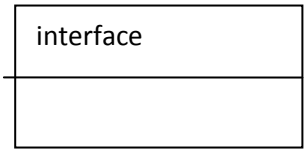
1.Structural things

Class diagram-class represents set of objects having similar responsibilities

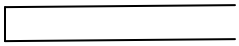
Eg-



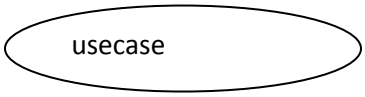
Interface- interface defines set of operations which specify responsibility of a class



Collaboration- collaboration defines interaction between elements



Usecase- usecase represents set of action and their performed by a system for specific goal

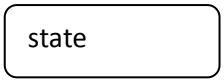


2. Behavioral things

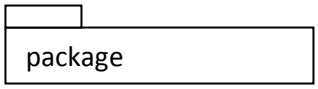
Interaction – Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task

Message

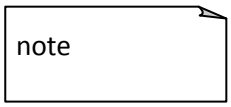
Statecthechine- state machine is useful when the state of an object in its lifecycle important. it defines the sequence of states an object goes through in response to events .



3. Grouping things- package is the only one grouping thing available for gathering structural and behavioral things



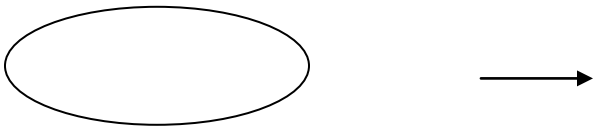
4. Annotational things- note is the only one Annotational thing available



2. Briefly explain the usecase model?

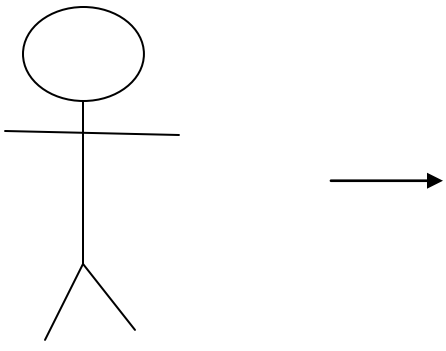
Primarily used to visualize the usecase corresponding actors and their interaction

The use case is description of a system's behavior from a user's standpoint



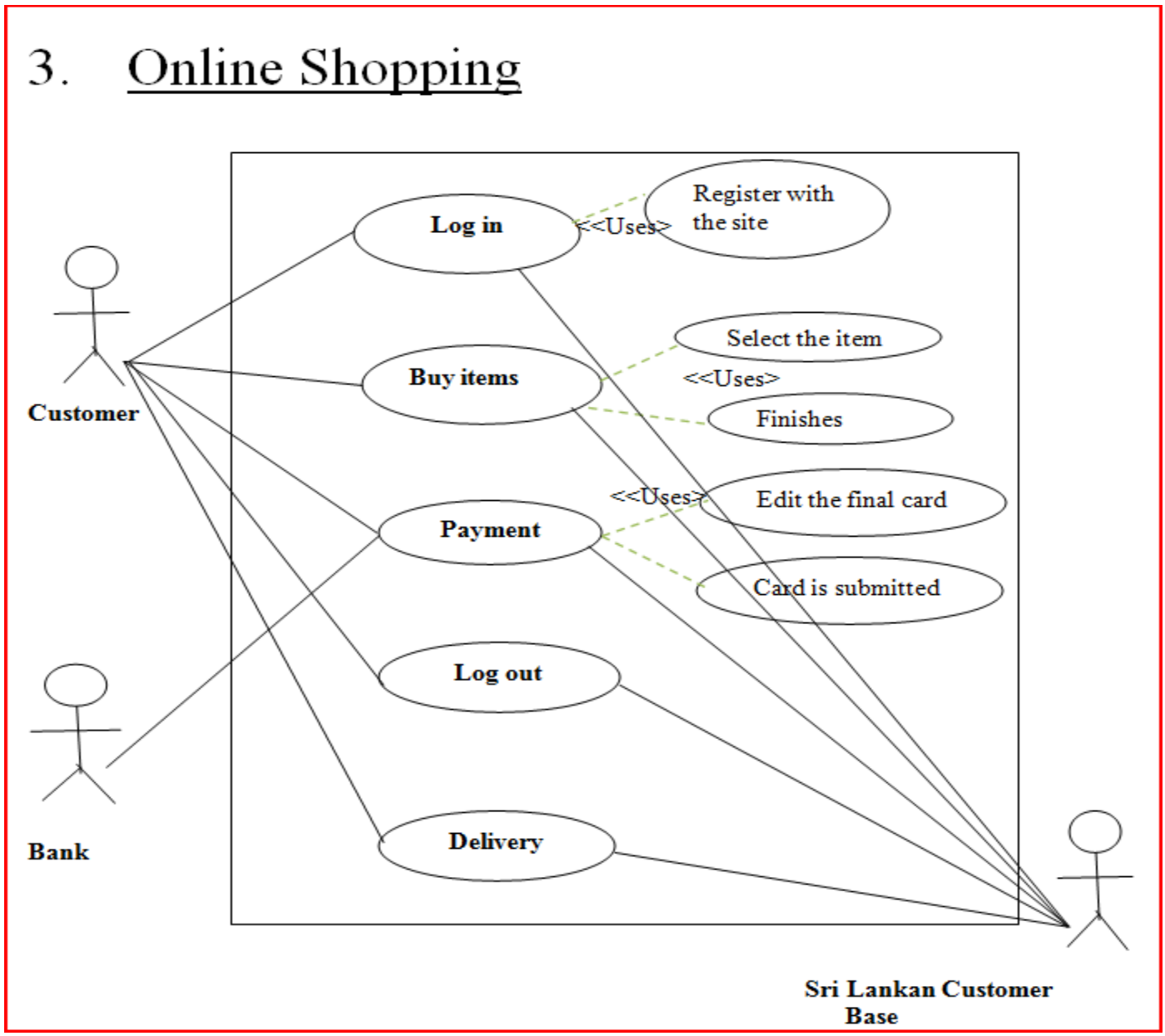
Usecase

Follow



Actor

3.Use case diagram.



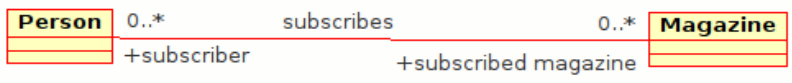
Q5.

1. Briefly explain the Class diagram in UML?

a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes.

In the diagram, classes are represented with boxes which contain three parts:

- The upper part holds the name of the class
- The middle part contains the attributes of the class
- The bottom part gives the methods or operations the class can take or undertake

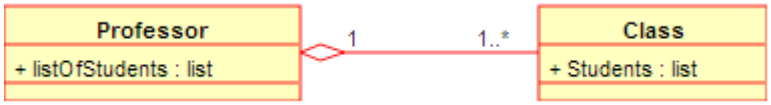


2.Explain the following relationships

a.Association

An *association* represents a family of links. Binary associations (with two ends) are normally represented as a line. An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties.

b.Aggregation



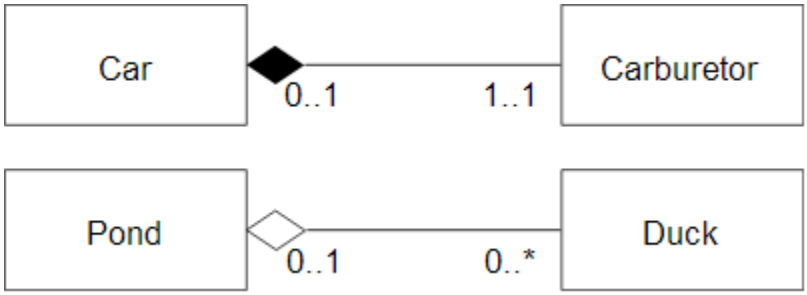
Aggregation is a variant of the "has a" or association relationship; aggregation is more specific than association. It is an association that represents a part-whole or part-of relationship. As a type of association, an aggregation can be named and have the same adornments that an association can. However, an aggregation may not involve more than two classes.

Aggregation can occur when a class is a collection or container of other classes, but where the contained classes do not have a strong *life cycle dependency* on the container—essentially, if the container is destroyed, its contents are not.

In UML, it is graphically represented as a *hollow* diamond shape on the containing class end of the tree with a single line that connects the contained class to the containing class. The aggregate is semantically an extended object that is treated as a unit in many operations, although physically it is made of several lesser objects.

Composition

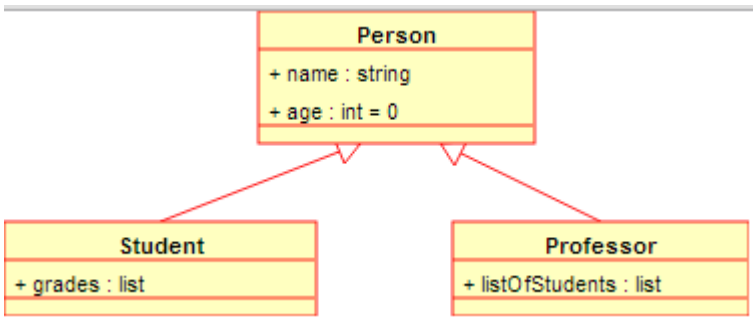
Composition is a stronger variant of the "owns a" or association relationship; composition is more specific than aggregation.



Composition usually has a strong *life cycle dependency* between instances of the container class and instances of the contained class(es): If the container is destroyed, normally every instance that it contains is destroyed as well. (Note that, where allowed, a part can be removed from a composite before the composite is deleted, and thus not be deleted as part of the composite.)

The UML graphical representation of a composition relationship is a *filled* diamond shape on the containing class end of the tree of lines that connect contained class(es) to the containing class.

c.Generalization



The Generalization relationship ("is a") indicates that one of the two related classes (the *subclass*) is considered to be a specialized form of the other (the *super type*) and superclass is considered as '*Generalization*' of subclass

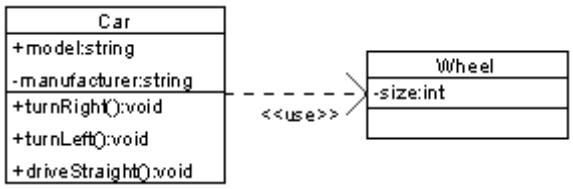
The generalization relationship is also known as the *inheritance* or *"is a"* relationship.

The *super class* (base class) in the generalization relationship is also known as the *"parent"*, *super class*, *base class*, or *base type*.

The *subtype* in the specialization relationship is also known as the *"child"*, *subclass*, *derived class*, *derived type*, *inheriting class*, or *inheriting type*.

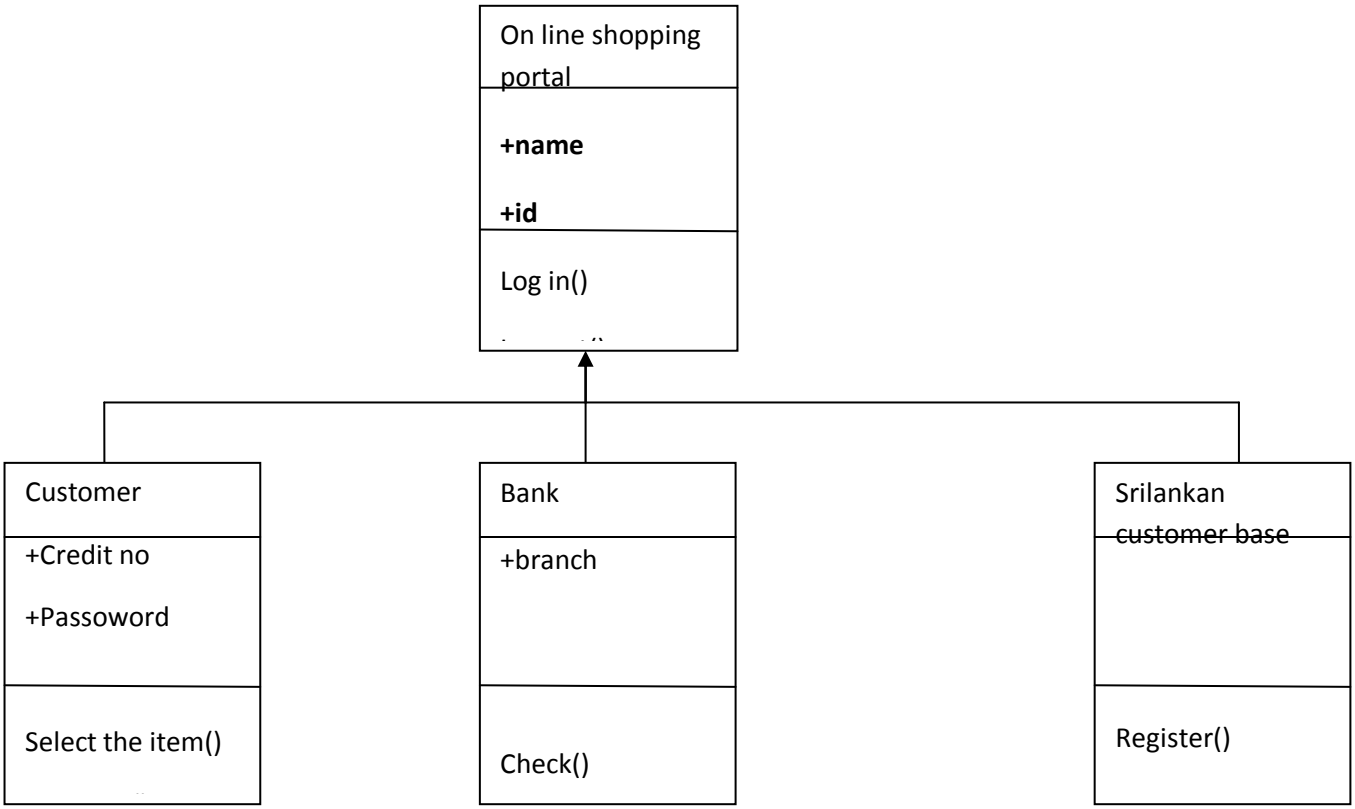
d.Dependancy:

Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time. One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class. This is different from an association, where an attribute of the dependent class is an instance of the independent class.



Class diagram showing dependency between "Car" class and "Wheel" class (An even clearer example would be "Car depends on Wheel", because Car already *aggregates* (and not just *uses*) Wheel)

3.Draw the class diagram for the case study given in question no four(04)?



Q6)

1) why do you model components diagram?

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

So from that point component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files etc.

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

A single component diagram cannot represent the entire system but a collection of diagrams are used to represent the whole.

So the purpose of the component diagram can be summarized as:

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

Brifly explain, if you need to show the physical releationship between software components and the hardware in the delivered system,which diagram can you use?

There is a strong link between components diagrams and deployment diagrams Deployment diagrams

Show the physical relationship between hardware and software in a system

Hardware elements:

Computers (clients, servers)

Embedded processors

Devices (sensors, peripherals)

Are used to show the nodes where software components reside in the run-time system

Deployment diagram

Contains nodes and connections

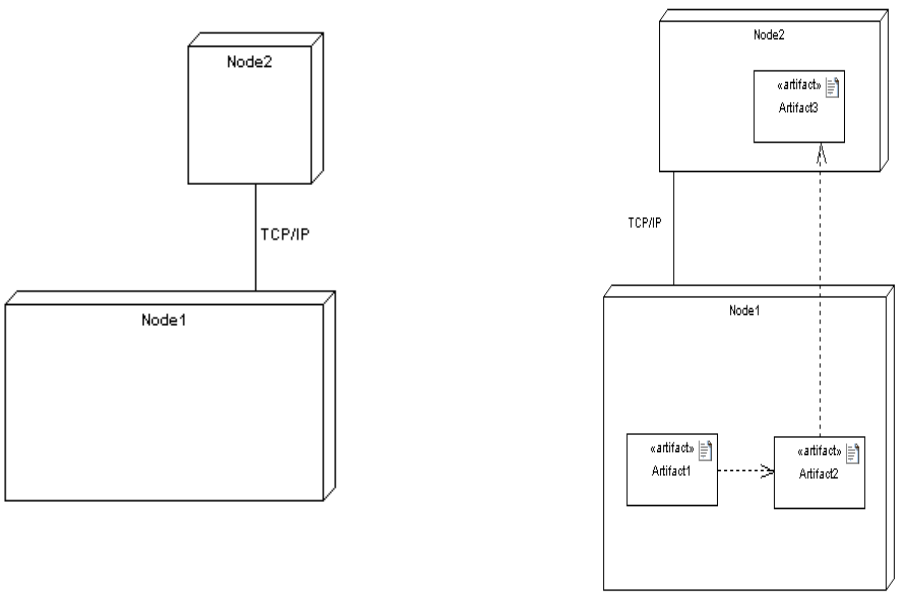
A node usually represent a piece of hardware in the system

A connection depicts the communication path used by the hardware to communicate

Usually indicates the method such as TCP/IP

3)

Draw the package diagram for the case study given in question no four(04)?

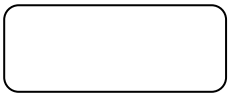
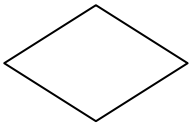
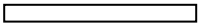


7.

What is the different between activity diagram and state diagram?

Activity Diagram	State Diagram
activity diagram is for describe process like flowcharts	State diagram shows the object undergoing a process. It gives a clear picture of the changes in the object's state in this process.
Activity is an action. You are doing something.	The State is the current value of something.
Activity shows the workflow behavior of an operation as set of action.	State diagram shows the dynamic behavior of an object
activity diagrams are activity centric	statecharts are state centric.
activity diagram is typically used for modeling the sequence of activities in a process.	Statechart is better suited to model the discrete stages of an object's lifetime.

1. Explain six(06) symbols used in activity diagram

Symbol Name	Symbol	Use
Initial Node		The initial node is the starting point an activity
Final Node		The activity final node indicates that an activity is completed.
Activity		An activity diagram illustrates one individual activity.
Action		Action is an individual step within an activity.
Decision		It represents a conditional branch point or decision node. A decision node has one input two or more output.
Synchronization bar		Parallel flows of activities.
Edge(control Flow)		Connect the individual components of activity.

2. Draw activity diagram to the above mention case statement