1. The following is an example of the FQDN (fully qualified domain name) on the Internet: nwest.sales.DomainName.com It consists of different subdomains. Each such subdomain can be mapped onto a specific directory on the file system of the computer where the DomainName.com is hosted. Each such subdomain can have different HTML files, which can be accessed through a URL. Thus a subdomain and the set of HTML files can be viewed as two main components of a Web site.

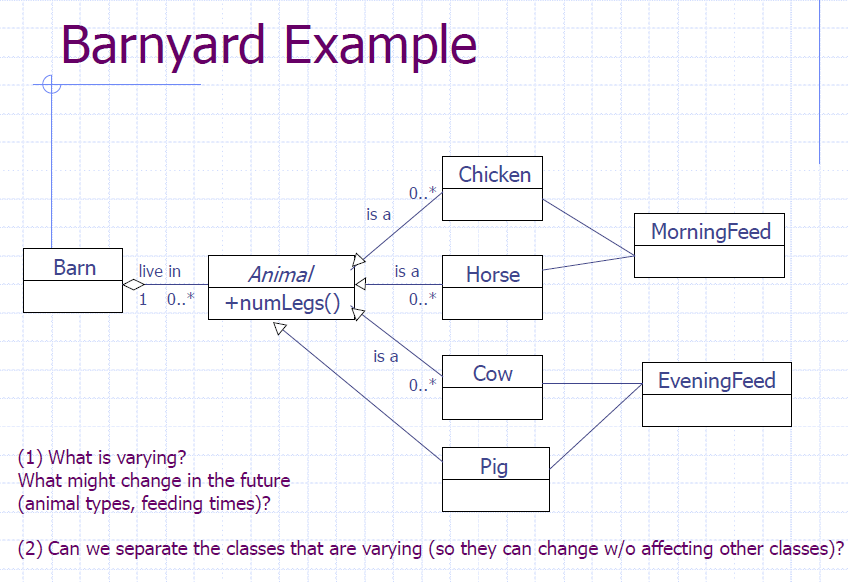
a. Define a subdomain hierarchy for an example domain.

b. Create an application that uses the appropriate pattern to:

i. Display the directory a given subdomain is mapped onto

ii. Display the URLs of Web site components (either subdomains or single HTML files) in a uniform manner

1. Refactor the following Design



1. Customer service representatives at some commercial banks handle queries from their existing and potential customers using an online chat application. At peak times, each representative may need to work with more than one customer simultaneously. Design this communication mechanism between different User objects and Representative objects
2. Design an application that reads and writes different types of data (plain text, binary, etc.) to and from different destinations such as a file, a URL or a database. Apply the required pattern in designing the data read/write abstraction.
3. Design a code formatting application using the appropriate pattern. In general, programs can be written in any computer language (e.g., Java, VB, etc.) and a given program can be formatted in different ways such as simple text formatting, HTML formatting, color formatting and others. In effect, the interface for code formatting can be implemented in many different ways. Apply the required pattern to separate the interface from its implementations

6.IN THE FOLLOWING CODE FIND THE DEBT AND REFACTOR THE DESIGN

class Icon

{

    float speed, glow, energy;

    int x, y;

    int subtype; // spinner, slider or hopper

    bool clockwise; // need for spinner

    bool expand; // need for spinner

    bool vertical; // need for slider

    int distance; // need for slider

    bool visible; // need for hopper

    int xcoord, ycoord; // need for hopper

    void spin() { }

    void slide() { }

    void hop() { }

    // constructor must set subtype: client must pass value

    public Icon(unsigned value)

    {

        subtype = value; // use enum for readability

        // and then use conditional to set associated fields

    }

    public void move()

    {

        if (subtype == 1) { spin(); }

        else if (subtype == 2)

        {

            slide();

        }

        else

        {

            hop();

        }

    }

// tedious subtype checking: subtype drives flair details

   public void flair()

   {

       if (subtype == 1) { spin(); }

       else if (subtype == 2)

       {

           slide();

       }

       else

       {

           hop();

       }

   }

}

7. A typical online job site maintains employer-, candidate- and jobs-related data. Let us build an application using the appropriate pattern that displays the necessary user interface to allow a user to search for different employers and candidates in the database. For simplicity, let us consider only three fields for each search, which users can use to specify the search criteria.

Employer Search

– Name

– City

– Membership Renewal Date

Candidate Search

– Name

– Experience (minimum number of years)

– Skill Set

The required user interface (UI) for each of these searches requires a different combination of UI controls. In terms of implementation, the required set of UI controls can be placed in a Window container. The appropriate pattern can be used in this case with different objects constructing the Window object with the necessary UI controls and initializing them appropriately.

8. Let us design the following functionality for an online shopping site. A server side component receives the order information submitted by a user in the form of an XML string.The order XML is then parsed and validated to create an Order object. The Order object is finally saved to the disk.

A typical order XML record is shown follows:

<Order>

<LineItems>

<Item>

<ID>100</ID>

<Qty>1</Qty>

</Item>

<Item>

<ID>200</ID>

<Qty>2</Qty>

</Item>

</LineItems>

<ShippingAddress>

<Address1>101 Arrowhead Trail </Address1>

<Address2> Suite 100</Address2>

<City>Anytown</City>

<State>OH</State>

<Zip>12345</Zip>

</ShippingAddress>

<BillingAddress>

<Address1>2669 Knox St </Address1>

<Address2> Unit 444</Address2>

<City>Anytown</City>

<State>CA</State>

<Zip>56789</Zip>

</BillingAddress>

</Order>

Payment details are not included in XML in order to keep the example simple.Let us consider three types of orders(Listed in Table).The save method can be used by different client objects to save the Order object to disk.

The series of steps required for the creation of an Order object can be

summarized as follows:

Parse the input XML string

Validate the data

Calculate the tax

Calculate the shipping

Create the actual object with:

– Line items from the input XML string

– Tax and shipping details calculated as per the details listed in Table

