Modern programming tools and techniques computer science

Q: 1 Define abstraction, encapsulation, modularity and hierarchy in your own terms.

Ans:-Abstraction” Abstraction denotes the essential characteristics of an Object that differ it from other objects, and thereby providing a boundary that is relative to the perspective of the viewer.” Abstraction focuses on the outside-view of the Object, and helps separate its behavior from its implementation, Think of it this way, to you, your car is an utility that helps you commute, it has a steering wheel , brakes etcA? a,¬A¦ but from an engineers point of view the very same car represents an entirely different view point, to the engineer the car is an entity that is composed of sub elements such and engine with a certain horse power, a certain cubic capacity, its power conversion ratio etc. It is the same car that we are talking about, but its behavior and properties have been encapsulated to the perspective of the viewer. This is what abstraction is.

Encapsulation

“ Encapsulation is breaking down the elements of an abstraction that constitute to its structure and behavior. Encapsulation serves as the interface between abstraction and its implementation”.

To understand encapsulation better, let’s consider an animal such as a dog. We know that a dog barks, it is its behavior, a property that defines a dog, but what is hidden is , how it barks, its implementation, this is encapsulation. The hiding of the implementation details of a behavior that defines a property of an entity is Encapsulation.

Modularity

“ The art of partitioning a program into individual components so as to reduce its complexity

to some degree can be termed as Modularity” In addition to this, the division of the code into modules helps provide clear boundaries between different parts of the program, thereby allowing it to be better documented and defined. In other words Modularity is building abstraction into discrete units. The direct bearing of modularity in Java is the use of packages. Elements of Analysis and Design

Hierarchy (Inheritance)

Abstraction is good, but in most real world cases we find more abstractions than we can comprehend at one time, though Encapsulation will help us to hide the implementation, and modularity to crisply cluster logically related abstractions, at times, it just isn’t enough. This is when Hierarchy comes into the picture, a set of Abstractions together form a Hierarchy, by identifying these hierarchies in our design; we greatly simplify our understanding of the problem.

Single Inheritance

Single Inheritance is the most important part of “ is – a” hierarchy. When a class shares the” structure” of another class it is said to single inherit a base class. To understand the concept better, let’s try this. Consider the base class “ Animal”. To define a bear terms of and animal, we say a Bear “ is – a”

kind of Animal. In simpler terms, the bear single inherits the structure of an animal.

Multiple Inheritance

Multiple Inheritance can be defined as a “ part of” inheritance where the subclasses inherit the “ Behavior” of more than one base type.

Q: 2 Sketch the object-oriented design or the Card game Black-Jack. What are the key objects? What are the properties and behaviours of these objects? How does the object interact

Ans:-Blackjack Implementation

It must write three new classes and link them with all of the previous classes in the project. The first class, DealerHand, implements the algorithm of playing Blackjack from the dealer’s perspective.

The classcontains a field which keeps track of the current number of points in a hand, and a methodthat calls in a counter-controlled loop the method of the previous class GameDeck to deal

cards one at a time from the top of the deck. As cards are being dealt, the current number of points in the hand is updated accordingly. Another method of GameDeck returns the value of the above field. The next class, PlayerHand, is a subclass of DealerHand. It overrides the method

for dealing cards: the cards are still dealt in a loop, but the loop is sentinel-controlled this

time, and the method incorporates interaction with the user.

The third class, GameApp, contains the method main in which objects of

DealerHand and PlayerHand are created. Methods for dealing cards are invoked on these

objects. When these methods return, the winner of the game is determined according to

the standard Blackjack algorithm.

The specific details of the algorithms for calculating points in each hand and for

determining the winner of the game are figured out by students with practically no

assistance from the instructor. By this point in the course, the students are able to write

this code independently, making use of the techniques, concepts, syntax and basic

structures of the Java language that they have learned during the semester.

While the application could be created using any development environment, Ibelieve that its success in my class is dependent upon the use of BlueJ. BlueJ enables this

project in two ways: (1) as a very simple-to-use tool for writing and editing code, and (2)

through the provided sample code that allows users to create images onscreen without any

prior knowledge of Java graphics (e. g., the Swing API). Because BlueJ minimizes the

hurdles associated with graphics programming, novice students are able to create an

interesting and fun application, which helps them master the basics of the object-oriented

approach in the earliest stages of their CS coursework. As an example, suppose you want to write a program that plays the card game, Blackjack. Youcan use the Card, Hand, and Deck classes developed. However, a hand in the game of Blackjack is a little different from a hand of cards in general, since it must be possible to compute the “ value” of a Blackjack hand according to the rules of the game. The rules are as follows: The value of a hand is obtained by adding up the values of the cards in the hand. The value of a numeric card such as a three or a ten is its numerical value. The value of a Jack, Queen, or King is 10. The value of an Ace can be either 1 or 11. An Ace should be counted as 11 unless doing so would put the total value of the hand over 21.

One way to handle this is to extend the existing Hand class by adding a method that computes the Blackjack value of the hand. Here’s the definition of such a class:

```
public class BlackjackHand extends Hand {

public int getBlackjackValue() {

// Returns the value of this hand for the

// game of Blackjack.

int val; // The value computed for the hand.

boolean ace; // This will be set to true if the

// hand contains an ace.

int cards; // Number of cards in the hand.

val = 0;

ace = false;

cards = getCardCount();

for ( int i = 0; i < cards; i++ ) {

// Add the value of the i-th card in the hand.

Card card; // The i-th card;

int cardVal; // The blackjack value of the i-th card.
```

```
card = getCard(i);

cardVal = card. getValue(); // The normal value, 1 to 13.

if (cardVal > 10) {

cardVal = 10; // For a Jack, Queen, or King.

}

if (cardVal == 1) {

ace = true; // There is at least one ace.

}

val = val + cardVal;

}

// Now, val is the value of the hand, counting any ace as 1.

// If there is an ace, and if changing its value from 1 to

// 11 would leave the score less than or equal to 21,

// then do so by adding the extra 10 points to val.

if ( ace == true && val + 10 <= 21 )

val = val + 10;

return val;
```

} // end getBlackjackValue()

} // end class BlackjackHand

Q: 3 Sketch the object-oriented design of a system to control a Soda dispensing machine. What are the key objects? What are the properties and behaviours of these objects? How does the object interact? ANS:-The state machine’s interface is encapsulated in the “ wrapper” class. The wrappee hierarchy’s interface mirrors the wrapper’s interface with the exception of one additional parameter. The extra parameter allows wrappee derived classes to call back to the wrapper class as necessary. Complexity that would otherwise drag down the wrapper class is neatly compartmented and encapsulated in a polymorphic hierarchy to which the wrapper objectA delegates. Example

The State pattern allows an object to change its behavior when its internal state changes. This pattern can be observed in a vending machine. Vending machines have states based on the inventory, amount of currency deposited, the ability to make change, the item selected, etc. When currency is deposited and a selection is made, a vending machine will either deliver a product and no change, deliver a product and change, deliver no product due to insufficient currency on deposit, or deliver no product due to inventoryA depletion.

Identify an existing class, or create a new class, that will serve as the “ state machine” from the client’s perspective. That class is the “ wrapper” A class.

Create a State base class that replicates the methods of the state machine interface. Each method takes one additional parameter: an instance of the wrapper class. The State base class specifies any useful “ default” A behavior.

Create a State derived class for each domain state. These derived classes only override the methods they need toA override.

The wrapper class maintains a “ current” StateA object.

All client requests to the wrapper class are simply delegated to the current State object, and the wrapper object’s this pointer isA passed.

The State methods change the “ current” state in the wrapper object asA appropriate.

.

public class VendingMachine {A

A A A A private double sales; A

A A A A private int cans; A

A A A A private int bottles; A

A A A A A

A A A A public VendingMachine() {A

A A A A A A A A fillMachine(); A

A A A A }AA A A A

A A A A public void fillMachine() {A

A A A A A A A A sales = 0; A

A A A A A A A A cans = 10; A

A A A A A A A bottles = 5; A

A A A A }AA A A A A

A A A public int getCanCount() {return this. cans; }A

A A A A public int getBottleCount() {return this. bottles; }A

A A A A public double getSales() { return this. sales;}A

A A A A A

A A A A public void vendCan() {A

A A A A A A A A if (this. cans== 0) {A

A A A A A A A A A A A A A System. out. println(“ Sorry, out of cans.”); A

A A A A A A A A } else {A

A A A A A A A A A A A A this. cans -= 1; A

A A A A A A A A A A A A this. sales += 0. 6; A

A A A A A A A }AA A A }AA A A A AA A AA A A A AA A A A A

A A A A public static void main(String[] argv) {A

A A A A A A A A VendingMachine machine = new VendingMachine(); A

A A A A A A AA A A A }A A A A}Part BQ: 4 In an object oriented inheritance hierarchy, the objects at each level are more specialized than the objects at the higher levels. Give three real world examples of a hierarchy with this property.

ANS:- Single Inheritance

Java implements what is known as a single-inheritance model. A new class can subclass (extend, in Java terminology) only one other class. Ultimately, all classes eventually inherit from the Object class, forming a tree structure with Object as its root. This picture illustrates the class hierarchy of the classes in the Java utility package, java. util

The HashTable class is a subclass of Dictionary, which in turn is a subclass of Object. Dictionary inherits all of Object’s variables and methods (behavior), then adds new variables and behavior of its own. Similarly, HashTable inherits all of Object’s variables and behavior, plus all of Dictionary’s variables and behavior, and goes on to add its own variables and behavior.

Then the Properties class subclasses HashTable in turn, inheriting all the variables and behavior of its class hierarchy. In a similar manner, Stack and ObserverList are subclasses of Vector, which in turn is a subclass of Object. The power of the object-oriented methodology is apparent–none of the subclasses needed to re-implement the basic functionality of their superclasses, but needed only add their own specialized behavior.

However, the above diagram points out the minor weakness with the single-inheritance model. Notice that there are two different kinds of enumerator classes in the picture, both of which inherit from Object. An enumerator class

implements behavior that iterates through a collection, obtaining the elements of that collection one by one. The enumerator classes define behavior that both HashTable and Vector find useful. Other, as yet undefined collection classes, such as list or queue, may also need the behavior of the enumeration classes. Unfortunately, they can inherit from only one superclass.

A possible method to solve this problem would be to enhance some superclass in the hierarchy to add such useful behavior when it becomes apparent that many subclasses could use the behavior. Such an approach would lead to chaos and bloat. If every time some common useful behavior were required for all subsequent subclasses, a class such as Object would be undergoing constant modification, would grow to enormous size and complexity, and the specification of its behavior would be constantly changing. Such a “ solution” is untenable. The elegant and workable solution to the problem is provided via Java interfaces, the subject of the next topic.

Multiple inheritance

Some object-oriented programming languages, such as C++, allow a class to extend two or more superclasses. This is called multiple inheritance. In the illustration below, for example, class E is shown as having both class A and class B as direct superclasses, while class F has three direct superclasses.

Such multiple inheritance is not allowed in Java. The designers of Java wanted to keep the language reasonably simple, and felt that the benefits of multiple inheritance were not worth the cost in increased complexity. However, Java does have a feature that can be used to accomplish many of the same goals as multiple inheritance: interfaces.

Class hierarchies

Classes in Java form hierarchies. These hierarchies are similar in structure to many more

familiar classification structures such as the organization of the biological world

originally developed by the Swedish botanist Carl Linnaeus in the 18th century. Portions

of this hierarchy are shown in the diagram . At the top of the chart is the

universal category of all living things. That category is subdivided into several

kingdoms, which are in turn broken down by phylum, class, order, family, genus, and

species. At the bottom of the hierarchy is the type of creature that

biologists name using the genus and species together. In this case, the bottom of the

hierarchy is occupied by Iridomyrmex purpureus, which is a type of red ant. The

individual red ants in the world correspond to the objects in a programming language.

Thus, each of the individuals

is an instance of the species purpureus. By virtue of the hierarchy, however, that

individual is also an instance of the genus Iridomyrmex, the class Insecta, and the phylum

Arthropoda. It is similarly, of course, both an animal and a living thing. Moreover, each

red ant has the characteristics that pertain to each of its ancestor categories. For example,

red ants have six legs, which is one of the defining characteristics of the class Insecta.

Real example of hyrarchy

Ques5 How do methods System. out. print() and System. out. println() differ? Define a java constant equal to 2. 9979 X 108 that approximates the speed of light in meters per second.

ANS:-1) public class Area{

```
public static void main(String[] args){

int length = 10;

int width = 5;

// calling the method or implementing it

int theArea = calculateArea();

System. out. println(theArea);

}

// our declaration of the method

public static int calculateArea(){

int methodArea = length * width;

return methodArea;

}}
```

2) public static void printHeader(){

System. out. println(“ Feral Production”);

System. out. println(“ For all your Forest Videos”);

System. out. println(“ 427 Blackbutt Way”);

System. out. println(“ Chaelundi Forest”);

System. out. println(“ NSW 2473”);

System. out. println(“ Australia”);

}

System. out. println(“ String argument”)

System. out. print(“ String argument”)

In the first case, the code fragment accesses the println() method of the object referred to by the class variable named out of the class named System. In the second case, the print() method is accessed instead of the println() method

The difference between the two is that the println() method automatically inserts a newlineat the end of the string argument whereas the print() method leaves the display cursor at the end of the string argument

Define a java constant equal to 2. 9979 X 108 that approximates the speed of light inmeters per second.

Floating-point values can also be written in a special programmer’s style of scientific notation, in which the value is represented as a floating-point number multiplied by aintegral power of 10. To write a number using this style, you write a floating-point number in standard notation, followed immediately by the letter E and an integerexponent, optionally preceded by a + or – sign. For example, the speed of light inmeters per second is approximately

2. 9979 x 108

which can be written in Java as

2. 9979E+8

where the E stands for the words times 10 to the power. Boolean constants and character constants also exist and are described in subsequent chapters along with their corresponding types.

Q: 6 Write a code segment that defines a Scanner variable stdin that is associated with System. in. The code segment should than define to int variables a and b, such that they are initialized with the next two input values from the standard input stream.

Ans:-

Import java. util.*;

Public class mathfun

{

Public static void main(string[] args)

{

Scanner stdin= new scanner (system. in);

System. out. print(“ enter a decimal number”);

Double x= stdin. nextdouble();

System. out. print(enter another decimalnumber”);

Double y= stdin. nextdouble();

Double squarerootx= math. sqrt(x);

System. out. println(“ square root of “+x+” is+square rootx);

}}A

A System. out. println(“ PersontHeighttShoe size”); A

System. out. println(“==========”); A

System. out. println(“ Hannaht5’1? t7”); A

System. out. println(“ Jennat5’10? t9”); A

System. out. println(“ JJt6’1? t14”); A A

AAQ: 7 Separately identify the keywords, variables, classes, methods and parameters in the following definition: import java. util.*; public class test{public static void main(String[] args){Scanner stdin = new Scanner(System. in); System. out. print(“ Number:”); double n = stdin. nextDouble(); System. out. println(n + ” * ” + n + ” = ” + n * n);}}

Ans:- public static void main(String[] args)-method

double n = stdin. nextDouble();-variables

public , static, void ,-keywords

stdin println-keyword

test -class

double-parameters

–