different types of cohesion with example. Cohesion measures the semantic strength of relationship between components within a functional unit. Several levels of cohesion can be identified in necessary order of strength, these are : (a) Coincidental cohesion : The parts of a component are not related but simply bundled into a single component. Ex : Module miscellaneous functions such as : use customer record, display customer record, read transaction record. (b) Logical cohesion : Components that perform similar functions such as input, error handling and so on are put together in a single document. Ex : if record-type is student then display student record else if record-type is staff then display staff record (c) Temporal cohesion : A temporally cohesive module is one whose elements are functions that are related in time. Ex :Set counter to 0, Open student file, Clear error message variable, initialize array. (d) Communicational Cohesion : All of the elements of a component operate on the same input data or produce the same output data. Ex : Use customer account, find customer name, find customer loan balance etc. (e) Procedural cohesion : A procedurally cohesive module is on whose elements are involved m different activities but the activities are sequential. Ex : Use out record, write out record, read in record, pad numeric fields to zero. (f) Functional cohesion :Each part of the component is necessary for the execution of a single function. Ex: Comput

Explain COCOMO model.

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality. It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models. Bohem's Cocomo model takes three forms :

• Model 1 : Basic Cocomo model : This model is the starting point for project estimation which computes software development effort & cost as function of program size expressed in estimated lines of code. Basic cocomo model provides an approximate estimation of software costs and is given by

E = a (KLOC) b

• Model 2: Intermediate Cocomomodel :Intermediate cocomo makes use of cost drives and their multiples to estimate the cost.Model utilizes 15 drives such as product attribute, hardware attribute, personal attribute, project attribute etc for cost estimation. Ex: Computers, skilled professional, administrative staff etc.

Model 3: Complete Cocomo model :Complex systems are made up of sub-systems, each parameter of a module must be summed up to get complete cost estimation. There are six phases in this model they are : • Planning and requirements • System design• Detailed design • Module code and test • Integration and test • Cost Constructive model

IEEE structure of SRS document.

The IEEE standards recognize the fact that different projects may require their requirements to be organized differently, that is no method that is suitable for all projects. It provides different ways of structuring the SRS. The first two sections of SRS are the same in all of them. IEEE suggests the following structure for requirements document. 1.Introduction 1.1 Purpose of the requirement's document 1.2 Scope of the product 1.3 Definitions, acronyms and abbreviations 1.4 References to supporting documents 1.5 Overview of rest of SRS 2.General description 2.1 Product perspective 2.2 Product functions 2.3 User characteristics 2.4 General constraints 2.5 Assumptions and dependencies. 3.Functional requirements 4.Non-functional requirements 5.System architecture 6.System models

Waterfall model

This model is a software life cycle where the stages are depicted as cascading from one to another. It was described by W.W. Royce in 1970. As the figure implies one development stage should be completed before the next begins.

RequirementAnalysis SystemDesign Coding Unit/System Testing Deployment Maintenance

Advantages of Waterfall Model

i. Easy to explain to the user. ii. Stages and activities are well defined iii. Verification at each stage ensures early detection of errors iv. Widely used to identify and meet the milestones v. Establishes communication between customer and developer to meet the specifications.

Disadvantages of Waterfall Model

i. The next stage begins only after the previous stage is complete, making it rigid. ii. User training is not given much importance. iii. Interaction with the user takes place right at the beginning and then at the time of deployment, which creates a gap between the phases. iv. Due to its cascading flow there is very little interaction from the user.

Boehmia's Spiral Model

Spiral model was proposed by Boehm in 1988.

• Each loop in the spiral represents a phase of the software process. • Innermost loop is concerned with system feasibility, next loop system requirement, followed by system design and so on

Each loop is split into 4 sectors: 1. Objective setting – Project risks are identified, alternative strategies depending on these risks may be planned. 2. Risk assessment and reduction - Project risks are identified ◊ detailed analysis carried ◊ steps taken to reduce risks. Ex: A prototype system (Toy like implementation with limited functional capabilities and low reliability just for the purpose of examining) 3. Development and Validation – Choosing the most appropriate development model. 4. Planning – Project is reviewed and decisions are made whether to continue with further loop of the spiral. Advantages of Spiral Model i. High amount of risk analysis ii. Supports large and high risk projects iii. Spiral model is one of the most flexible SDLC models. iv. Changes can be introduced later in the life cycle as well v. Project monitoring is easy as each loop requires a review from concerned people. Disadvantages of Spiral model i. When to stop the spiral process is not clear. ii. Cost involved in this model is usually high. iii. Does not work well for smaller projects. iv. Project's success is

highly dependent on the risk analysis phase.

SDLC is a sequence of activities carried out by analyst, designer and user to develop and implement an information system. These activities can be carried out in different stages. SDLC can be broadly classified into 7 phases. 1. Feasibility Study: The main aim is to determine whether the product is financially worthwhile and technically feasible. 2. Requirement analysis: In this phase the aim is to find exact requirement of the customers, Requirements are classified into a) Functional Requirements (Input /Output needs) and b) Non-functional requirements (Constraints like time, cost etc). Finally a SRS document is prepared as an output.

3. System Design: Software architecture is derived from SRS document. A new system is designed according to the needs of the user. 4. Development: This is the actual phase where the system is developed. The whole design is built and implemented. 5. Testing: During implementation phase each module of the design is coded and each module is unit tested individually. This is to check if each individual module works correctly. This is the most critical phase. 6. Deployment: The developed system is handed over to the client. The old system is dispensed and the new system is put to operations and used. 7. Software Maintenance: In this phase adding enhancements, improvements and updates to the new versions are done. Different types of SDLC Models are: 1) Waterfall Model, 2) Prototype model, 3) Prototype model, 4) Iterative enhancement model

B3.frequency of each vowel
import java.io. *; import java.

B4. static nested class and non-static inner classes.
class OuterClass{ static class
StaticNestedClass { public
void printStatus (){
System.out.println(&quot;I am inside static member&quot;);}} class
InnerClass{
public void showStatus (){
System.out.println(&quot;You are inside non static class&quot;);}}}
public class NestedInnerClass //non-static inner class{
public static void main (String [] args){
OuterClass.StaticNestedClass obj = new
OuterClass.StaticNestedClass();
obj.printStatus();
OuterClass objOuter = new OuterClass();
OuterClass.InnerClass objInner =
objOuter.new InnerClass ();
objInner.showStatus();}}

B5. Create a package containing a
package mypackage; public class
Student{
public void display (String name,int rollno,int marks){
System.out.println(&quot;Name:&quot; +name);
System.out.println(&quot;Rollno:&quot;
+rollno);
System.out.println(&quot;Marks:&quot;
+marks);}} package pack; import
mypackage. *; public class Academics {
public static void main (String [] args) {
System.out.println(&quot;-----Records of
students----&quot;);
System.out.println(&quot;\n Student-1:&quot;);
Student s1 = new Student();
s1.display(&quot;Ram&quot;, 1111, 90);
System.out.println(&quot;\n Student-2:&quot;);
Student s2 = new Student();
s2.display(&quot;Benny&quot;, 1112,98);}}

B6.invoke constructor class
BaseConstructor{
int x;
public BaseConstructor (int p) //base class constructor{
System.out.println(&quot;Base class constructor called&quot;);
x=p;
System.out.println(&quot;Value for X is initialized!&quot;);}
void display (){
System.out.println(&quot;Value for X
=&quot;+x);}}
class DerivedConstructor extends
BaseConstructor{
public DerivedConstructor () //derived class constructor{
super (25); //calling base class parameterized constructor
System.out.println(&quot;Derived class constructor called&quot;);}} class
ParameterizedSuper{
public static void main (String [] args){
DerivedConstructor obj = new
DerivedConstructor ();
obj.display(); //invoke base class method
display()}}

B8.data streams for reading import
java.io.FileOutputStream; import
java.io.DataOutputStream; import
java.io.FileInputStream; import
java.io.DataInputStream; import
java.io.IOException; class
DataStream{
public static void main(String[] args) throws
IOException{
FileOutputStream fos = new
FileOutputStream(&quot;da&quot;);
DataOutputStream dos = new
DataOutputStream(fos);
dos.writeInt(50000); dos.writeShort(2500);
dos.writeByte(120);
dos.close();
FileInputStream fis = new
FileInputStream(&quot;da&quot;);
DataInputStream dis = new
DataInputStream(fis);
System.out.println(&quot;Int:&quot;
+dis.readInt());
System.out.println(&quot;Short&quot;
+dis.readShort());
System.out.print(&quot;Byte:&quot;
+dis.readByte());
dis.close();}}

B3.frequency of each vowel
import java.io. *; import
java. util.Scanner; public
class Frequency{
public static void main (String [] args){ String
str;
int a=0,e=0,i=0,o=0,u=0;
Scanner obj = new Scanner (System.in);
try{
System.out.print(&quot;Enter a string&quot;);
str = obj.nextLine(); str=str.toLowerCase();
char s[]=str. toCharArray (); for(int
k=0;k&lt;s.length;k++){
if(s[k]==&#39;a&#39;) a=a+1; else
if(s[k]==&#39;e&#39;) e=e+1;
else
if(s[k]==&#39;i&#39;)
i=i+1; else
if(s[k]==&#39;o&#39;)
o=o+1;
else if(s[k]==&#39;u&#39;)
u=u+1;}
System.out.println(&quot;\n\t Vowel \t
Frequency&quot;);
System.out.println(&quot;\t a \t&quot;+a);
System.out.println(&quot;\t e \t&quot;+e);
System.out.println(&quot;\t i \t&quot;+i);
System.out.println(&quot;\t o \t&quot;+o);
System.out.println(&quot;\t u \t&quot;+u);} catch
(Exception ex){
System.out.println(&quot;Invalid Input&quot;);}}}

B4. static nested class and non-static inner classes.
class OuterClass{ static class
StaticNestedClass { public
void printStatus (){
System.out.println(&quot;I am inside static member&quot;);}} class
InnerClass{
public void showStatus (){
System.out.println(&quot;You are inside non static class&quot;);}}}
public class NestedInnerClass //non-static inner class{
public static void main (String [] args){
OuterClass.StaticNestedClass obj = new
OuterClass.StaticNestedClass();
obj.printStatus();
OuterClass objOuter = new OuterClass();
OuterClass.InnerClass objInner =
objOuter.new InnerClass ();
objInner.showStatus();}}

B5. Create a package containing a
package mypackage; public class
Student{
public void display (String name,int rollno,int marks){
System.out.println(&quot;Name:&quot; +name);
System.out.println(&quot;Rollno:&quot; +rollno);
System.out.println(&quot;Marks:&quot;
+marks);}} package pack; import
mypackage. *; public class Academics {
public static void main (String [] args) {
System.out.println(&quot;-----Records of
students----&quot;);
System.out.println(&quot;\n Student-1:&quot;);
Student s1 = new Student();
s1.display(&quot;Ram&quot;, 1111, 90);
System.out.println(&quot;\n Student-2:&quot;);
Student s2 = new Student();
s2.display(&quot;Benny&quot;, 1112,98);}}

B6.invoke constructor class
BaseConstructor{
int x;
public BaseConstructor (int p) //base class constructor{
System.out.println(&quot;Base class constructor called&quot;);
x=p;
System.out.println(&quot;Value for X is initialized!&quot;);}
void display (){
System.out.println(&quot;Value for X
=&quot;+x);}}
class DerivedConstructor extends
BaseConstructor{
public DerivedConstructor () //derived class constructor{ super (25); //calling base class parameterized constructor
System.out.println(&quot;Derived class constructor called&quot;);}} class
ParameterizedSuper{
public static void main (String [] args){
DerivedConstructor obj = new
DerivedConstructor ();
obj.display(); //invoke base class method
display()}}

B8.data streams for reading import
java.io.FileOutputStream; import
java.io.DataOutputStream; import
java.io.FileInputStream; import
java.io.DataInputStream; import
java.io.IOException; class
DataStream{
public static void main(String[] args)
throws IOException{ FileOutputStream
fos = new
FileOutputStream(&quot;da&quot;);
DataOutputStream dos = new
DataOutputStream(fos);
dos.writeInt(50000);
dos.writeShort(2500);
dos.writeByte(120);
dos.close();
FileInputStream fis = new
FileInputStream(&quot;da&quot;);
DataInputStream dis = new
DataInputStream(fis);
System.out.println(&quot;Int:&quot;
+dis.readInt());
System.out.println(&quot;Short&quot;
+dis.readShort());
System.out.print(&quot;Byte:&quot;
+dis.readByte());
dis.close();}}