**Question 1**

Explain AutoBoxing and Unboxing feature of java.

Java 5 (and hence AspectJ 1.5) supports automatic conversion of primitive types (int, float, double etc.) to their object equivalents (Integer, Float, Double,...) in assignments and method and constructor invocations. This conversion is know as autoboxing.

Java 5 also supports automatic unboxing, where wrapper types are automatically converted into their primitive equivalents if needed for assignments or method or constructor invocations.

**Question 2**

Illustrate shallow and deep comparison in java with example.

Shallow Equal

When you compare a reference variable with another reference variable, if both the references are points to the same object, they are shallow equal.

Deep Equal

When you compare two reference variables, if both the objects (which are pointed by above reference variables) have same attributes they are deep equal.

**Question 3**

Explain the use of class Object and class Class in java.

Class

Instances of the class Class represent classes and interfaces in a running Java application. Contains helper methods that give information about methods, contructors and attributes defined in a class.

Object

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

**Question 4**

What are chained exceptions in java and how do you handle then?

An application often responds to an exception by throwing another exception. In effect, the first exception *causes* the second exception. It can be very helpful to know when one exception causes another. *Chained Exceptions* help the programmer do this.

The following are the methods and constructors in Throwable that support chained exceptions.

Throwable getCause()

Throwable initCause(Throwable)

Throwable(String, Throwable)

Throwable(Throwable)

**Question 5**

Explain final, finally and finalize methods significance in java.

**final** – constant declaration.

**finally** – The finally block *always* executes when the try block exits, except System.exit(0) call. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

**finalize()** – method helps in garbage collection. A method that is invoked before an object is discarded by the garbage collector, allowing it to clean up its state. Should not be used to release non-memory resources like file handles, sockets, database connections etc because Java has only a finite number of these resources and you do not know when the garbage collection is going to kick in to release these non-memory resources through the finalize() method.

**Question 6**

What is the output of this program?

**public** **class** TestFour {

**public** **void** finalize() **throws** Throwable {

System.*out*.println("Calling finalize");

}

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

**try** {

TestFour four = **new** TestFour();

four = **null**;

**for** (**int** i = 0; i < 5; i++) {

System.*runFinalization*();

System.gc();

}

} **catch** (OutOfMemoryError error) {

}

}

}

Completely JVM dependent and cannot be determined

**Question 7**

What is the output of this program?

**public** **class** TestFive {

**static** String *s*;

**static** **class** Inner {

**public** **void** Inner() {

s = "Set from Inner";

}

};

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

TestFive t = **new** TestFive();

*s* = "Setting from outer";

Inner in = **this**.**new** Inner();

System.*out*.println(*s*);

}

}

Compilation fails. Reason: this cannot be used in static context

**Question 8**

What is the output of this program?

**public** **interface** I1 {

String *NAME* = "codemonkeyism";

**public** **void** test();

}

**public** **interface** I2 {

String *NAME* = "stephan";

**public** **void** test();

}

**public** **class** TestSix **implements** I1, I2 {

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

TestSix I = new TestSix();

System.*out*.println(I.NAME);

}

**public** **void** test() {

}

}

Compilation fails: i.NAME is ambiguos

**Question 9**

What is the use of Comparator interface in java?

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's natural ordering, and the class's compareTo method is referred to as its natural comparison method.

Lists (and arrays) of objects that implement this interface can be sorted automatically by Collections.sort (and Arrays.sort). Objects that implement this interface can be used as keys in a sorted map or elements in a sorted set, without the need to specify a comparator.

**Question 10**

How is thread synchronization handled in java?

Thread synchronization in java is handled through the use of word synchronized that can be applied to a statement, method or instance in java.

Example:

public void addName(String name) {

synchronized(this) {

lastName = name;

nameCount++;

}

nameList.add(name);

}

**Question 11**

Write a program that given a positive integer gives the Fibonacci series upto that number?

(Formula for Fibonnaci series: Fn = Fn-1+Fn-2 and F0 = 0 and F1 = 1)

public class Example {

public static int Fibonacci(int n) {

if(n == 0 || n==1)

return n;

else

return fibonacci (n-1) + fibonacci (n-2);

}

}

**Question 12**

Given a Collection of integer’s build (pseudo code) a Binary Search Tree which supports the insertion, and search operation.

Search:

Boolean BinarySearchTree::search(int val)

{

Node\* next = this->root();

while (next != 0)

{

if (val == next->value())

{

return true;

}

else if (val < next->value())

{

next = next->left();

}

else if (val > next->value())

{

next = next->right();

}

}

//not found

return false;

}

Insert:

public void insert(int data) {

if (root == null) {

root = new TreeNode(data, null, null);

} else {

TreeNode current = root;

while (current != null) {

if (data < current.getData()) {

// insert left

if (current.getLeft() == null) {

current.setLeft(new TreeNode(data, null, null));

return;

} else {

current = current.getLeft();

}

} else {

// insert right

if (current.getRight() == null) {

current.setRight(new TreeNode(data, null, null));

return;

} else {

current = current.getRight();

}

}

}

}

}

**Question 13**

EMPLOYEE ADDRESS

EMP\_ID NAME ID ADDRESS EMP\_ID

1 X 1 ABC ADRESS 1

2 Y 2 EFG ADRESS 2

3 Z 3 L Address 4

4 L

a) Write a query to give a report of All Employee and their Address if any.

b) Give a List of Employees that don’t have any address.

1. Select e.emp\_id, e.name, a.id, a.address From EMPLOYEE e, ADDRESS a where e.emp\_id = a.emp\_id (+)
2. Select \* from EMPLOYEE where emp\_id not in (select emp\_id from ADDRESS)

**Question 14**

There is a HR department class that is concerned about an employee hire event, employee fire event and employee vacation event for employees in an organization and has to shoot an email to employee group each time these events occur. Employees should also have an opportunity to subscribe or unsubscribe from the email group.

Illustrate your design for solving this problem using UML or pseudo class structures.

**Question 15**

Explain with algorithm for the dining philosopher’s problem.

**MONITOR:**

**DATA:**

*condition* can\_eat[NUM\_PHILS];

enum states {THINKING, HUNGRY, EATING} state[NUM\_PHILS-1];

int index;

**INITIALIZATION:**

for (index=0; index<NUM\_PHILS; index++) {

flags[index] = THINKING;

}

**MONITOR PROCEDURES:**

/\* request the right to pickup chopsticks and eat \*/

*entry* void pickup(int mynum) {

/\* announce that we're hungry \*/

state[mynum] = HUNGRY;

/\* if neighbor's aren't eating, proceed \*/

if ((state[mynum-1 *mod* NUM\_PHILS] != EATING) &&

(state [mynum+1 *mod* NUM\_PHILS] != EATING)) {

state[mynum] = EATING;

}

/\* otherwise wait for them \*/

else *can\_eat[mynum].wait*;

/\* ready to eat now \*/

state[mynum] = EATING;

}

/\* announce that we're finished, give others a chance \*/

*entry* void putdown(int mynum) {

/\* announce that we're done \*/

state[mynum] = THINKING;

/\* give left (lower) neighbor a chance to eat \*/

if ((state [mynum-1 *mod* NUM\_PHILS] == HUNGRY) &&

(state [mynum-2 *mod* NUM\_PHILS] != EATING)) {

*can\_eat[mynum-1 mod NUM\_PHILS].signal*;

}

/\* give right (higher) neighbor a chance to eat \*/

if ((state [mynum+1 *mod* NUM\_PHILS] == HUNGRY) &&

(state [mynum+2 *mod* NUM\_PHILS] != EATING)) {

*can\_eat[mynum+1 mod NUM\_PHILS].signal*;

}

}

**PHILOSOPHER:**

/\* find out our id, then repeat forever \*/

me = get\_my\_id();

while (TRUE) {

/\* think, wait, eat, do it all again ... \*/

think();

pickup(me);

eat();

putdown(me);

}

**Question 16**

Explain the MVC pattern in web application

**Model:** The model object knows about all the data that need to be displayed. It is model who is aware about all the operations that can be applied to transform that object. It only represents the data of an application. The model represents enterprise data and the business rules that govern access to and updates of this data. Model is not aware about the presentation data and how that data will be displayed to the browser.

**View :** The view represents the presentation of the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains same if there is any modification in the business logic. In other words, we can say that it is the responsibility of the of the view's to maintain the consistency in its presentation when the model changes.

**Controller:**  Whenever the user sends a request for something then it always go through the controller. The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In  GUIs, the views and the controllers often work very closely together.

**Question 17**

What are ServletFilters

The Java Servlet specification version 2.3 introduces a new component type, called a filter. A *filter* dynamically intercepts requests and responses to transform or use the information contained in the requests or responses. Filters typically do not themselves create responses, but instead provide universal functions that can be "attached" to any type of servlet or JSP page.

**Question 18**

Explain servlet life cycle.

**The servlet life cycle methods defined in Servlet interface are init(), service() and destroy()**. The life cycle starts when container instantiates the object of servlet class and calls the init() method, and ends with the container calling the destroy() method.

The signature of this methods are shown below.

public void init(ServletConfig config) throws ServletException<br />

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException<br />

public void destroy()

service method generally delegates to doGet or doPost.

**Question 19**

What is URL Encoding?

URLs can only be sent over the Internet using the [ASCII character-set](http://www.w3schools.com/tags/ref_ascii.asp).

Since URLs often contains characters outside the ASCII set, the URL has to be converted. URL encoding converts the URL into a valid ASCII format.

URL encoding replaces unsafe ASCII characters with "%" followed by two hexadecimal digits corresponding to the character values in the ISO-8859-1 character-set.

URLs cannot contain spaces. URL encoding normally replaces a space with a + sign.

**Question 20**

What are JSP Expressions. Give example

Syntax of JSP Expressions are:

  <%="Any thing"   %>

JSP Expressions start with

Syntax of JSP Scriptles are with <%= and ends with  %>. Between these this you can put anything and that will converted to the String and that will be displayed.

Example:  
  <%="Hello World!" %>  
Above code will display 'Hello World!'.