Learn Java/J2EE core concepts and key areas

With

Java/J2EE Job Interview Companion

By

K.Arulkumaran
&
A.Sivayini

Technical Reviewers

Craig Malone
Stuart Watson
Arulazi Dhesiaseelan
Lara D'Albreo

Cover Design, Layout, & Editing

A.Sivayini

Acknowledgements

A. Sivayini
Mr. & Mrs. R. Kumaraswamipillai

Java/J2EE
Job Interview Companion

 ISBN 978-1-4116-6824-9

Please e-mail feedback & corrections (technical, grammatical and/or spelling) to
java-interview@hotmail.com

First Edition (220+ Q&A): **Dec 2005**
Second Edition (400+ Q&A): **March 2007**

Outline

SECTION	DESCRIPTION
	What this book will do for you?
	Motivation for this book
	Key Areas index
SECTION 1	Interview questions and answers on: **Java** ▪ Fundamentals ▪ Swing ▪ Applet ▪ Performance and Memory issues ▪ Personal and Behavioral/Situational ▪ Behaving right in an interview ▪ Key Points
SECTION 2	Interview questions and answers on: **Enterprise Java** ▪ J2EE Overview ▪ Servlet ▪ JSP ▪ JDBC / JTA ▪ JNDI / LDAP ▪ RMI ▪ EJB ▪ JMS ▪ XML ▪ SQL, Database, and O/R mapping ▪ RUP & UML ▪ Struts ▪ Web and Application servers. ▪ Best practices and performance considerations. ▪ Testing and deployment. ▪ Personal and Behavioral/Situational ▪ Key Points
SECTION 3	Putting it all together section. **How would you go about…?** 1. How would you go about documenting your Java/J2EE application? 2. How would you go about designing a Java/J2EE application? 3. How would you go about identifying performance problems and/or memory leaks in your Java application? 4. How would you go about minimizing memory leaks in your Java/J2EE application? 5. How would you go about improving performance of your Java/J2EE application? 6. How would you go about identifying any potential thread-safety issues in your Java/J2EE application? 7. How would you go about identifying any potential transactional issues in your Java/J2EE

	application? 8. How would you go about applying the Object Oriented (OO) design concepts in your Java/J2EE application? 9. How would you go about applying the UML diagrams in your Java/J2EE project? 10. How would you go about describing the software development processes you are familiar with? 11. How would you go about applying the design patterns in your Java/J2EE application? 12. How would you go about designing a Web application where the business tier is on a separate machine from the presentation tier. The business tier should talk to 2 different databases and your design should point out the different design patterns? 13. How would you go about determining the enterprise security requirements for your Java/J2EE application? 14. How would you go about describing the open source projects like JUnit (unit testing), Ant (build tool), CVS (version control system) and log4J (logging tool) which are integral part of most Java/J2EE projects? 15. How would you go about describing Service Oriented Architecture (**SOA**) and Web services?
SECTION 4	**Emerging Technologies/Frameworks** ▪ Test Driven Development (**TDD**). ▪ Aspect Oriented Programming (**AOP**). ▪ Inversion of Control (**IoC**) (Also known as **Dependency Injection**). ▪ Annotations or attributes based programming (xdoclet etc). ▪ Spring framework. ▪ Hibernate framework. ▪ EJB 3.0. ▪ JavaServer Faces (**JSF**) framework.
SECTION 5	**Sample interview questions …** ▪ **Java** ▪ **Web Components** ▪ **Enterprise** ▪ **Design** ▪ **General**
	GLOSSARY OF TERMS
	RESOURCES
	INDEX

Table of contents

What this book will do for you?

Have you got the time to read 10 or more books and articles to add value prior to the interview? This book has been written mainly from the perspective of **Java/J2EE job seekers** and **interviewers**. There are numerous books and articles on the market covering specific topics like Java, J2EE, EJB, Design Patterns, ANT, CVS, Multi-Threading, Servlets, JSP, emerging technologies like AOP (Aspect Oriented Programming), Test Driven Development (TDD), Dependency Injection DI (aka IoC – Inversion of Control) etc. But from an interview perspective it is not possible to brush up on all these books where each book usually has from 300 pages to 600 pages. The basic purpose of this book is to cover all the core concepts and key areas, which all Java/J2EE developers, designers and architects should be conversant with to perform well in their current jobs and to launch a successful career by doing well at interviews. The interviewer can also use this book to make sure that they hire the right candidate depending on their requirements. This book contains a wide range of topics relating to Java/J2EE development in a concise manner supplemented with diagrams, tables, sample codes and examples. This book is also appropriately categorized to enable you to choose the area of interest to you.

This book will assist all Java/J2EE practitioners to become better at what they do. Usually it takes years to understand all the core concepts and key areas when you rely only on your work experience. The best way to fast track this is to read appropriate technical information and proactively apply these in your work environment. It worked for me and hopefully it will work for you as well. I was also at one stage undecided whether to name this book "**Java/J2EE core concepts and key areas**" or "**Java/J2EE Job Interview Companion**". The reason I chose "**Java/J2EE Job Interview Companion**" is because the core concepts and key areas discussed in this book helped me to be successful in my interviews, helped me to survive and succeed at my work regardless what my job (junior developer, senior developer, technical lead, designer, contractor etc) was and also gave me thumbs up in code reviews. This book also has been set out as a handy reference guide and a roadmap for building enterprise Java applications.

Motivation for this book

I started using Java in 1999 when I was working as a junior developer. During those two years as a permanent employee, I pro-actively spent many hours studying the core concepts behind Java/J2EE in addition to my hands on practical experience. Two years later I decided to start contracting. Since I started contracting in 2001, my career had a much-needed boost in terms of contract rates, job satisfaction, responsibility etc. I moved from one contract to another with a view of expanding my skills and increasing my contract rates.

In the last 5 years of contracting, I have worked for 5 different organizations both medium and large on 8 different projects. For each contract I held, on average I attended 6-8 interviews with different companies. In most cases multiple job offers were made and consequently I was in a position to negotiate my contract rates and also to choose the job I liked based on the type of project, type of organization, technology used, etc. I have also sat for around 10 technical tests and a few preliminary phone interviews.

The success in the interviews did not come easily. I spent hours prior to each set of interviews wading through various books and articles as a preparation. The motivation for this book was to collate all this information into a single book, which will save me time prior to my interviews but also can benefit others in their interviews. What is in this book has helped me to go from **just a Java/J2EE job to a career in Java/J2EE** in a short time. It has also given me the job security that 'I can find a contract/permanent job opportunity even in the difficult job market'.

I am not suggesting that every one should go contracting but by performing well at the interviews you can be in a position to pick the permanent role you like and also be able to negotiate your salary package. Those of you who are already in good jobs can impress your team leaders, solution designers and/or architects for a possible promotion by demonstrating your understanding of the key areas discussed in this book. You can discuss with your senior team members about **performance issues, transactional issues, threading issues (concurrency issues)** and **memory issues**. In most of my previous contracts I was in a position to impress my team leads and architects by pinpointing some of the critical performance, memory, transactional and threading issues with the code and subsequently fixing them. Trust me it is not hard to impress someone if you understand the key areas.

For example:

- Struts action classes are not thread-safe (Refer **Q113** in Enterprise section).
- JSP variable declaration is not thread-safe (Refer **Q34** in Enterprise section).
- Valuable resources like database connections should be closed properly to avoid any memory and performance issues (Refer **Q45** in Enterprise section).
- Throwing an application exception will not rollback the transaction in EJB. (Refer **Q77** in Enterprise section).

The other key areas, which are vital to any software development, are a good understanding of some of **key design concepts, design patterns,** and a **modeling language** like **UML**. These key areas are really worthy of a mention in your resume and interviews.

For example:

- Know how to use inheritance, polymorphism and encapsulation (Refer **Q7**, **Q8**, **Q9**, and **Q10** in Java section.).
- Why use design patterns? (Refer **Q5** in Enterprise section).
- Why is UML important? (Refer **Q106** in Enterprise section).

If you happen to be in an interview with an organization facing serious issues with regards to their Java application relating to memory leaks, performance problems or a crashing JVM etc then you are likely to be asked questions on these topics. Refer **Q72 – Q74** in Java section and **Q123, Q125** in Enterprise section.

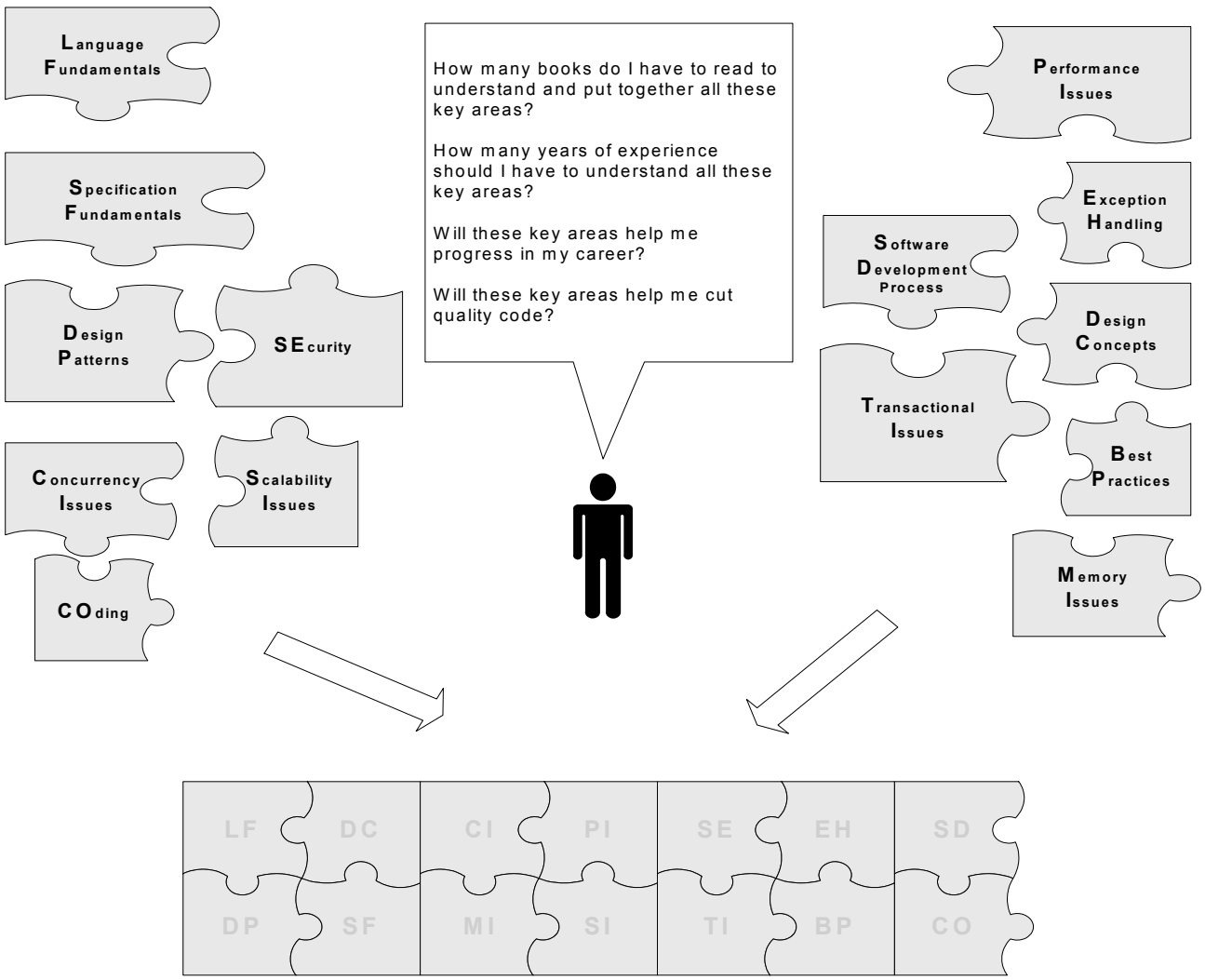
If you happen to be in an interview with an organization which is working on a pilot project using a different development methodology like agile methodology etc or has just started adopting a newer **development process** or **methodology** then you are likely to be asked questions on this key area.

If the team lead/architect of the organization you are being interviewed for feels that the current team is lacking skills in the key areas of **design concepts** and **design patterns** then you are likely to be asked questions on these key areas.

Another good reason why these key areas like transactional issues, design concepts, design patterns etc are vital are because solution designers, architects, team leads, and/or senior developers are usually responsible for conducting the technical interviews. These areas are their favorite topics because these are essential to any software development.

Some interviewers request you to write a small program during interview or prior to getting to the interview stage. This is to ascertain that you can code using object oriented concepts and design patterns. So I have included a **coding key area** to illustrate what you need to look for while coding.

- Apply OO concepts like inheritance, polymorphism and encapsulation: Refer **Q10** in Java section.
- Program to interfaces not to implementations: Refer **Q12, Q17** in Java section.
- Use of relevant design patterns: Refer **Q11, Q12** in How would you go about… section.
- Use of Java collections API and exceptions correctly: Refer **Q16** and **Q39** in Java section.
- Stay away from hard coding values: Refer **Q05** in Java section.

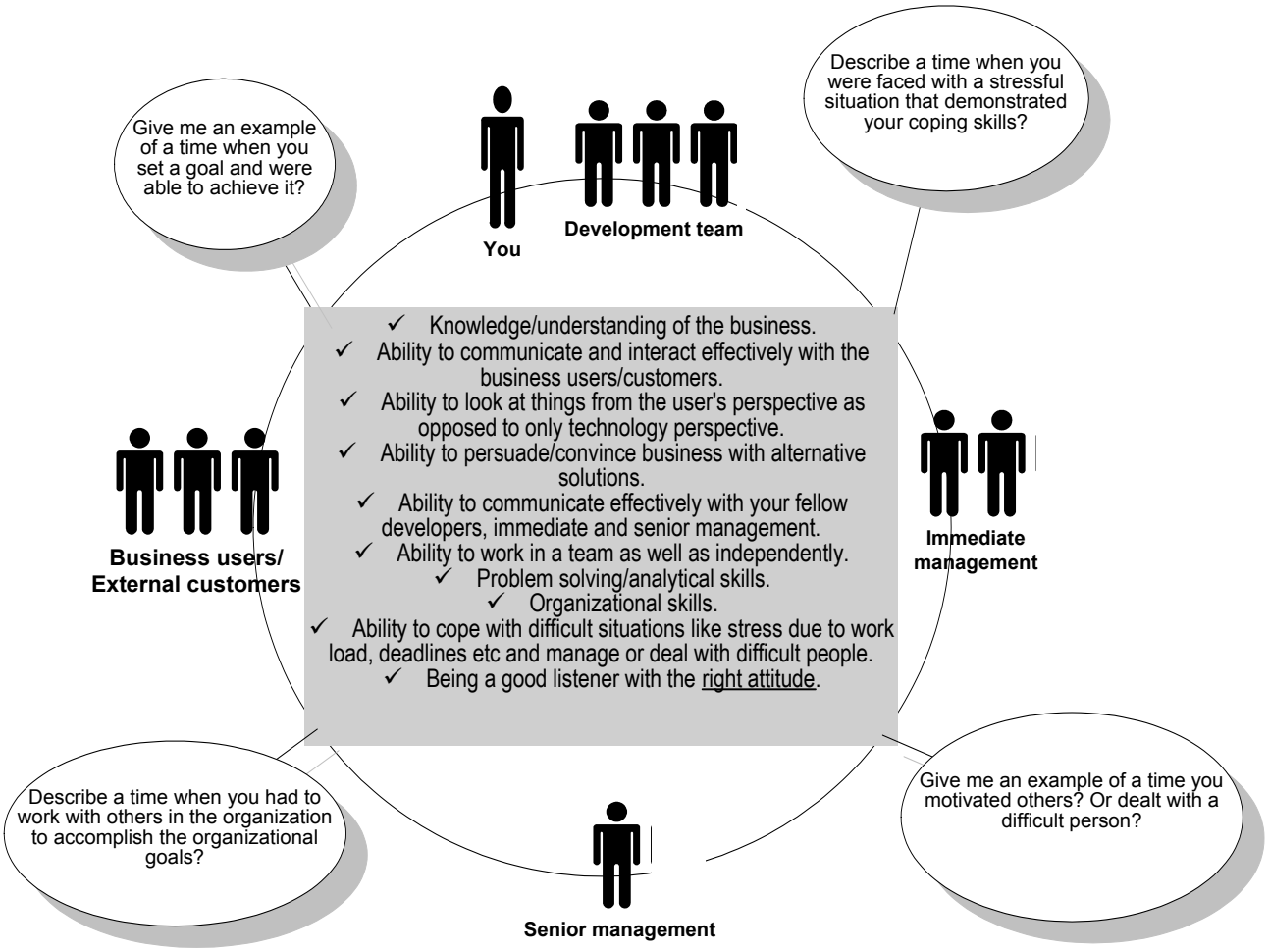


This book aims to solve the above dilemma.

My dad keeps telling me to find a permanent job (instead of contracting), which in his view provides better job security but I keep telling him that in my view in Information Technology the job security is achieved only by keeping your knowledge and skills sharp and up to date. The 8 contract positions I held over the last 5.5 years have given me broader experience in Java/J2EE and related technologies. It also kept me motivated since there was always something new to learn in each assignment, and not all companies will appreciate your skills and expertise until you decide to leave. Do the following statements sound familiar to you when you hand in your resignation or decide not to extend your contract after getting another job offer? “Can I tempt you to come back? What can I do to keep you here?” etc. You might even think why you waited so long. The best way to make an impression in any organizations is to understand and proactively apply and

resolve the issues relating to the **Key Areas** discussed in this book. But **be a team player, be tactful** and **don't be critical of everything, do not act in a superior way** and **have a sense of humor.**

"Technical skills must be complemented with good business and interpersonal skills."



IMPORTANT: Technical skills alone are not sufficient for you to perform well in your interviews and progress in your career. Your technical skills **must be complemented** with business skills (i.e. knowledge/understanding of the business, ability to communicate and interact effectively with the business users/customers, ability to look at things from the users' perspective as opposed to only from technology perspective, ability to persuade/convince business with alternative solutions, which can provide a win/win solution from users' perspective as well as technology perspective), ability to communicate effectively with your fellow developers, immediate and senior management, ability to work in a team as well as independently, problem solving/analytical skills, organizational skills, ability to cope with difficult situations like stress due to work load, deadlines etc and manage or deal with difficult people, being a good listener with the right attitude (It is sometimes possible to have "**I know it all attitude**", when you have strong technical skills. These are discussed in "**Java – Personal"** and "**Enterprise Java – Personal"** sub-sections with examples.

Quick Read guide: It is recommended that you go through all the questions in all the sections (all it takes is to read a few questions & answers each day) but if you are pressed for time or would like to read it just before an interview then follow the steps shown below:

-- Read/Browse all questions marked as **"FAQ"** in all four sections.
-- Read/Browse **Key Points** in Java and Enterprise Java sections.

Key Areas Index

I have categorized the core concepts and issues into **14 key areas** as listed below. These key areas are vital for any good software development. This index will enable you to refer to the questions based on **key areas**. Also note that each question has an icon next to it to indicate which key area or areas it belongs to. Additional reading is recommended for beginners in each of the key areas.

Key Areas	icon	---------- Question Numbers ----------			
		Java section	**Enterprise Java section**	**How would you go about…?**	**Emerging Technologies / Frameworks**
Language Fundamentals	LF	Q1-Q6, Q12-Q16, Q18-Q24, Q26-Q33, Q35-Q38, Q41-Q50, Q53-Q71	-		Q10, Q15, Q17, Q19
Specification Fundamentals	SF	-	Q1, Q2, Q4, Q6, Q7-Q15, Q17-Q19, Q22, Q26-Q33, Q35-Q38, Q41, Q42, Q44, Q46-Q81, Q89-Q93, Q95-Q97, Q99, 102, Q110, Q112-Q115, Q118-Q119, Q121, Q126, Q127, Q128	Q15	
Design Concepts	DC	Q1, Q7-Q12, Q15, Q26, Q22, Q56	Q2, Q3, Q19, Q20, Q21, Q31, Q45, Q91, Q94, Q98, Q101, Q106, Q107, Q108, Q109, Q111	Q02, Q08, Q09, Q15	Q3 - Q13, Q13, Q14, Q16, Q17, Q18, Q20
Design Patterns	DP	Q12, Q16, Q24, Q36, Q51, Q52, Q58, Q63, Q75	Q5, Q5, Q22, Q24, Q25, Q41, Q83, Q84, Q85, Q86, Q87, Q88, Q110, Q111, Q116	Q11, Q12	Q9 - Q13
Transactional Issues	TI	-	Q43, Q71, Q72, Q73, Q74, Q75, Q77, Q78, Q79	Q7	
Concurrency Issues	CI	Q15, Q17, Q21, Q34, Q42, Q46, Q62	Q16, Q34, Q72, Q78, Q113	Q6	
Performance Issues	PI	Q15, Q17,Q20-Q26, Q46, Q62, Q72	Q10, Q16, Q43, Q45, Q46, Q72, Q83-Q88, Q93, Q97, Q98, Q100, Q102, Q123, Q125, Q128	Q3, Q5	
Memory Issues	MI	Q26, Q34, Q37,Q38, Q42, Q51, Q73, Q74	Q45, Q93	Q3, Q4	
Scalability Issues	SI	Q23, Q24	Q20, Q21, Q120, Q122		
Exception Handling	EH	Q39, Q40	Q76, Q77		
Security	SE	Q10, Q35, Q70	Q12, Q13, Q23, Q35, Q46, Q51, Q58, Q81, Q92	Q13	
Best Practices	BP	Q17, Q25, Q39, Q72, Q73	Q10, Q16, Q39, Q40, Q41, Q46, Q82, Q124, Q125	Q1, Q2	

Software Development Process	SD	-	Q103-Q109, Q129, Q130, Q132, Q136	Q1, Q9, Q10, Q14	Q1, Q2
Coding[1]	CO	Q05, Q10, Q12, Q14 – Q21, Q23, Q25, Q26, Q33, Q35, Q39, Q51, Q52, Q55	Q10, Q18, Q21, Q23, Q36, Q38, Q42, Q43, Q45, Q74, Q75, Q76, Q77, Q112, Q114, Q127, Q128	Q11, Q12	
Frequently Asked Questions	FAQ	Q1, Q6, Q7, Q9, Q10, Q12, Q13, Q14, Q15, Q16, Q18, Q20, Q21, Q22, Q23, Q27, Q28, Q29, Q30, Q31, Q32, Q36, Q37, Q43, Q45, Q46, Q48, Q51, Q52, Q55, Q58, Q60, Q62, Q63, Q64, Q67, Q68, Q69, Q70, Q71 Q72 – Q86	Q1, Q2, Q3, Q7, Q10, Q11, Q12, Q13, Q16, Q19, Q22, Q24, Q25, Q27, Q28, Q30, Q31, Q32, Q34, Q35, Q36, Q39, Q40, Q41, Q42, Q43, Q45, Q46, Q48, Q49, Q50, Q52, Q53, Q61, Q63, Q65, Q66, Q69, Q70, Q71, Q72, Q73, Q76, Q77, Q82, Q83, Q84, Q85, Q86, Q87, Q90, Q91, Q93, Q95, Q96, Q97, Q98, Q100, Q101, Q102, Q107, Q108, Q110, Q113, Q115, Q116, Q118, Q123, Q124, Q125, Q126, Q129, Q130, Q131, Q133, Q134, Q135, Q136.	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q12, Q15	Q1, Q6, Q7, Q9, Q10, Q11, Q15, Q16, Q17, Q18

[1] Some interviewers request you to write a small program during interview or prior to getting to the interview stage. This is to ascertain that you can code using object oriented concepts and design patterns. I have included a coding key area to illustrate what you need to look for while coding. Unlike other key areas, the CO is not always shown against the question but shown above the actual section of relevance within a question.

SECTION ONE

Java – Interview questions & answers

KEY AREAS

- Language Fundamentals LF
- Design Concepts DC
- Design Patterns DP
- Concurrency Issues CI
- Performance Issues PI
- Memory Issues MI
- Exception Handling EH
- Security SE
- Scalability Issues SI
- Coding[1] CO

FAQ - Frequently **A**sked **Q**uestions

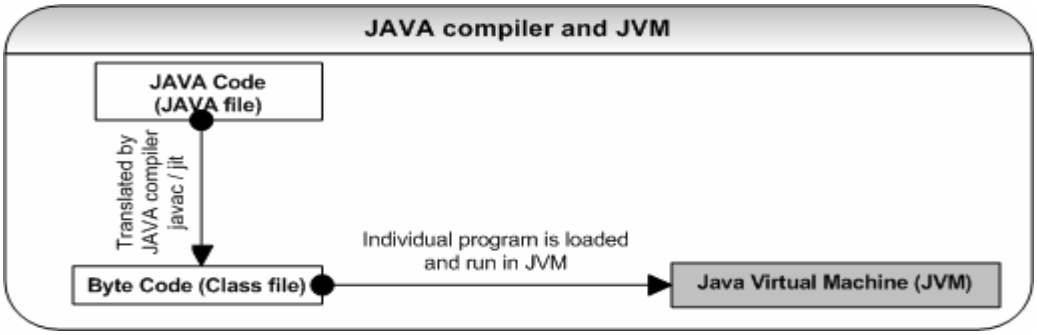
[1] Unlike other key areas, the CO is not always shown against the question but shown above the actual content of relevance within a question.

Java – Fundamentals

Q 01: Give a few reasons for using Java? LF DC FAQ
A 01: Java is a fun language. Let's look at some of the reasons:

- Built-in support for multi-threading, socket communication, and memory management (automatic garbage collection).
- Object Oriented (OO).
- Better portability than other languages across operating systems.
- Supports Web based applications (Applet, Servlet, and JSP), distributed applications (sockets, RMI, EJB etc) and network protocols (HTTP, JRMP etc) with the help of extensive standardized APIs (Application Programming Interfaces).

Q 02: What is the main difference between the Java platform and the other software platforms? LF
A 02: Java platform is a software-only platform, which runs on top of other hardware-based platforms like UNIX, NT etc.



The Java platform has 2 components:

- Java Virtual Machine (**JVM**) – 'JVM' is a software that can be ported onto various hardware platforms. Byte codes are the machine language of the JVM.
- Java Application Programming Interface (**Java API**) – set of classes written using the Java language and run on the JVM.

Q 03: What is the difference between C++ and Java? LF
A 03: Both C++ and Java use similar syntax and are Object Oriented, but:

- Java does not support pointers. Pointers are inherently tricky to use and troublesome.
- Java does not support multiple inheritances because it causes more problems than it solves. Instead Java supports **multiple interface inheritance**, which allows an object to inherit many method signatures from different interfaces with the condition that the inheriting object must implement those inherited methods. The multiple interface inheritance also allows an object to behave **polymorphically** on those methods. [Refer **Q9** and **Q10** in Java section.]
- Java does not support destructors but adds a finalize() method. Finalize methods are invoked by the garbage collector prior to reclaiming the memory occupied by the object, which has the finalize() method. This means you do not know when the objects are going to be finalized. **Avoid using finalize() method 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 resources through the finalize() method.
- Java does not include structures or unions because the traditional data structures are implemented as an object oriented framework (Java Collections Framework – Refer **Q16**, **Q17** in Java section).

- All the code in Java program is encapsulated within classes therefore Java does not have global variables or functions.
- C++ requires explicit memory management, while Java includes automatic garbage collection. [Refer **Q37** in Java section].

Q 04: What are the usages of Java packages? **LF**

A 04: It helps resolve naming conflicts when different packages have classes with the same names. This also helps you organize files within your project. **For example:** **java.io** package do something related to I/O and **java.net** package do something to do with network and so on. If we tend to put all .java files into a single package, as the project gets bigger, then it would become a nightmare to manage all your files.

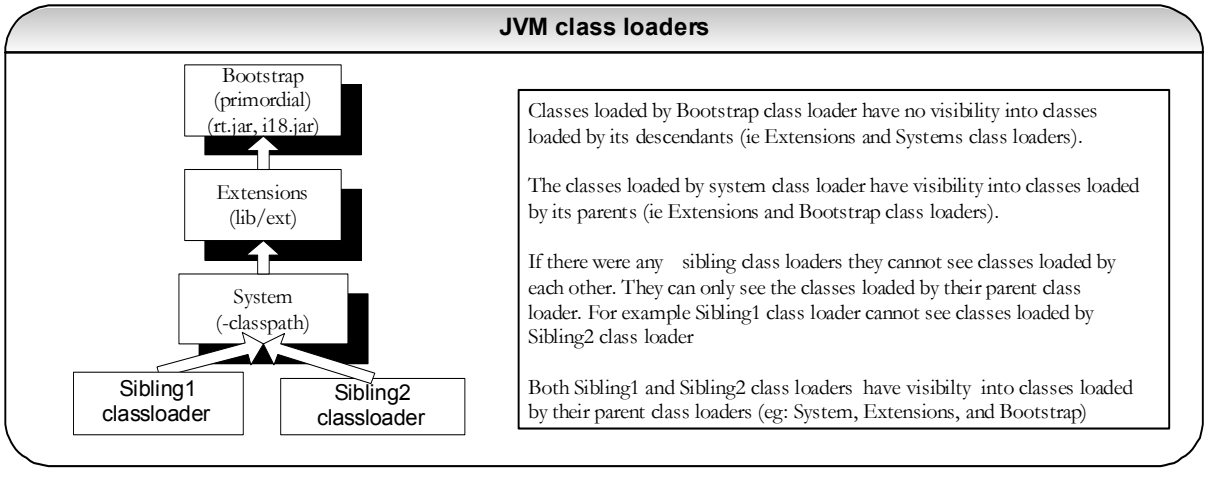
You can create a package as follows with **package** keyword, which is the first keyword in any Java program followed by **import** statements. The **java.lang** package is imported implicitly by default and all the other packages must be explicitly imported.

```
package com.xyz.client ;
import  java.io.File;
import  java.net.URL;
```

Q 05: Explain Java class loaders? If you have a class in a package, what do you need to do to run it? Explain dynamic class loading? **LF**

A 05: Class loaders are hierarchical. Classes are introduced into the JVM as they are referenced by name in a class that is **already running** in the JVM. So, how is the very first class loaded? The very first class is especially loaded with the help of *static main(*) method declared in your class. All the subsequently loaded classes are loaded by the classes, which are already loaded and running. A class loader creates a namespace. All JVMs include at least one class loader that is embedded within the JVM called the primordial (or bootstrap) class loader. Now let's look at non-primordial class loaders. The JVM has hooks in it to allow user defined class loaders to be used in place of primordial class loader. Let us look at the class loaders created by the JVM.

CLASS LOADER	reloadable?	Explanation
Bootstrap (**primordial**)	No	Loads JDK internal classes, *java.** packages. (as defined in the sun.boot.class.path system property, typically loads rt.jar and i18n.jar)
Extensions	No	Loads jar files from JDK extensions directory (as defined in the java.ext.dirs system property – usually lib/ext directory of the JRE)
System	No	Loads classes from system classpath (as defined by the java.class.path property, which is set by the **CLASSPATH** environment variable or –classpath or –cp command line options)



Class loaders are hierarchical and use a **delegation model** when loading a class. Class loaders request their parent to load the class first before attempting to load it themselves. When a class loader loads a class, the child class loaders in the hierarchy will never reload the class again. Hence **uniqueness** is maintained. Classes loaded

by a child class loader have **visibility** into classes loaded by its parents up the hierarchy but the reverse is not true as explained in the above diagram.

Q. What do you need to do to run a class with a main() method in a package?

Example: Say, you have a class named "Pet" in a project folder "c:\myProject" and package named com.xyz.client, will you be able to compile and run it as it is?

```
package com.xyz.client;

public class Pet {
    public static void main(String[] args) {
         System.out.println("I am found in the classpath");
    }
}
```

To run → c:\myProject> `java com.xyz.client.Pet`

The answer is no and you will get the following exception: "Exception in thread "main" java.lang.-NoClassDefFoundError: com/xyz/client/Pet". You need to set the classpath. How can you do that? One of the following ways:

1. Set the operating system **CLASSPATH** environment variable to have the project folder "c:\myProject". [Shown in the above diagram as the System –classpath class loader]
2. Set the operating system **CLASSPATH** environment variable to have a jar file "c:/myProject/client.jar", which has the *Pet.class* file in it. [Shown in the above diagram as the System –classpath class loader].
3. Run it with –cp or –classpath option as shown below:

```
c:\>java -cp  c:/myProject  com.xyz.client.Pet
              OR
c:\>java -classpath c:/myProject/client.jar  com.xyz.client.Pet
```

Important: Two objects loaded by different class loaders are never equal even if they carry the same values, which mean a class is uniquely identified in the context of the associated class loader. This applies to **singletons** too, where **each class loader will have its own singleton**. [Refer **Q51** in Java section for singleton design pattern]

Q. Explain static vs. dynamic class loading?

Static class loading	Dynamic class loading
Classes are statically loaded with Java's "new" operator. `class MyClass {` `  public static void main(String args[]) {` `  Car c = new Car();` `  }` `}`	Dynamic loading is a technique for programmatically invoking the functions of a class loader at run time. Let us look at how to load classes dynamically. **Class.forName** (String *className*); //static method which returns a Class The above static method returns the class object associated with the class name. The string *className* can be supplied dynamically at run time. Unlike the static loading, the dynamic loading will decide whether to load the class *Car* or the class *Jeep at* runtime based on a properties file and/or other runtime conditions. Once the class is dynamically loaded the following method returns an instance of the loaded class. It's just like creating a class object with no arguments. **class.newInstance ()**; //A non-static method, which creates an instance of a //class (i.e. creates an object). Jeep myJeep = null ; //myClassName should be read from a .properties file or a Constants class. **// stay away from hard coding values in your program. CO** String myClassName = "au.com.Jeep" ; Class vehicleClass = **Class.forName**(myClassName) ; myJeep = (Jeep) vehicleClass.**newInstance**(); myJeep.setFuelCapacity(50);
A ***NoClassDefFoundException*** is thrown if a class is referenced with Java's "*new*" operator (i.e. static loading) but the runtime system cannot find the referenced class.	A ***ClassNotFoundException*** is thrown when an application tries to load in a class through its string name using the following methods but no definition for the class with the specified name could be found: ▪ The forName(..) method in class - ***Class***. ▪ The findSystemClass(..) method in class - ***ClassLoader***. ▪ The loadClass(..) method in class - ***ClassLoader***.

Q. What are "static initializers" or "static blocks with no function names"? When a class is loaded, all blocks that are declared static and don't have function name (i.e. static initializers) are executed even before the constructors are executed. As the name suggests they are typically used to initialize static fields. **CO**

```
public class StaticInitializer {
   public static final int A = 5;
   public static final int B; //note that it is not → public static final int B = null;
   //note that since B is final, it can be initialized only once.

   //Static initializer block, which is executed only once when the class is loaded.

   static {
      if(A == 5)
         B = 10;
      else
         B = 5;
   }

   public StaticInitializer(){}  //constructor is called only after static initializer block
}
```

The following code gives an **Output of** A=5, B=10.

```
public class Test {
    System.out.println("A =" + StaticInitializer.A + ", B =" + StaticInitializer.B);
}
```

Q 06: What is the difference between constructors and other regular methods? What happens if you do not provide a constructor? Can you call one constructor from another? How do you call the superclass's constructor? **LF** **FAQ**

A 06:

Constructors	Regular methods
Constructors must have the same name as the class name and cannot return a value. The constructors are called only once per creation of an object while regular methods can be called many times. E.g. for a Pet.class `public Pet() {}  // constructor`	Regular methods can have any name and can be called any number of times. E.g. for a Pet.class. `public void Pet(){}  // regular method has a void return type.` **Note:** method name is shown starting with an uppercase to differentiate a constructor from a regular method. Better naming convention is to have a meaningful name starting with a lowercase like: `public void createPet(){}  // regular method has a void return type`

Q. What happens if you do not provide a constructor? Java does not actually require an explicit constructor in the class description. If you do not include a constructor, the Java compiler will create a default constructor in the byte code with an empty argument. This default constructor is equivalent to the explicit "Pet(){}". If a class includes one or more explicit constructors like "public Pet(int id)" or "Pet(){}" etc, the java compiler does *not* create the default constructor "Pet(){}".

Q. Can you call one constructor from another? Yes, by using **this()** syntax. E.g.

```
public Pet(int id) {
    this.id = id;                       // "this" means this object
}
public Pet (int id, String type) {
    this(id);                           // calls constructor public Pet(int id)
    this.type = type;                  // "this" means this object
}
```

Q. How to call the superclass constructor? If a class called "*SpecialPet"* extends your "*Pet"* class then you can use the keyword "**super**" to invoke the superclass's constructor. E.g.

```
public SpecialPet(int id) {
     super(id);                        //must be the very first statement in the constructor.
}
```

To call a regular method in the super class use: "**super**.myMethod( );". This can be called at any line. Some frameworks based on JUnit add their own initialization code, and not only do they need to remember to invoke

their parent's setup() method, you, as a user, need to remember to invoke theirs after you wrote your initialization code:

```
public class DBUnitTestCase extends TestCase {
  public void setUp() {
    super.setUp();
    // do my own initialization
  }
}
```

```
public void cleanUp() throws Throwable
{
     try {
       … // Do stuff here to clean up your object(s).
     }
     catch (Throwable t) {}
     finally{
         super.cleanUp(); //clean up your parent class. Unlike constructors
                          // super.regularMethod() can be called at any line.
     }
}
```

Q 07: What are the advantages of Object Oriented Programming Languages (OOPL)? DC FAQ

A 07: The Object Oriented Programming Languages directly represent the real life objects like *Car*, *Jeep*, *Account*, *Customer* etc. The features of the OO programming languages like ***polymorphism, inheritance*** *and* ***encapsulation*** make it powerful. [**Tip:** remember **pie** which, stands for **P**olymorphism**, I**nheritance and **E**ncapsulation are the **3 pillars** of OOPL]

Q 08: How does the Object Oriented approach improve software development? DC

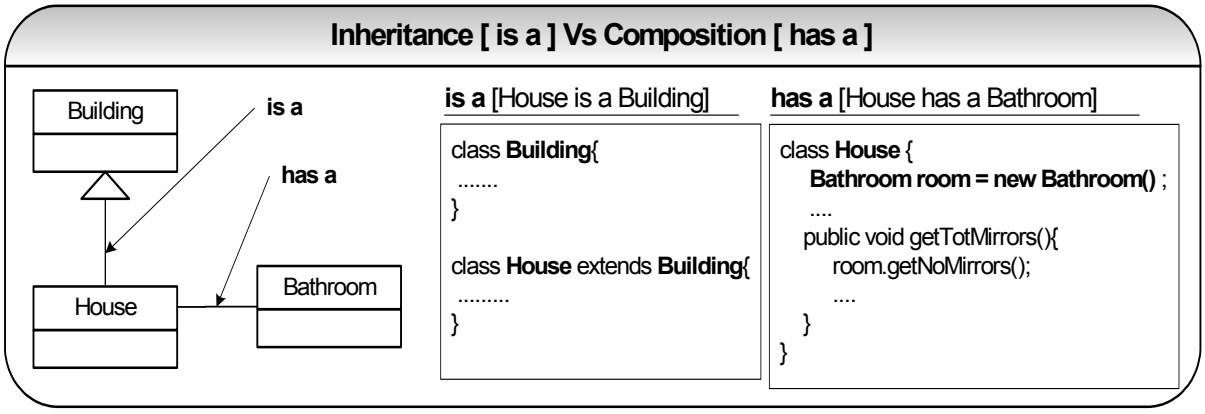
A 08: The key benefits are:

- ***Re-use*** of previous work: using **implementation inheritance** and **object composition**.
- ***Real mapping to the problem domain:*** Objects map to real world and represent vehicles, customers, products etc: with **encapsulation**.
- ***Modular Architecture:*** Objects, systems, frameworks etc are the building blocks of larger systems.

The ***increased quality*** and ***reduced development time*** are the by-products of the key benefits discussed above. If 90% of the new application consists of proven existing components then only the remaining 10% of the code have to be tested from scratch.

Q 09: How do you express an *'**is a**'* relationship and a *'**has a**'* relationship or explain inheritance and composition? What is the difference between composition and aggregation? DC FAQ

A 09: The *'**is a**'* relationship is expressed with **inheritance** and *'**has a**'* relationship is expressed with **composition**. Both inheritance and composition allow you to place sub-objects inside your new class. Two of the main techniques for **code reuse** are **class inheritance** and **object composition.**



Inheritance is uni-directional. For example *House* **is a** *Building*. But *Building* is not a *House*. Inheritance uses **extends** key word. **Composition*:*** is used when *House* **has a** *Bathroom.* It is incorrect to say *House* is a

Bathroom. Composition simply means using instance variables that refer to other objects. The class *House* will have an instance variable, which refers to a *Bathroom* object.

Q. Which one to favor, composition or inheritance? The guide is that inheritance should be only used when *subclass* '**is a**' *superclass*.

- Don't use inheritance just to get code reuse. If there is no '**is a**' relationship then use composition for code reuse. Overuse of **implementation inheritance** (uses the "extends" key word) can break all the subclasses, if the superclass is modified.
- Do not use inheritance just to get polymorphism. If there is no '**is a**' relationship and all you want is **polymorphism** then use **interface inheritance** with **composition,** which gives you **code reuse** (Refer **Q10** in Java section for interface inheritance).

What is the difference between aggregation and composition?

Aggregation	Composition
Aggregation is an association in which one class belongs to a collection. This is a part of a whole relationship where a part can exist without a whole. **For example** a line item is a whole and product is a part. If a line item is deleted then corresponding product need not be deleted. So **aggregation has a weaker relationship**.	Composition is an association in which one class belongs to a collection. This is a part of a whole relationship where a part cannot exist without a whole. If a whole is deleted then all parts are deleted. **For example** An order is a whole and line items are parts. If an order is deleted then all corresponding line items for that order should be deleted. So **composition has a stronger relationship**.

Q 10: What do you mean by polymorphism, inheritance, encapsulation, and dynamic binding? **DC** **SE** **FAQ**

A 10: ***Polymorphism*** – means the ability of a single variable of a given type to be used to reference objects of different types, and automatically call the method that is specific to the type of object the variable references. In a nutshell, polymorphism is a bottom-up method call. The benefit of polymorphism is that it is **very easy to add new classes of derived objects without breaking the calling code** (i.e. getTotArea() in the sample code shown below) that uses the polymorphic classes or interfaces. When you send a message to an object even though you don't know what specific type it is, and the right thing happens, that's called ***polymorphism***. The process used by object-oriented programming languages to implement polymorphism is called ***dynamic binding***. Let us look at some sample code to demonstrate polymorphism: **CO**

Sample code:

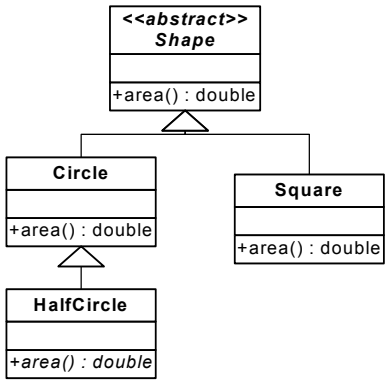
```
//client or calling code
double dim = 5.0; //ie 5 meters radius or width
List listShapes = new ArrayList(20);

Shape s = new Circle();
 listShapes.add(s); //add circle

s = new Square();
listShapes.add(s); //add square

getTotArea (listShapes,dim); //returns 78.5+25.0=103.5

//Later on, if you decide to add a half circle then define
//a HalfCircle  class, which extends Circle and then provide an
//area(). method but your called method getTotArea(...) remains
//same.

s = new HalfCircle();
listShapes.add(s); //add HalfCircle

getTotArea (listShapes,dim); //returns 78.5+25.0+39.25=142.75

/** called method: method which adds up areas of various
**  shapes supplied to it.
**/
public double getTotArea(List listShapes, double dim){
    Iterator it = listShapes.iterator();
    double totalArea = 0.0;
    //loop through different shapes
    while(it.hasNext()) {
        Shape s =  (Shape) it.next();
        totalArea += s.area(dim); //polymorphic method call
    }
    return  totalArea ;
}
```

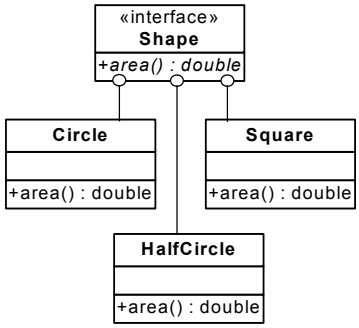
For example: given a base class/interface *Shape*, polymorphism allows the programmer to define different area(double dim1) methods for any number of derived classes such as *Circle*, *Square* etc. No matter what shape an object is, applying the area method to it will return the right results.

Later on *HalfCicle* can be added without breaking your called code i.e. method getTotalArea(...)

Depending on what the shape is, appropriate area(double dim) method gets called and calculated.

Circle → area is 78.5sqm
Square → area is 25sqm
HalfCircle → area is 39.25 sqm





Inheritance – is the inclusion of behavior (i.e. methods) and state (i.e. variables) of a base class in a derived class so that they are accessible in that derived class. The key benefit of Inheritance is that it provides the formal mechanism for **code reuse**. Any shared piece of business logic can be moved from the derived class into the base class as part of refactoring process to improve maintainability of your code by avoiding code duplication. The existing class is called the *superclass* and the derived class is called the *subclass*. **Inheritance** can also be defined as the process whereby one object acquires characteristics from one or more other objects the same way children acquire characteristics from their parents. **There are two types of inheritances:**

1. **Implementation inheritance** (aka class inheritance): You can extend an application's functionality by reusing functionality in the parent class by inheriting all or some of the operations already implemented. In Java, you can only inherit from one superclass. Implementation inheritance promotes reusability but improper use of class inheritance can cause programming nightmares by breaking encapsulation and making future changes a problem. With implementation inheritance, the subclass becomes tightly coupled with the superclass. This will make the design fragile because if you want to change the superclass, you must know all the details of the subclasses to avoid breaking them. So when using implementation inheritance, **make sure that the subclasses depend only on the behavior of the superclass, not on the actual implementation**. For example in the above diagram, the subclasses should only be concerned about the behavior known as area() but not how it is implemented.

2. **Interface inheritance** (aka type inheritance): This is also known as subtyping. Interfaces provide a mechanism for specifying a relationship between otherwise unrelated classes, typically by specifying a set of common methods each implementing class must contain. Interface inheritance promotes the design concept of **program to interfaces not to implementations**. This also reduces the coupling or implementation dependencies between systems. In Java, you can implement any number of interfaces. This is more flexible than implementation inheritance because it won't lock you into specific implementations which make subclasses difficult to maintain. So care should be taken not to break the implementing classes by modifying the interfaces.

Which one to use? Prefer interface inheritance to implementation inheritance because it promotes the design concept of **coding to an interface** and **reduces coupling**. Interface inheritance can achieve **code reuse** with the help of **object composition**. If you look at Gang of Four (GoF) design patterns, you can see that it favors interface inheritance to implementation inheritance. **CO**

Implementation inheritance

Let's assume that savings account and term deposit account have a similar behavior in terms of depositing and withdrawing money, so we will get the super class to implement this behavior and get the subclasses to reuse this behavior. But saving account and term deposit account have specific behavior in calculating the interest.

Super class *Account* has **reusable code** as methods **deposit** (double amount) and **withdraw** (double amount).

```
public abstract class Account  {
   public void deposit (double amount) {
        System.out.println("depositing " + amount);
   }

   public void withdraw (double amount) {
        System.out.println ("withdrawing " + amount);
   }

   public abstract double calculateInterest(double  amount);
}
```

```
public class SavingsAccount extends Account {

    public double calculateInterest (double amount) {
       // calculate interest for SavingsAccount
       return amount * 0.03;
    }

    public void deposit (double amount) {
       super.deposit (amount); // get code reuse
       // do something else
    }

    public void withdraw (double amount) {
```

Interface inheritance with composition

Let's look at an **interface inheritance** code sample, which makes use of **composition** for reusability. In the following example the methods deposit(…) and withdraw(…) share the same piece of code in *AccountHelper* class. The method calculateInterest(…) has its specific implementation in its own class.

```
public interface Account {
    public abstract double calculateInterest(double amount);
    public abstract void deposit(double amount);
    public abstract void withdraw(double amount);
}
```

Code to interface so that the implementation can change.

```
public interface AccountHelper {
    public abstract void deposit (double amount);
    public abstract void withdraw (double amount);
}
```

class **AccountHelperImpl** has **reusable code** as methods **deposit** (double amount) and **withdraw** (double amount).

```
public class AccountHelperImpl implements AccountHelper {
   public void deposit(double amount) {
        System.out.println("depositing " + amount);
   }

   public void withdraw(double amount) {
        System.out.println("withdrawing " + amount);
   }

}
```

```
public class SavingsAccountImpl implements Account {
```

```
        super.withdraw (amount); // get code reuse
        // do something else
    }
}
```

```
public class TermDepositAccount extends Account {

    public double calculateInterest (double amount) {
        // calculate interest for SavingsAccount
        return amount * 0.05;
    }

    public void deposit(double amount) {
        super.deposit (amount); // get code reuse
        // do something else
    }

    public void withdraw(double amount) {
        super.withdraw (amount); // get code reuse
        // do something else
    }
}
```

```
    // composed helper class (i.e. composition).
    AccountHelper helper = new AccountHelperImpl ();

    public double calculateInterest (double amount) {
        // calculate interest for SavingsAccount
        return amount * 0.03;
    }

    public void deposit (double amount) {
        helper.deposit( amount); // code reuse via composition
    }

    public void withdraw (double amount) {
        helper.withdraw (amount); // code reuse via composition
    }
}
```

```
public class TermDepositAccountImpl implements Account {

    // composed helper class (i.e. composition).
    AccountHelper helper = new AccountHelperImpl  ();

    public double calculateInterest (double amount) {
        //calculate interest for SavingsAccount
        return amount * 0.05;
    }

    public void deposit (double amount) {
         helper.deposit (amount) ;  // code reuse via composition
    }

    public void withdraw (double amount) {
          helper.withdraw (amount) ;  // code reuse via composition
    }

}
```

The Test class:

```
public class Test {
    public static void main(String[] args) {
        Account acc1 = new SavingsAccountImpl();
        acc1.deposit(50.0);

        Account acc2 = new TermDepositAccountImpl();
        acc2.deposit(25.0);

        acc1.withdraw(25);
        acc2.withdraw(10);

        double cal1 = acc1.calculateInterest(100.0);
        double cal2 = acc2.calculateInterest(100.0);

        System.out.println("Savings --> " + cal1);
        System.out.println("TermDeposit -->  " + cal2);
     }
}
```

The output:

```
depositing 50.0
depositing 25.0
withdrawing 25.0
withdrawing 10.0
Savings --> 3.0
TermDeposit -->  5.0
```

Q. Why would you prefer code reuse via composition over inheritance? Both the approaches make use of polymorphism and gives code reuse (in different ways) to achieve the same results but:

- The advantage of class inheritance is that it is done statically at compile-time and is easy to use. The disadvantage of class inheritance is that because it is static, implementation inherited from a parent class cannot be changed at run-

time. In object composition, functionality is acquired dynamically at run-time by objects collecting references to other objects. The advantage of this approach is that implementations can be replaced at run-time. This is possible because objects are accessed only through their interfaces, so one object can be replaced with another just as long as they have the same type. **For example:** the composed class ***AccountHelperImpl*** can be replaced by another more efficient implementation as shown below if required:

```
public class EfficientAccountHelperImpl implements AccountHelper {
    public void deposit(double amount) {
        System.out.println("efficient depositing " + amount);
    }

    public void withdraw(double amount) {
        System.out.println("efficient withdrawing " + amount);
    }
}
```

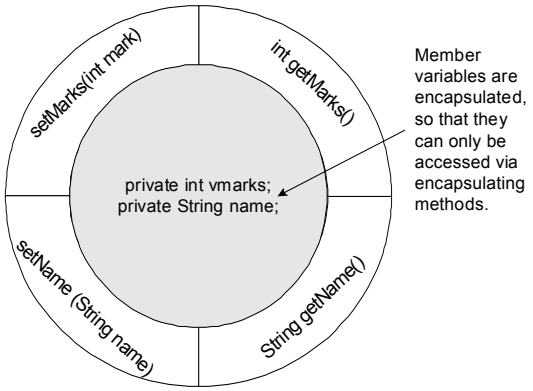
- Another problem with class inheritance is that the subclass becomes dependent on the parent class implementation. This makes it harder to reuse the subclass, especially if part of the inherited implementation is no longer desirable and hence can break encapsulation. Also a change to a superclass can not only ripple down the inheritance hierarchy to subclasses, but can also ripple out to code that uses just the subclasses making the design fragile by tightly coupling the subclasses with the super class. But it is easier to change the interface/implementation of the composed class.

Due to the flexibility and power of object composition, **most design patterns emphasize object composition over inheritance whenever it is possible**. Many times, a design pattern shows a clever way of solving a common problem through the use of object composition rather then a standard, less flexible, inheritance based solution.

Encapsulation – refers to keeping all the related members (variables and methods) together in an object. Specifying member variables as private can hide the variables and methods. Objects should hide their inner workings from the outside view. Good **encapsulation improves code modularity by preventing objects interacting with each other in an unexpected way**, which in turn makes future development and refactoring efforts easy. **CO**

Sample code

```
Class MyMarks {
  private int vmarks =  0;
  private String name;

  public void setMarks(int mark)
                    throws MarkException {
     if(mark > 0)
        this.vmarks =  mark;
     else {
        throw new MarkException("No negative
                                  Values");
     }
  }

   public int getMarks(){
       return vmarks;
   }
   //getters and setters for attribute name goes here.
}
```



Being able to encapsulate members of a class is important for **security** and **integrity**. We can protect variables from unacceptable values. The sample code above describes how encapsulation can be used to protect the *MyMarks* object from having negative values. Any modification to member variable "*vmarks*" can only be carried out through the setter method *setMarks(int mark)*. This prevents the object "*MyMarks*" from having any negative values by throwing an exception.

Q 11: What is design by contract? Explain the *assertion* construct? **DC**

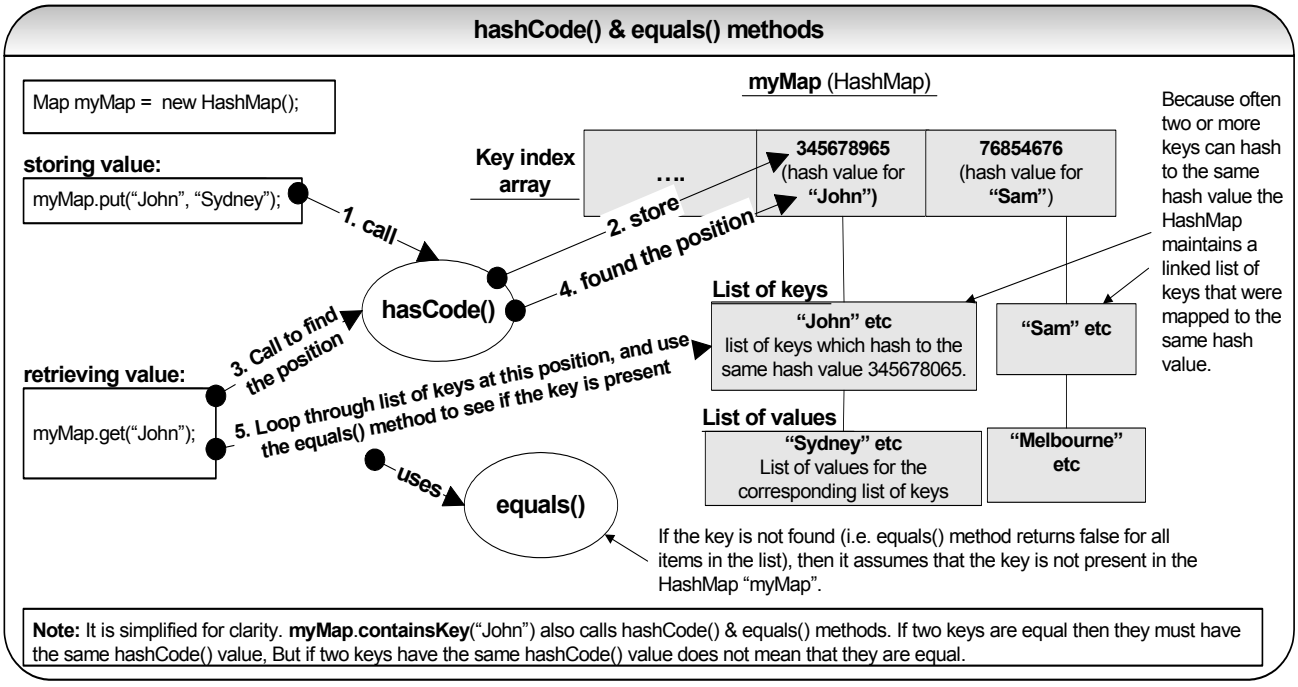
A 11: Design by contract specifies the obligations of a calling-method and called-method to each other. Design by contract is a valuable technique, which should be used to build well-defined interfaces. The strength of this programming methodology is that it gets the programmer to **think clearly about what a function does**, what pre and post conditions it must adhere to and also it **provides documentation for the caller**. Java uses the ***assert*** statement to implement pre- and post-conditions. Java's exceptions handling also support design by contract especially **checked exceptions** (Refer **Q39** in Java section for checked exceptions). In design by contract in addition to specifying programming code to carrying out intended operations of a method the programmer also specifies:

method with public access modifier	followed by an "@" sign and then unsigned hexadecimal representation of the hashcode, for example Pet@162b91. This hexadecimal representation is not what the users of your class want to see. Providing your toString() method makes your class much more pleasant to use and it is recommended that all subclasses override this method. The toString() method is invoked automatically when your object is passed to println(), assert() or the string concatenation operator (+). `public class Pet {` `  int id;` `  String name;` `  public boolean equals(Object obj){` `  //as shown above.` `  }` `  public int hashCode() {` `  //as shown before` `  }` `  public String toString() {` `  StringBuffer sb = new StringBuffer();` `  sb.append("id=").append(id);` `  sb.append(",name=").append(name);` `  return sb.toString();` `  }` `}`
clone() method with protected access modifier	You should override the clone() method very judiciously. Implementing a properly functioning clone method is complex and it is rarely necessary. You are better off providing some alternative means of object copying (refer **Q26** in Java section) or simply not providing the capability. A better approach is to provide a copy constructor or a static factory method in place of a constructor. //constructor `public Pet(Pet petToCopy){` `  …` `}` //static factory method `public static Pet newInstance(Pet petToCopy){` `  …` `}` The clone() method can be disabled as follows: `public final Object clone() throws CloneNotSupportedException {` `  throw new CloneNotSupportedException();` `}`
finalize() method with protected access modifier	Unlike C++ destructors, the finalize() method in Java is unpredictable, often dangerous and generally unnecessary. Use try{} finally{} blocks as discussed in **Q32** in Java section & **Q45** in Enterprise section. The finalize() method should only be used in rare instances as a safety net or to terminate non-critical native resources. If you do happen to call the finalize() method in some rare instances then remember to call the super.finalize() as shown below: `protected void finalize() throws Throwable {` `  try{` `  //finalize subclass state` `  }` `  finally {` `  super.finalize();` `  }` `}`

Q 20: When providing a user defined key class for storing objects in the HashMaps or Hashtables, what methods do you have to provide or override (i.e. **method overriding**)? LF PI CO FAQ

A 20: You should override the **equals()** and **hashCode()** methods from the *Object* class. The default implementation of the equals() and hashcode(), which are inherited from the java.lang.Object uses an object instance's memory location (e.g. MyObject@6c60f2ea). This can cause problems when two instances of the car objects have the same color but the inherited equals() will return false because it uses the memory location, which is different for

the two instances. Also the **toString()** method can be overridden to provide a proper string representation of your object.



Q. What are the primary considerations when implementing a user defined key?

- If a class overrides **equals()**, it must override **hashCode()**.
- If 2 objects are equal, then their hashCode values must be equal as well.
- If a field is not used in equals(), then it must not be used in hashCode().
- If it is accessed often, hashCode() is a candidate for caching to enhance performance.
- It is a best practice to implement the user defined key class as an immutable (refer **Q21**) object.

Q. Why it is a best practice to implement the user defined key class as an immutable object?

Problem: As per the code snippet shown below if you use a mutable user defined class "**UserKey**" as a HashMap key and subsequently if you mutate (i.e. modify via setter method e.g. `key.setName("Sam")`) the key after the object has been added to the HashMap then you will not be able to access the object later on. The original key object will still be in the HashMap (i.e. you can iterate through your HashMap and print it – both prints as "Sam" as opposed to "John" & Sam) but you cannot access it with `map.get(key)` or querying it with `map.containsKey(key)` will return false because the key "John" becomes "Sam" in the "List of keys" at the key index "345678965" if you mutate the key after adding. These types of errors are very hard to trace and fix.

```
Map myMap = new HashMap(10);
//add the key "John"
UserKey key = new UserKey("John");  //Assume UserKey class is mutable
myMap.put(key, "Sydney");
//now to add the key "Sam"
key.setName("Sam"); // same key object is mutated instead of creating a new instance.
                    // This line modifies the key value "John" to "Sam" in the "List of keys"
                    // as shown in the diagram above. This means that the key "John" cannot be
                    // accessed. There will be two keys with "Sam" in positions with hash
                    // values 345678965 and 76854676.
myMap.put(key, "Melbourne");

myMap.get(new UserKey("John")); // key cannot be accessed. The key hashes to the same position
             // 345678965 in the "Key index array" but cannot be found in the "List of keys"
```

Solution: Generally you use a java.lang.Integer or a java.lang.String class as the key, which are immutable Java objects. If you define your own key class then it is a best practice to make the key class an immutable object (i.e. do not provide any setXXX() methods in your key class. e.g. no `setName(…)` method in the ***UserKey*** class). If a programmer wants to insert a new key then he/she will always have to instantiate a new object (i.e. cannot mutate the existing key because immutable key object class has no setter methods).

performance benefit in explicitly declaring your serialVersionUID (because does not have to be calculated). So, it is best practice to add your own **serialVersionUID** to your Serializable classes as soon as you create them as shown below:

```
public class Car {
     static final long serialVersionUID = 1L; //assign a long value
}
```

Note: Alternatively you can use the serialver tool comes with Sun's JDK. This tool takes a full class name on the command line and returns the serialVersionUID for that compiled class. **For example:**

```
static final long serialVersionUID = 10275439472837494L; //generated by serialver tool.
```

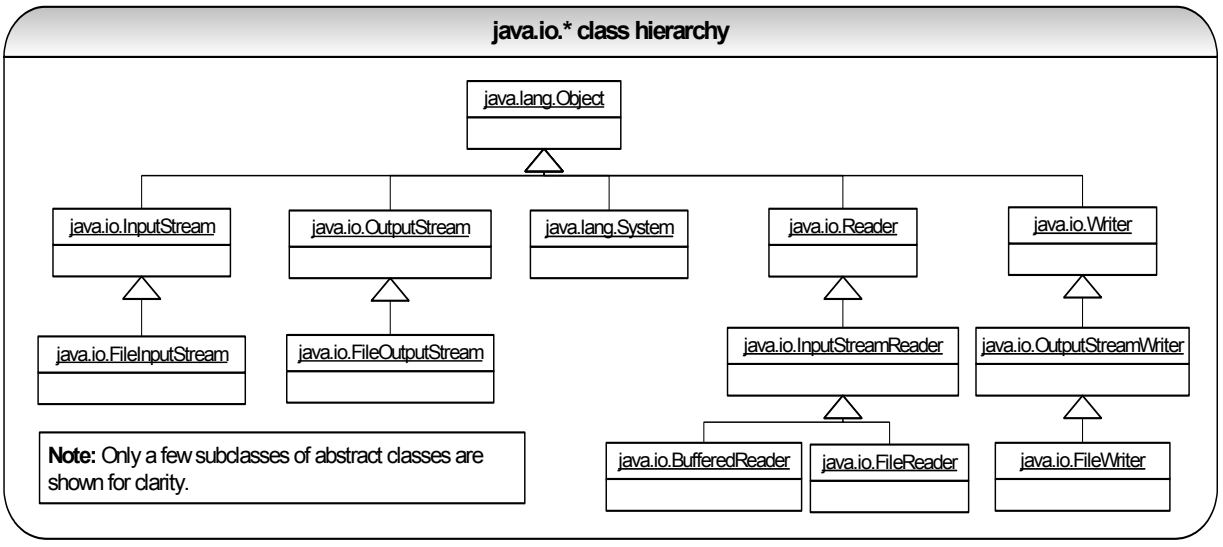
Q 24: Explain the Java I/O streaming concept and the use of the decorator design pattern in Java I/O? **LF** **DP** **PI** **SI**

A 24: Java input and output is defined in terms of an abstract concept called a "**stream**", which is a sequence of data. There are 2 kinds of streams.

- Byte streams (8 bit bytes) → Abstract classes are: **InputStream** and **OutputStream**
- Character streams (16 bit UNICODE) → Abstract classes are: **Reader** and **Writer**

Design pattern: *java.io.** classes use the **decorator design pattern**. The decorator design pattern **attaches responsibilities to objects at runtime**. Decorators are more flexible than inheritance because the **inheritance attaches responsibility to classes at compile time**. The *java.io.** classes use the decorator pattern to construct different combinations of behavior at runtime based on some basic classes.

Attaching responsibilities to classes at compile time using subclassing.	Attaching responsibilities to objects at runtime using a decorator design pattern.
Inheritance (aka subclassing) attaches responsibilities to classes at compile time. When you extend a class, each individual changes you make to child class will affect all instances of the child classes. Defining many classes using inheritance to have all possible combinations is problematic and inflexible.	By attaching responsibilities to **objects at runtime**, you can apply changes to each individual object you want to change. `File file = new File("c:/temp");` `FileInputStream fis = new FileInputStream(file);` `BufferedInputStream bis = new BufferedInputStream(fis);` Decorators decorate an object by enhancing or restricting functionality of an object it decorates. The decorators add or restrict functionality to decorated objects either before or after forwarding the request. At runtime the BufferedInputStream (bis), which is a **decorator** (aka a **wrapper** around decorated object), forwards the method call to its **decorated** object FileInputStream (fis). The "bis" will apply the additional functionality of buffering around the lower level file (i.e. fis) I/O.



Q. How does the new I/O (NIO) offer better scalability and better performance?

It is recommended to use logging frameworks like **Log4J** with **SLF4J** (Simple Logging Façade for Java), which uses buffering instead of using default behavior of **System.out.println(…..)** for better performance. Frameworks like Log4J are configurable, flexible, extensible and easy to use.

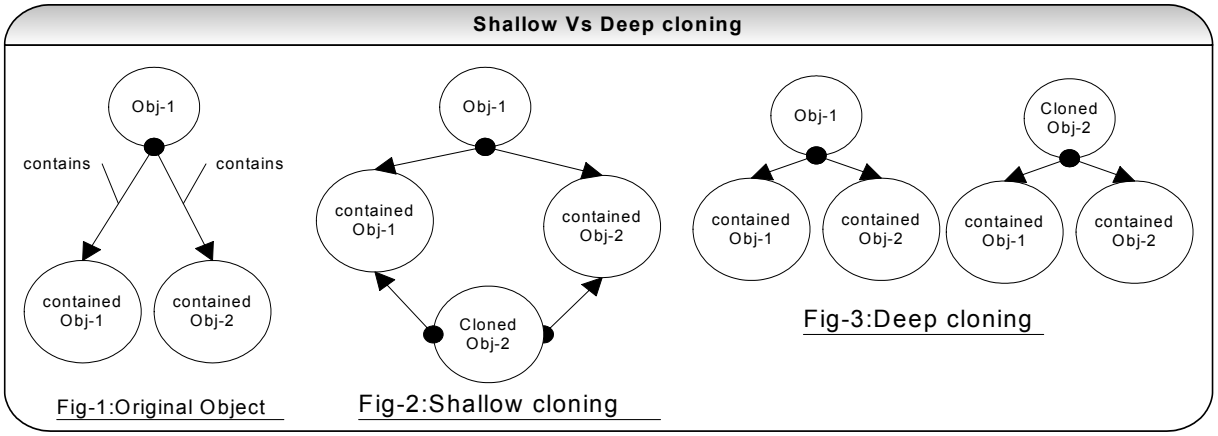
- Use the NIO package, if you are using JDK 1.4 or later, which uses performance-enhancing features like buffers to hold data, memory mapping of files, non-blocking I/O operations etc.
- I/O performance can be improved by minimizing the calls to the underlying operating systems. The Java runtime itself cannot know the length of a file, querying the file system for isDirectory(), isFile(), exists() etc must query the underlying operating system.
- Where applicable caching can be used to improve performance by reading in all the lines of a file into a Java *Collection* class like an ArrayList or a HashMap and subsequently access the data from an in-memory collection instead of the disk.

Q 26: What is the main difference between shallow cloning and deep cloning of objects? DC LF MI PI

A 26: The default behavior of an object's clone() method automatically yields a shallow copy. So to achieve a deep copy the classes must be edited or adjusted.

Shallow copy: If a shallow copy is performed on obj-1 as shown in fig-2 then it is copied but its contained objects are not. The contained objects Obj-1 and Obj-2 are affected by changes to cloned Obj-2. Java supports shallow cloning of objects by default when a class implements the *java.lang.Cloneable* interface.

Deep copy: If a deep copy is performed on obj-1 as shown in fig-3 then not only obj-1 has been copied but the objects contained within it have been copied as well. Serialization can be used to achieve deep cloning. Deep cloning through serialization is faster to develop and easier to maintain but carries a performance overhead.

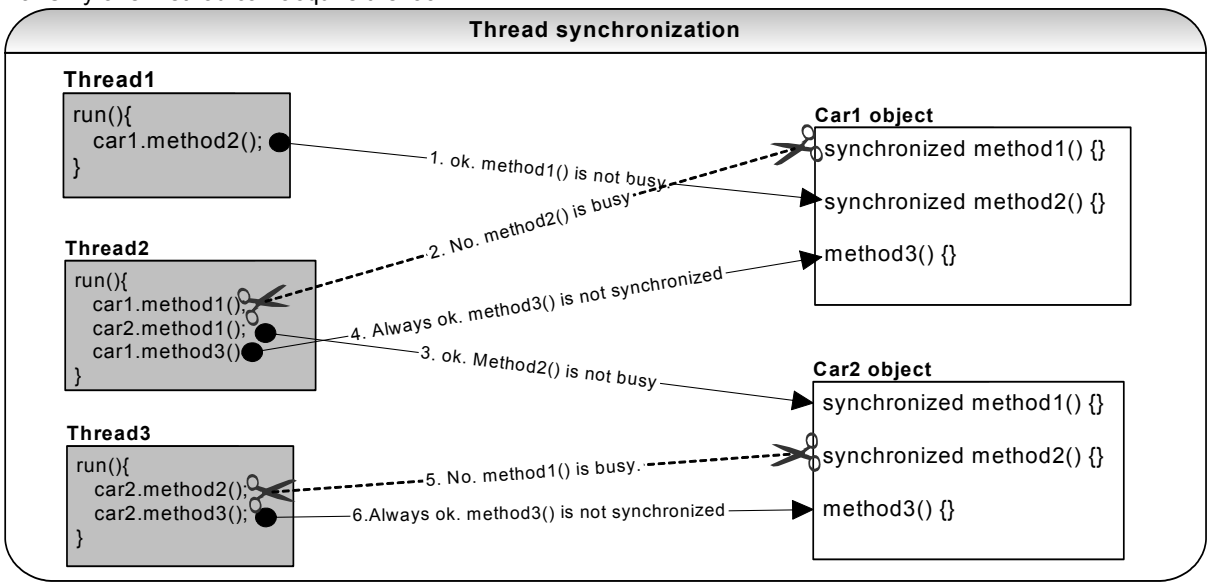


For example invoking clone() method on a collection like *HashMap, List etc* returns a shallow copy of *HashMap, List,* instances. This means if you clone a *HashMap,* the map instance is cloned but *the* **keys and values themselves are not cloned**. If you want a deep copy then a simple method is to serialize the *HashMap* to a *ByteArrayOutputSream* and then deserialize it. This creates a deep copy but does require that all keys and values in the *HashMap* are Serializable. Main advantage of this approach is that it will deep copy any arbitrary object graph. Refer **Q23** in Java section for deep copying using Serialization. Alternatively you can provide a static factory method to deep copy. **Example:** to deep copy a list of *Car* objects.

```
public static List deepCopy(List listCars) {
     List copiedList = new ArrayList(10);
     for (Object object : listCars) {   //JDK 1.5 for each loop
        Car original = (Car)object;
        Car carCopied = new Car();  //instantiate a new Car object
        carCopied.setColor((original.getColor()));
        copiedList.add(carCopied);
    }
    return copiedList;
}
```

Q 49: If 2 different threads hit 2 different synchronized methods in an object at the same time will they both continue? LF

A 49: No. Only one method can acquire the lock.



Note: If your job requires deeper understanding of threads then please refer to the following articles by Allen Holub at http://www.javaworld.com. There are number of parts (part 1 – Part - 8) to the article entitled "**Programming Java threads in the real world**". URLs for some of the parts are: http://www.javaworld.com/javaworld/jw-09-1998/jw-09-threads.html, http://www.javaworld.com/javaworld/jw-10-1998/jw-10-toolbox.html, etc.

Q 50: Explain threads blocking on I/O? LF

A 50: Occasionally threads have to block on conditions other than object locks. I/O is the best example of this. Threads block on I/O (i.e. enters the waiting state) so that other threads may execute while the I/O operation is performed. When threads are blocked (say due to time consuming reads or writes) on an I/O call inside an object's synchronized method and also if the other methods of the object are also synchronized then the object is essentially frozen while the thread is blocked.

Be sure to not synchronize code that makes blocking calls, or make sure that a non-synchronized method exists on an object with synchronized blocking code. Although this technique requires some care to ensure that the resulting code is still thread safe, it allows objects to be responsive to other threads when a thread holding its locks is blocked.

Note: The **java.nio.*** package was introduced in JDK1.4. The coolest addition is non-blocking I/O (aka NIO that stands for New I/O). Refer **Q24** in Java section for NIO.

Note: Q51 & **Q52** in Java section are very popular questions on design patterns.

Q 51: What is a **singleton** pattern? How do you code it in Java? DP MI CO FAQ

A 51: A singleton is a class that can be instantiated **only one time in a JVM per class loader**. Repeated calls always return the same instance. Ensures that a class has only one instance, and provide a **global point of access**. It can be an issue if singleton class gets loaded by multiple class loaders or JVMs.

```
public class OnlyOne {

    private static OnlyOne one = new OnlyOne();

    // private constructor. This class cannot be instantiated from outside and
    // prevents subclassing.
    private OnlyOne(){}

    public static OnlyOne getInstance() {
             return one;
    }
}
```

Java – Performance and Memory issues

Q. Give me an instance where you made a significant contribution in improving performance ?

There is a good chance that the position you are being interviewed for require someone with skills to identify performance and/or memory issues and ability to optimize performance and solve memory issues. If you happen to be in an interview with an organization facing serious issues with regards to their Java application relating to memory leaks, performance problems or a crashing JVM etc then you are likely to be asked questions on these topics. You will find more questions and answers relating to these key areas (i.e. performance and memory issues) in the **Enterprise Java** section and "**How would you go about…"** sections. You could also demonstrate your skills in these key areas by reflecting back on your past experiences as discussed in **Q82** in Java section. Even though **Q82** is a **situational** or **behavioral** question, you can streamline your answer to demonstrate your technical strengths relating to these key areas as well as your behavioral ability to cope with stress.

Q 72: How would you improve performance of a Java application? **PI** **BP** **FAQ**

A 72:

- **Pool valuable system resources** like threads, database connections, socket connections etc. Emphasize on reuse of threads from a pool of threads. Creating new threads and discarding them after use can adversely affect performance. Also consider using multi-threading in your single-threaded applications where possible to enhance performance. Optimize the pool sizes based on system and application specifications and requirements. Having too many threads in a pool also **can result in performance and scalability problems due to consumption of memory stacks** (i.e. each thread has its own stack. Refer **Q34**, **Q42** in Java section) and **CPU context switching** (i.e. switching between threads as opposed to doing real computation.).

- **Minimize network overheads** by retrieving several related items simultaneously in one remote invocation if possible. Remote method invocations involve a network round-trip, marshaling and unmarshaling of parameters, which can cause huge performance problems if the remote interface is poorly designed. (Refer **Q125** in Enterprise section).

  Most applications need to retrieve data from and save/update data into one or more databases. Database calls are remote calls over the network. In general data should be **lazily loaded** (i.e. load only when required as opposed to pre-loading from the database with a view that it can be used later) from a database to conserve memory but there are use cases (i.e. need to make several database calls) where **eagerly loading** data and caching can improve performance by minimizing network trips to the database. Data can be eagerly loaded with a help of SQL scripts with complex joins or stored procedures and cached using third party frameworks or building your own framework. At this point your interviewer could intercept you and ask you some pertinent questions relating to caching like:

  Q: How would you refresh your cache?
  A: You could say that one of the two following strategies can be used:

  1. **Timed cache** strategy where the cache can be replenished periodically (i.e. every 30 minutes, every hour etc). This is a simple strategy applicable when it is acceptable to show dirty data at times and also the data in the database does not change very frequently.

  2. **Dirty check** strategy where <u>your application is the only one which can mutate</u> (i.e. modify) the data in the database. You can set a "isDirty" flag to true when the data is modified in the database through your application and consequently your cache can be refreshed based on the "isDirty" flag.

  Q: How would you refresh your cache if your database is shared by more than one application?
  A: You could use one of the following strategies:

  1. **Database triggers**: You could use database triggers to communicate between applications sharing the same database and write pollers which polls the database periodically to determine when the cache should be refreshed. (Refer **Q102** in Enterprise section)

  2. **XML messaging** (Refer **Enterprise – JMS** subsection in Enterprise section) to communicate between other applications sharing the same database or separate databases to determine when the cache should be refreshed.

behavioral technique is used to evaluate a candidate's future success from past behaviors. The answers to these questions must describe in detail a particular situation like an event, a project or an experience and how you acted on that situation and what the results were. Prepare your answers prior to the interview using the "**S**ituation **A**ction **R**esult (SAR)" approach and avoid fabricating or memorizing your answers. You should try to relate back to your past experiences at your previous employments, community events, sporting events etc. Sample questions and answers are shown below:

Q 82: Give me an example of a time when you set a goal and were able to achieve it? Give me an example of a time you showed initiatiative and took the lead? Tell me about a difficult decision you made in the last year? Give me an example of a time you motivated others? Tell me about a most complex project you were involved in? **FAQ**

A 82:

Situation: When you were working for the ZCC Software Technology Corporation, the overnight batch process called the "Data Pacakager" was developed for a large fast food chain which has over 100 stores. This overnight batch process is responsible for performing a very database intensive search and compute changes like cost of ingredients, selling price, new menu item etc made in various retail stores and package those changes into XML files and send those XML data to the respective stores where they get uploaded into their point of sale registers to reflect the changes. This batch process had been used for the past two years, but since then the number of stores had increased and so did the size of the data in the database. The batch process, which used to take 6-8 hours to complete, had increased to 14-16 hours, which obviously started to adversely affect the daily operations of these stores. The management assigned you with the task of improving the performance of the batch process to 5-6 hours (i.e. suppose to be an overnight process).

Action: After having **analyzed the existing design** and code for the "Data Packager", **you had to take the difficult decision** to let the management know that this batch process needed to be re-designed and re-written as opposed to modifying the existing code, since it was poorly designed. It is hard to extend, maintain (i.e. making a change in one place can break the code some where else and so on) and had no object reuse through caching (makes too many unnecessary network trips to the database) etc. The management was not too impressed with this approach and concerned about the time required to rewrite this batch process since the management had promised the retail stores to provide a solution within 8-12 weeks. **You took the initiative and used your persuasive skills** to **convince the management** that you would be able to provide a re-designed and re-written solution within the 8-12 weeks with the assistance of 2-3 additional developers and two testers. You were entrusted with the task to rewrite the batch process and **you set your goal to complete the task in 8 weeks**. You decided to build the software iteratively by building individual vertical slices as opposed to the big bang waterfall approach [Refer subsection "**Enterprise – Software development process**" in Enterprise – Java section]. You redesigned and wrote the code for a typical use case from end to end (i.e. full vertical slice) within 2 weeks and subsequently carried out functional and integration testing to iron out any unforeseen errors or issues. Once the first iteration is stable, **you effectively communicated** the architecture to the management and to your fellow developers. **Motivated** and **mentored** your fellow developers to build the other iterations, based on the first iteration. At the end of iteration, it was tested by the testers, while the developers moved on to the next iteration.

Results: After having **enthusiastically** worked to your plan with **hard work**, **dedication** and **teamwork**, you were able to have the 90% of the functionality completed in 9 weeks and spent the next 3 weeks fixing bugs, tuning performance and coding rest of the functionality. The fully functional data packager was completed in 12 weeks and took only 3-4 hours to package XML data for all the stores. The team was under pressure at times but you made them believe that it is more of a **challenge as opposed to think of it as a stressful situation**. The newly designed data packager was also easier to maintain and extend. The management was impressed with the outcome and rewarded the team with an outstanding achievement award. The performance of the newly developed data packager was further improved by 20% by tuning the database (i.e. partitioning the tables, indexing etc).

Q 83: Describe a time when you were faced with a stressful situation that demonstrated your coping skills? Give me an example of a time when you used your fact finding skills to solve a problem? Describe a time when you applied your analytical and/or problem solving skills? **FAQ**

A 83:

Situation: When you were working for the Surething insurance corporation pty ltd, you were responsible for the migration of an online insurance application (i.e. external website) to a newer version of application server (i.e. the current version is no longer supported by the vendor). The migration happened smoothly and after a couple of days of going live, you started to experience "OutOfMemoryError", which forced you to restart the application server every day. This raised a red alert and the immediate and the senior management were very concerned and consequently constantly calling for meetings and updates on the progress of identifying the root cause of this issue. This has created a stressful situation.

SECTION TWO

Enterprise Java – Interview questions & answers

KEY AREAS

- Specification Fundamentals SF
- Design Concepts DC
- Design Patterns DP
- Concurrency Issues CI
- Performance Issues PI
- Memory Issues MI
- Exception Handling EH
- Transactional Issues TI
- Security SE
- Scalability Issues SI
- Best Practices BP
- Coding[1] CO

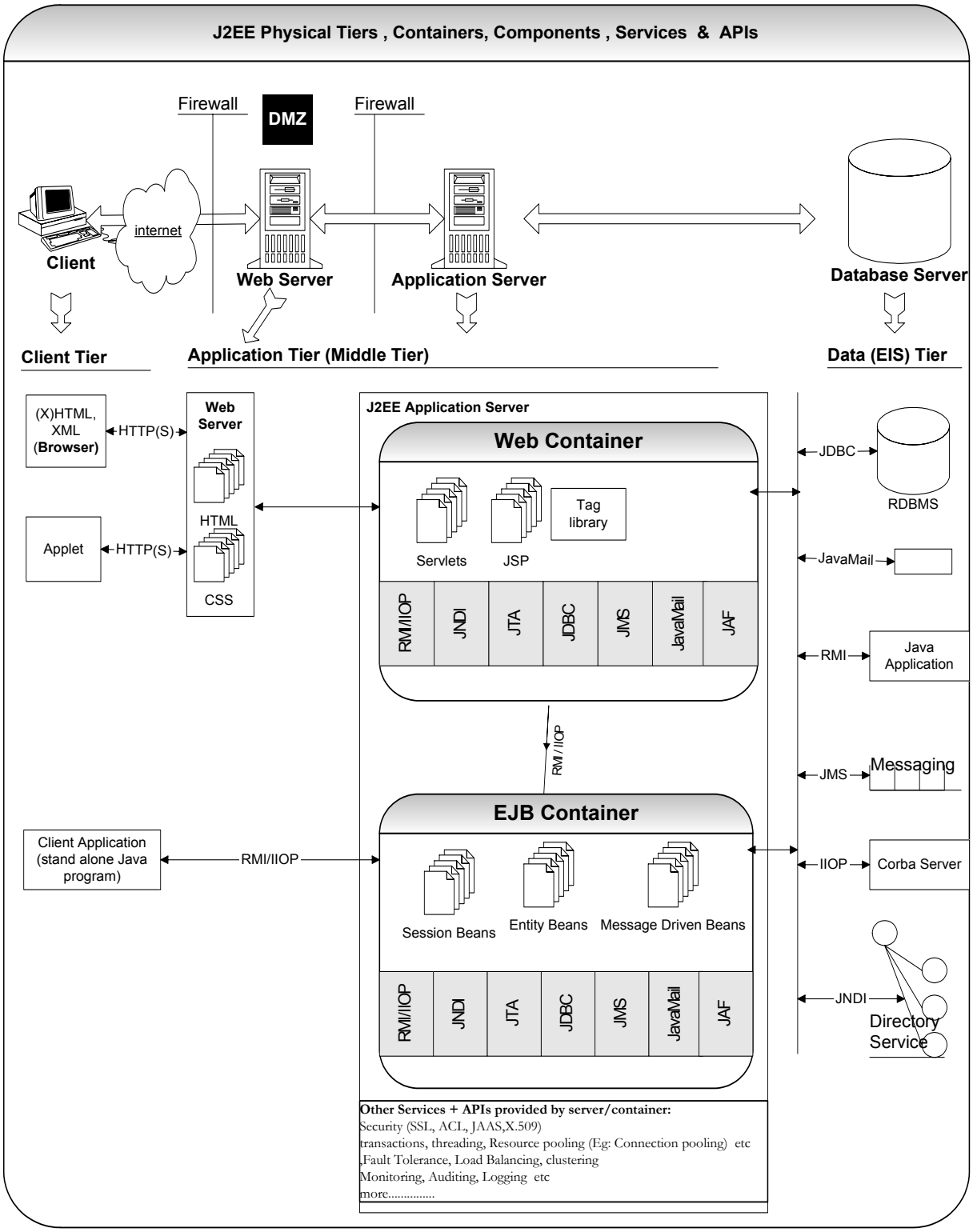
FAQ - Frequently **A**sked **Q**uestions

[1] Unlike other key areas, the CO is not always shown against the question but shown above the actual content of relevance within a question.

Enterprise - J2EE Overview

Q 01: What is J2EE? What are J2EE components and services? SF FAQ

A 01: J2EE (**Java 2 E**nterprise **E**dition) is an environment for developing and deploying enterprise applications. The J2EE platform consists of J2EE components, services, Application Programming Interfaces (APIs) and protocols that provide the functionality for developing multi-tiered and distributed Web based applications.



A **J2EE component** is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and communicates with other components. The J2EE specification defines the following J2EE components:

Component type	Components	Packaged as
Applet	applets	JAR (**J**ava **AR**chive)
Application client	Client side Java codes.	JAR (**J**ava **AR**chive)
Web component	JSP, Servlet	WAR (**W**eb **AR**chive)
Enterprise JavaBeans	Session beans, Entity beans, Message driven beans	JAR (EJB Archive)
Enterprise application	WAR, JAR, etc	EAR (**E**nterprise **AR**chive)
Resource adapters	Resource adapters	RAR (**R**esource Adapter **AR**chive)

Q. So what is the difference between a component and a service?

A component is an application level software unit as shown in the table above. All the J2EE components depend on the container for the system level support like transactions, security, pooling, life cycle management, threading etc. A service is a component that can be used remotely through a remote interface either synchronously or asynchronously (e.g. Web service, messaging system, sockets, RPC etc). A service is a step up from "distributed objects". A service is a function that has a clearly defined service contract (e.g. interface, XML contract) to their consumers or clients, self contained and does not depend on the context or state of other services.

Q. What is a Service Oriented Architecture (SOA)?

SOA is an evolution of the fundamentals governing a component based development. Component based development provides an opportunity for greater code reuse than what is possible with **O**bject **O**riented (**OO**) development. SOA provides even greater code reuse by utilizing OO development, component based development and also by identifying and organizing right services into a hierarchy of composite services. SOA results in loosely coupled application components, in which code is not necessarily tied to a particular database. SOAs are very popular and there is a huge demand exists for development and implementation of SOAs. Refer **Q14** in **How would you go about…?** section for a more detailed discussion on **SOA** and **Web services**.

Q. What are Web and EJB containers?

Containers (Web & EJB containers) are the interface between a J2EE component and the low level platform specific functionality that supports J2EE components. Before a Web, enterprise bean (EJB), or application client component can be executed, it must be assembled into a J2EE module (jar, war, and/or ear) and deployed into its container.

Q. Why do you need a J2EE server? What services does a J2EE server provide?

A J2EE server provides **system level support services** such us security, transaction management, JNDI (Java Naming and Directory Interface) lookups, remote access etc. J2EE architecture provides configurable and non-configurable services. The configurable service enables the J2EE components within the same J2EE application to behave differently based on where they are deployed. For example the security settings can be different for the same J2EE application in two different production environments. The non-configurable services include enterprise bean (EJB) and servlet life cycle management, resource pooling etc.

Server supports various protocols. **Protocols** are used for access to Internet services. J2EE platform supports HTTP (HyperText Transfer Protocol), TCP/IP (Transmission Control Protocol / Internet Protocol), RMI (Remote Method Invocation), SOAP (Simple Object Access Protocol) and SSL (Secured Socket Layer) protocol.

The J2EE API can be summarized as follows:

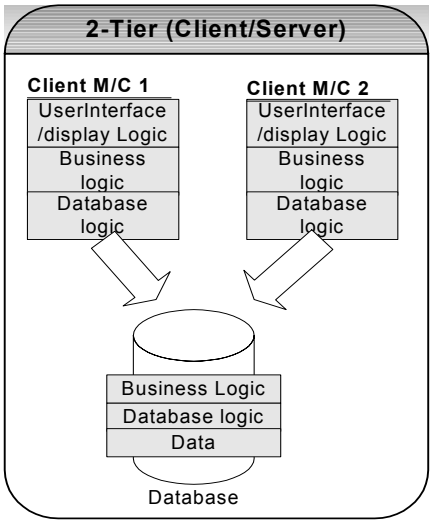
J2EE technology category	API (Application Programming Interface)
Component model technology	**Java Servlet**, JavaServer Pages(**JSP**), Enterprise JavaBeans(**EJB**).
Web Services technology	**JAXP** (Java API for XML Processing), **JAXR** (Java API for XML Registries), **SAAJ** (SOAP with attachment API for Java), **JAX-RPC** (Java API for XML-based RPC), **JAX-WS** (Java API for XML-based Web Services).

Other	**JDBC** (**J**ava **D**ata**B**ase **C**onnectivity), **JNDI** (**J**ava **N**aming and **D**irectory **I**nterface), **JMS** (**J**ava **M**essaging **S**ervice), **JCA** (**J**2EE **C**onnector **A**rchitecture), **JTA** (**J**ava **T**ransaction **A**PI), **JavaMail**, **JAF** (**J**avaBeans **A**ctivation **F**ramework – used by JavaMail), **JAAS** (**J**ava **A**uthentication and **A**uthorization **S**ervice), **JMX** (**J**ava **M**anagement e**X**tensions).

Q 02: Explain the J2EE 3-tier or n-tier architecture? SF DC FAQ

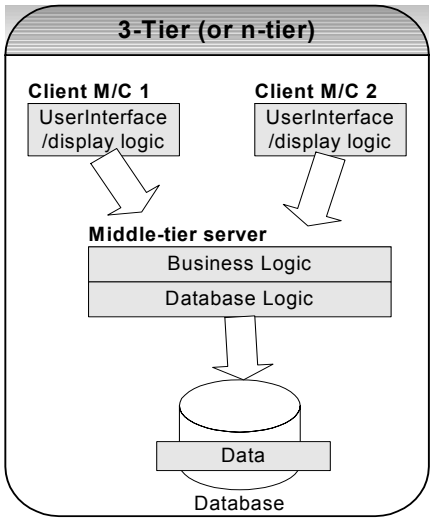
A 02: This is a very commonly asked question. Be prepared to draw some diagrams on the board. The J2EE platform is a multi-tiered system. A tier is a logical or functional partitioning of a system.

2 – tier system



When the developers are not disciplined, the display logic, business logic and database logic are muddled up and/or duplicated in a 2-tier client server system.

3 – tier system



The advantages of the multi-tier architecture are:

- Forced separation of user interface logic and business logic.
- Business logic sits on small number of centralized machines (may be just one).
- Easy to maintain, to manage, to scale, loosely coupled etc.

Each tier is assigned a unique responsibility in a 3-tier system. Each tier is logically separated and loosely coupled from each other, and may be distributed.

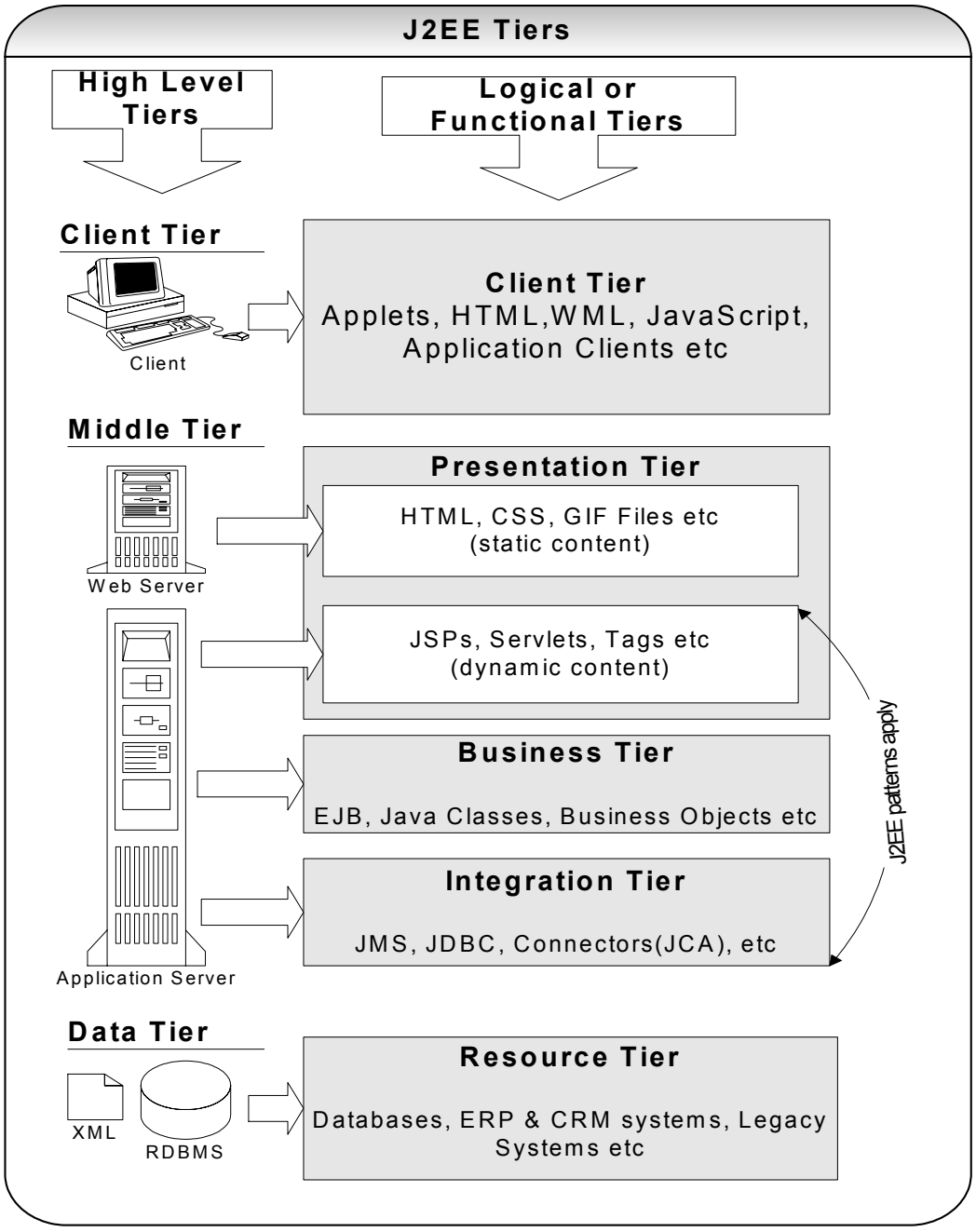
Client tier represents Web browser, a Java or other application, Applet, WAP phone etc. The client tier makes requests to the Web server who will be serving the request by either returning static content if it is present in the Web server or forwards the request to either Servlet or JSP in the application server for either static or dynamic content.

Presentation tier encapsulates the presentation logic required to serve clients. A Servlet or JSP in the presentation tier intercepts client requests, manages logons, sessions, accesses the business services, and finally constructs a response, which gets delivered to client.

Business tier provides the business services. This tier contains the business logic and the business data. All the business logic is centralized into this tier as opposed to 2-tier systems where the business logic is scattered between the front end and the backend. The benefit of having a centralized business tier is that same business logic can support different types of clients like browser, WAP (Wireless Application Protocol) client, other stand-alone applications written in Java, C++, C# etc.

Integration tier is responsible for communicating with external resources such as databases, legacy systems, ERP systems, messaging systems like MQSeries etc. The components in this tier use JDBC, JMS, J2EE Connector Architecture (JCA) and some proprietary middleware to access the resource tier.

Resource tier is the external resource such as a database, ERP system, Mainframe system etc responsible for storing the data. This tier is also known as Data Tier or EIS (Enterprise Information System) Tier.



Note: On a high level J2EE can be construed as a **3-tier** system consisting of **Client Tier, Middle Tier** (or Application Tier) and **Data Tier**. But logically or functionally J2EE is a multi-tier (or n-tier) platform.

The advantages of a 3-tiered or n-tiered application: 3-tier or multi-tier architectures force separation among presentation logic, business logic and database logic. Let us look at some of the key benefits:

- **Manageability:** Each tier can be monitored, tuned and upgraded independently and different people can have clearly defined responsibilities.
- **Scalability:** More hardware can be added and allows clustering (i.e. horizontal scaling).
- **Maintainability:** Changes and upgrades can be performed without affecting other components.
- **Availability:** Clustering and load balancing can provide availability.
- **Extensibility:** Additional features can be easily added.

The following diagram gives you a bigger picture of the logical tiers and the components.

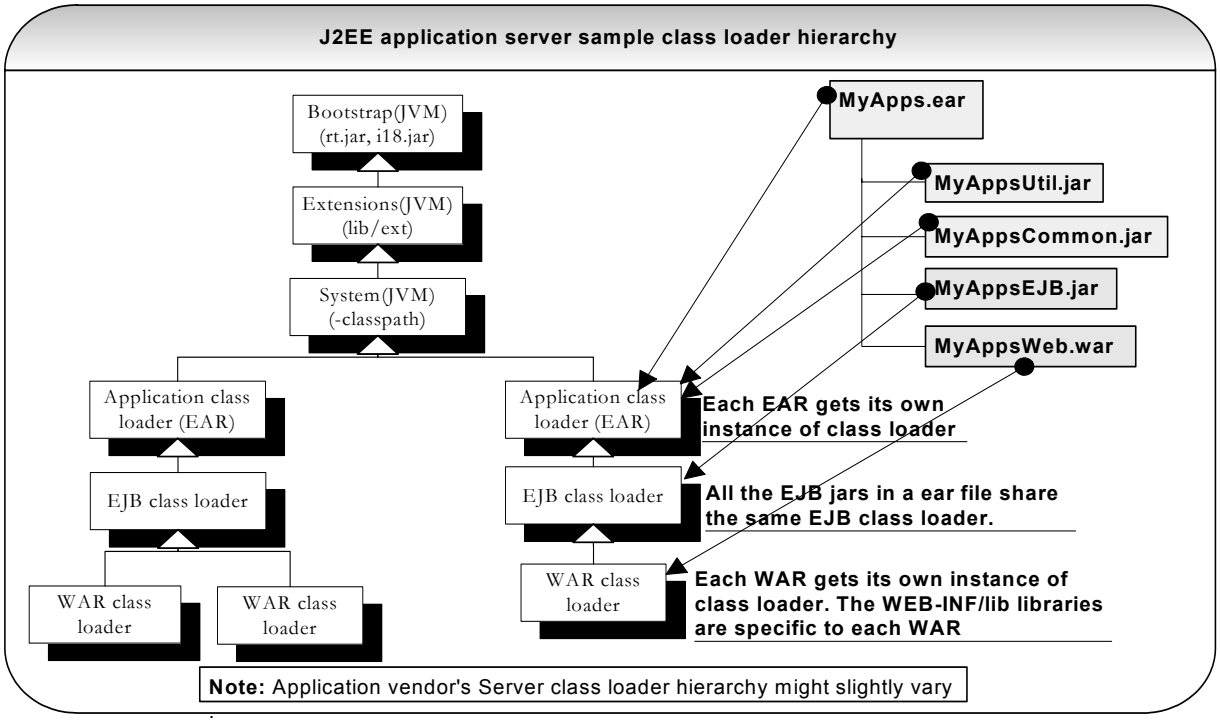
```
  <security-role>
      <description>Advisor</description>
      <role-name>advisor</role-name>
  </security-role>

</web-app>
```

Q 08: Explain J2EE class loaders? **SF**

A 08: J2EE application server sample class loader hierarchy is shown below. (Also refer to **Q5** in Java section). As per the diagram the J2EE application specific class loaders are children of the "*System –classpath"* class loader. When the parent class loader is above the "*System –classpath"* class loader in the hierarchy as shown in the diagram (i.e. bootstrap class loader or extensions class loader) then child class loaders implicitly have visibility to the classes loaded by its parents. When a parent class loader is below a "*System -classpath"* class loader in the hierarchy then the child class loaders will only have visibility into the classes loaded by its parents **only if they are explicitly specified in a manifest file** (MANIFEST.MF) of the child class loader.

Example As per the diagram, if the EJB module *MyAppsEJB.jar* wants to refer to *MyAppsCommon.jar* and *MyAppsUtil.jar* we need to add the following entry in the MyAppsEJB.jar's manifest file MANIFEST.MF.

```
class-path: MyAppsCommon.jar MyAppsUtil.jar
```



This is because the application (EAR) class loader loads the *MyAppsCommon.jar* and *MyAppsUtil*.jar. The EJB class loader loads the *MyAppsEJB.jar*, which is the child class loader of the application class loader. The WAR class loader loads the MyAppsWeb.war.

Every J2EE application or EAR gets its own instance of the application class loader. This class loader is also responsible for loading all the dependency jar files, which are shared by both Web and EJB modules. **For example** third party libraries like log4j, utility (e.g. MyAppsUtility.jar) and common (e.g. MyAppsCommon.jar) jars etc. Any application specific exception like *MyApplicationException* thrown by an EJB module should be caught by a Web module. So the exception class *MyApplicationException* is shared by both Web and EJB modules.

The key difference between the EJB and WAR class loader is that all the EJB jars in the application **share the same EJB class loader** whereas WAR files get their own class loader. This is because the EJBs have inherent relationship between one another (i.e. EJB-EJB communication between EJBs in different applications but hosted on the same JVM) but the Web modules do not. Every WAR file should be able to have its own WEB-INF/lib third

SECTION THREE

How would you go about…?

- This section basically assesses your knowledge of how to perform certain tasks like documenting your project, identifying any potential performance, memory, transactional, and/or design issues etc.

- It also assesses if you have performed any of these tasks before. If you have not done a particular task, you can demonstrate that you know how to go about it if the task is assigned to you.

- This section also recaps some of the key considerations discussed in the **Java** and **Enterprise** sections. Question numbers are used for cross-referencing with **Java** and **Enterprise** sections.

- **Q11** & **Q14** are discussed in more detail and can be used as a quick reference guide in a software project. All the other questions excluding **Q11** & **Q14** can be read just before an interview.

Q 11: How would you go about applying the design patterns in your Java/J2EE application?
A 11: It is really worth reading books and articles on design patterns. It is sometimes hard to remember the design patterns, which you do not use regularly. So if you do not know a particular design pattern you can always honestly say that you have not used it and subsequently suggest that you can explain another design pattern, which you have used recently or more often. It is always challenging to decide, which design pattern to use when? How do you improve your design pattern skills? Practice, practice, practice. I have listed some of the design patterns below with scenarios and examples:

To understand design patterns you need to have a basic understanding of object-oriented concepts like:

Decomposition: The process of dividing a problem into smaller pieces (i.e. divide and conquer approach). The following examples will break different scenarios into objects, each with specific responsibilities. A good decomposition will often result in improved reusability.

Polymorphism, Inheritance, and Encapsulation: Refer **Q10** in Java section.

Loose coupling: The process of making objects independent of each other rather than dependent of one another. Loosely coupled objects are easier to reuse and change.

Note: To keep it simple, System.out.println(…) is used. In real practice, use logging frameworks like log4j. Also package constructs are not shown. In real practice, each class should be stored in their relevant packages like com.items etc. Feel free to try these code samples by typing them into a Java editor of your choice and run the main class *Shopping*. Also constants should be declared in a typesafe manner as shown below:

```
/**
 * use typesafe enum pattern as shown below if you are using below J2SE 5.0 or use "enum" if you are using J2SE 5.0
 */
public class ItemType {
    private final String name;

    public static final ItemType Book = new ItemType("book");
    public static final ItemType CD = new ItemType("cd");
    public static final ItemType COSMETICS = new ItemType("cosmetics");
    public static final ItemType CD_IMPORTED = new ItemType("cd_imported");

    private ItemType(String name) {this.name = name;}
    public String toString() {return name;}
    //add compareTo(), readResolve() methods etc as required ...
}
```

Scenario: A company named XYZ Retail is in the business of selling *Books*, *CDs* and *Cosmetics*. *Books* are sales tax exempt and *CDs* and *Cosmetics* have a sales tax of 10%. CDs can be imported and attracts an import tax of 5%. Write a shopping basket program, which will calculate extended price (qty * (unitprice + tax)) inclusive of tax for each item in the basket, total taxes and grand total.

Solution: Sample code for the items (i.e. Goods) sold by XYZ Retail. Let's define an *Item* interface to follow the design principle of **code to an interface not to an implementation**. **CO**

```
public interface Item {

    public static final int TYPE_BOOK = 1;
    public static final int TYPE_CD = 2;
    public static final int TYPE_COSMETICS = 3;
    public static final int TYPE_CD_IMPORTED = 4;

    public double getExtendedTax();
    public double getExtendedTaxPrice() throws ItemException;
    public void setImported(boolean b);
    public String getDescription();
}
```

The following class *Goods* cannot be instantiated (since it is **abstract**). You use this abstract class to achieve **code reuse**.

SECTION FOUR

Emerging Technologies/Frameworks…

This section covers some of the popular emerging technologies you need to be at least aware of, if you have not already used them. If there are two or more interview candidates with similar skills and experience then awareness or experience with some of the emerging technologies can play a role in the decision making. Some organizations might be considering or already started using these technologies. So it is well worth your effort to demonstrate that you understand the basic concepts or have an appreciation for the following technologies/frameworks and an eagerness to learn.

- Test Driven Development (**TDD**).
- Aspect Oriented Programming (**AOP**).
- Inversion of Control (**IoC**) (Also known as **Dependency Injection**).
- Annotation or attribute based programming (xdoclet etc).
- Spring framework.
- Hibernate framework.
- EJB 3.0
- Component based Web frameworks like (**JSF, Tapestry etc**)

Note: It is out of scope for this book to cover all of these technologies/frameworks in detail. Important and popular technologies (TDD, AOP, IoC, and Annotations) and frameworks (Hibernate, Spring, EJB 3.0) are discussed with examples. If you hire smart people with a good understanding of Java/J2EE core concepts and key areas with some basic understanding of emerging technologies and frameworks then their current skills are not as important as their ability to learn quickly, eagerness to learn, and be productive.

Q. What is the hot trend in Enterprise apps these days?

This section covers some of the recent and popular design paradigms such as Plain Old Java Objects (**POJO**s) and Plain Old Java Interfaces (**POJI**) based services and interceptors, Aspect Oriented Programming (**AOP**), Dependency Injection (aka **IoC)**, attributes or annotations oriented programming, etc and tools and frameworks which apply these new paradigms such as **Spring** (IoC and AOP), **Hibernate** (O/R mapping), **EJB 3.0** (POJO, POJI, and annotations), **XDoclet** (attributes oriented programming) **, JSF** (component based Web framework)**, Tapestry** (component based Web framework) etc. All these have emerged over the past 2-4 years.

Q. Why should you seriously consider these technologies?

These new paradigms and frameworks can offer great benefits such as ease of maintenance, reduction in code size, elimination of duplication of code, ease of unit testing, loose coupling among components, light weight and fine grained objects, and developer productivity.

Q. How would you convince a development team to use these new paradigms/frameworks?

Build a vertical slice with some code for a business use case to demonstrate the above mentioned benefits.

Q 01: What is Test Driven Development (TDD)? **FAQ**

A 01: TDD is an iterative software development process where you **first write the test with the idea that it must fail**. This is a different approach to the traditional development where you write the application functionality first and then write test cases. The major benefit of this approach is that the code becomes thoroughly unit tested (you can use **JUnit** or other unit testing frameworks). For JUnit refer **Q14** on "How would you go about…" section. TDD is based on two important principles preached by its originator Kent Beck:

- **Write new business code only if an automated unit test has failed:** Business application requirements drive the tests and tests drive the actual functional code. Each test should test only one business concept, which means avoid writing a single test which tests withdrawing money from an account and depositing money into an account. Any change in the business requirements will impact pre and post conditions of the test. Talking about pre and post conditions, following **design by contract** methodology (Refer **Q11** in Java section) helps achieving TDD. In design by contract, you specify the pre and post conditions that act as contracts of a method, which provides a specification to write your tests against.

- **Eliminate duplication from the code:** A particular business concept should be implemented only once within the application code. Code for checking an account balance should be centralized to only one place within the application code. This makes your code decoupled, more maintainable and reusable.

I can hear some of you all saying how can we write the unit test code without the actual application code. Let's look at how it works in steps. The following steps are applied iteratively for business requirements.

STEP: 1 write some tests for a specific business requirement.

STEP: 2 write some basic structural code so that your **test compiles but the test should fail** (failures are the pillars of success). **For example** just create the necessary classes and corresponding methods with skeletal code.

STEP: 3 write the required business code to pass the tests which you wrote in step 1.

STEP: 4 finally refactor the code you just wrote to make it is as simple as it can be. You can refactor your code with confidence that if it breaks the business logic then you have unit test cases that can quickly detect it.

STEP: 5 run your tests to make sure that your refactored code still passes the tests.

STEP: 6 Repeat steps 1-5 for another business requirement.

To write tests efficiently some basic guidelines need to be followed:

- You should be able to run each test in isolation and in any order.
- The test code should not have any duplicate business logic.
- You should test for all the pre and post conditions as well as exceptions.
- Each test should concentrate on one business requirement as mentioned earlier.
- There are many ways to write test conditions so proper care and attention should be taken. In some cases pair programming can help by allowing two brains to work in collaboration. You should have strategies to overcome issues around state of data in RDBMS (Should you persist sample test data, which is a snapshot of your actual data prior to running your tests? Or should you hard code data? Or Should you combine both strategies? Etc).

SECTION FIVE

Sample interview questions…

Tips:

- Try to find out the needs of the project in which you will be working and the needs of the people within the project.
- 80% of the interview questions are based on your own resume.
- Where possible briefly demonstrate how you applied your skills/knowledge in the key areas as described in this book. Find the right time to raise questions and answer those questions to show your strength.
- Be honest to answer technical questions, you are not expected to know everything (for example you might know a few design patterns but not all of them etc).
- Do not be critical, focus on what you can do. Also try to be humorous.
- Do not act in superior way.

General Tip #11:

There is a difference between looking excited about a job or a job offer and looking desperate for one. Do not immediately jump at the opportunity. If you have any impending interviews ask the interviewer for some time to respond to the offer. Never give into the pressure (e.g. this is the best job and if you do not take it right now you might miss out etc) from the job agencies. Interviewers are generally happy to wait for the right candidate. Give yourself attention to all the aspects on offer like salary, type of industry (finance, telecom, consulting etc), opportunity for growth, type of project (large scale mission critical, medium sized etc), type of role (design, development and design, team lead, architect etc), type of technology used and opportunity to learn new things (e.g. Spring, Hibernate, Tapestry, JSF, Web services, messaging etc) to keep you motivated at your job as well as improve your future job prospects. Never think of salary aspect alone. You should have a long term plan. Sometimes it is worth your while to compromise on a few quid to acquire most sought after skills (at the time of writing Spring, Hibernate, JSF, Tapestry etc) and/or valuable skills (design skills, leadership skills etc). So for each interview you attend keep a checklist of aspects on offer and always act calmly and professionally to make the right decision for you.

GLOSSARY OF TERMS

TERM	DESCRIPTION
ACID	Atomicity, Consistency, Isolation, Durability.
Ajax	Asynchronous JavaScript And XML
aka	also known as.
AOP	Aspect Oriented Programming
API	Application Programming Interface
AWT	Abstract Window Toolkit
BLOB	Binary Large Object
BMP	Bean Managed Persistence
CGI	Common Gateway Interface
CLOB	Character Large OBject
CMP	Container Managed Persistence
CORBA	Common Object Request Broker Architecture
CRM	Customer Relationships Management
CRUD	Create, Read, Update and Delete
CSS	Cascading Style Sheets
csv	Comma Separated Value
CRC	Cyclic Redundancy Checks
DAO	Data Access Object
DNS	Domain Name Service
DOM	Document Object Model
DTD	Document Type Definition
EAR	Enterprise ARchive
EIS	Enterprise Information System
EJB	Enterprise JavaBean
EL	Expression Language
ERP	Enterprise Resource Planning
FDD	Feature Driven Development
GIF	Graphic Interchange Format
GOF	Gang Of Four
HQL	Hibernate Query Language.
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
I/O	Input/Output
IDE	Integrated Development Environment
IIOP	Internet Inter-ORB Protocol
IoC	Inversion of Control
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
JAAS	Java Authentication and Authorization Service
JAF	JavaBeans Activation Framework
JAR	Java ARchive
JAXB	Java API for XML Binding
JAXP	Java API for XML Parsing
JAXR	Java API for XML Registries
JAX-RPC	Java API for XML-based RPC
JAX-WS	Java API for XML-based Web Services
JCA	J2EE Connector Architecture
JDBC	Java Database Connectivity
JDK	Java Development Kit
JFC	Java Foundation Classes
JMS	Java Messaging Service
JMX	Java Management eXtensions
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JRMP	Java Remote Method Protocol
JSF	JavaServer Faces
JSP	Java Server Pages
JSTL	Java Standard Tag Library
JTA	Java Transaction API
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol

MOM	Message Oriented Middleware
MVC	Model View Controller
NDS	Novell Directory Service
NIO	New I/O
O/R mapping	Object to Relational mapping.
OO	Object Oriented
OOP	Object Oriented Programming
OOPL	Object Oriented Programming Language
ORB	Object Request Broker
ORM	Object to Relational Mapping.
POJI	Plain Old Java Interface
POJO	Plain Old Java Object
RAR	Resource adapter ARchive
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RUP	Rational Unified Process
SAAJ	SOAP with attachment API for Java
SAX	Simple API for XML
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TDD	Test Driven Development
UDDI	Universal Description Discovery and Integration
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	
VO	Value Object which is a plain Java class which has attributes or fields and corresponding getter → getXXX() and setter → setXXX() methods .
WAR	Web ARchive
WML	Wireless Markup Language
WSDL	Web Service Description Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XP	Extreme Programming
XPath	XML Path
XSD	XML Schema Definition
XSL	Extensible Style Language
XSL-FO	Extensible Style Language – Formatting Objects
XSLT	Extensible Style Language Transformation

RESOURCES

Articles

- Sun Java Certified Enterprise Architect by Leo Crawford on http://www.leocrawford.org.uk/work/jcea/part1/index.html.
- Practical UML: A Hands-On Introduction for Developers by Randy Miller on http://bdn.borland.com/article/0,1410,31863,00.html
- W3 Schools on http://www.w3schools.com/default.asp.
- LDAP basics on http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzahy/rzahyovrco.htm.
- Java World articles on design patterns: http://www.javaworld.com/columns/jw-Java-design-patterns-index.shtml.
- Web Servers vs. App Servers: Choosing Between the Two By Nelson King on http://www.serverwatch.com/tutorials/article.php/1355131.
- Follow the Chain of Responsibility by David Geary on Java World - http://www.javaworld.com/javaworld/jw-08-2003/jw-0829-designpatterns.html.
- J2EE Design Patterns by Sue Spielman on http://www.onjava.com/pub/a/onjava/2002/01/16/patterns.html.
- The New Methodology by Martin Fowler on http://www.martinfowler.com/articles/newMethodology.html.
- Merlin brings nonblocking I/O to the Java platform by Aruna Kalagnanam and Balu G on http://www.ibm.com//developerworks/Java/library/j-javaio.
- Hibernate Tips and Pitfalls by Phil Zoio on http://www.realsolve.co.uk/site/tech/hib-tip-pitfall-series.php.
- Hibernate Reference Documentation on http://www.hibernate.org/hib_docs/reference/en/html_single/.
- Object-relation mapping without the container by Richard Hightower on http://www-128.ibm.com/developerworks/library/j-hibern/?ca=dnt515.
- Object to Relational Mapping and Relationships with Hibernate by Mark Eagle on http://www.meagle.com:8080/hibernate.jsp.
- Mapping Objects to Relational databases: O/R Mapping In detail by Scott W. Ambler on http://www.agiledata.org/essays/mappingObjects.html.
- I want my AOP by Ramnivas Laddad on Java World.
- WebSphere Application Server 5.0 for iSeries – Performance Considerations by Jill Peterson.
- Dependency Injection using pico container by Subbu Ramanathan .
- WebSphere Application Server & Database Performance tuning by Michael S. Pallos on http://www.bizforum.org/whitepapers/candle-5.htm.
- A beginners guide to Dependency Injection by Dhananjay Nene on http://www.theserverside.com/articles/article.tss?l=IoCBeginners.
- The Spring series: Introduction to the Spring framework by Naveen Balani on http://www-128.ibm.com/developerworks/web/library/wa-spring1.
- The Spring Framework by Benoy Jose.
- Inversion of Control Containersband the Dependency Injection pattern by Martin Fowler.
- Migrate J2EE Applications for EJB 3.0 by Debu Panda on JavaPro.
- EJB 3.0 in a nutshell by Anil Sharma on JavaWorld.
- Preparing for EJB 3.0 by Mike Keith on ORACLE Technology Network.
- Simplify enterprise Java development with EJB 3.0 by Michael Juntao Yuan on JavaWorld.
- J2SE: New I/O by John Zukowski on http://java.sun.com/developer/technicalArticles/releases/nio/.
- High-Performance I/O arrives by Danniel F. Savarese on JavaPro.
- Hibernate – Proxy Visitor Pattern by Kurtis Williams.
- Best Practices for Exception Handling by Gunjan Doshi.
- Three Rules for Effective Exception Handling by Jim Cushing.
- LDAP and JNDI: Together forever – by Sameer Tyagi.
- Introduction To LDAP – by Brad Marshall.
- Java theory and practice: Decorating with dynamic proxies by Brian Goetz.
- Java Dynamic Proxies: One Step from Aspect-Oriented Programming by Lara D'Abreo.
- Java Design Patterns on http://www.allapplabs.com/java_design_patterns .
- Software Design Patterns on http://www.dofactory.com/Patterns/Patterns.aspx .
- JRun: Core Dump and Dr. Watson Errors on http://www.macromedia.com/cfusion/knowledgebase/index.cfm?id=tn_17534
- The Guerrilla Guide to Interviewing by Joel Spolsky at http://www.joelonsoftware.com/printerFriendly/articles/fog0000000073.html
- The Riddle of Job Interviews by Kate Kane at http://www.fastcompany.com/online/01/jobint_Printer_Fiendly.html
- An Introduction to Aspect-Oiented Programming with the Spring Framework, Part 1 by Russell Miles at http://www.onjava.com/lpt/a/4994
- 5 Habits Of Best Software Developers by Angusman Chakraborty at http://blog.taragana.com/index.php/archive/5-habits-of-best-doftware-developers/
- Getting started with Hibernate by Alan P Saxton at http://www.cs.bham.ac.uk/~aps/syllabi/2004_2005/issws/h03/hibernate.html
- Hibernate Tips by Jason Carreira at http://jroller.com/page/jcarreira/20050223
- Five Things I Love About Spring by Bruce A. Tate at http://www.onjava.com/lpt/a/5833
- Service-oriented modeling and architecture by Ali Arsanjani , Ph.D at http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/
- Delving into Service-Oriented Architecture by Bernhad Borges, Kerrie Holley and Ali Arsanjani at http://www.developer.com/design/print.php/10925_3409221_1
- SOA: Are We Reinventing the Wheel? By Nick Simha at http://dev2dev.bea.com/lpt/a/435
- Getting a little closer to SOA by Fabrice Marguerie at http://madgeek.com/Articles/SOA/EN/SOA-Softly.html
- What is sevice-oriented architecture by Raghu R. Kodali at http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa_p.html

- J2EE-Supported Web Service standards and Technologies by Vijay Ramachandran, Sean Brydon, Greg Murray. Inderjeet Singh, Beth Stearns, Thierry Violleau.
- J2EE 1.4 eases Web service development by Frank Sommers at http://www.javaworld.com/javaworld/jw-06-2003/jw-0620-webservices_p.html
- A developer's introduction to JAX-RPC, Part 1 & 2 by Joshy Joseph at http://www-128.ibm.com/developerworks/webservices/library/
- Developing Web Services with Java 2 Platform, Enterprise Edition (J2EE) 1.4 Platform by Qusay H. Mahmoud at http://java.sun.com/developer/technicalArticles/J2EE/j2ee_ws/
- Scriptless JSP Pages: The Front Man by Bear Bibeault at http://www.javaranch.com/journal/200603/Journal200603.jsp
- Advanced DAO programming by Sean Sullivan at http://www-128.ibm.com/developerworks/library/j-dao/
- Understanding JavaServer Pages Model 2 architecture by Govind Seshadri at http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc_p.html
- A Fast Introduction to Basic Servlet Programming by Marty Hall at http://www.informit.com/articles/printerfriendly.asp/p=29817&r1=1
- What's new in J2Se 5.0? based on Joshua Bloch's on-line talk.
- Introducing Java 5 by Andy Grant at http://www.sitepoint.com/print/introducing-java-5
- Experiences with the New Java 5 Language Features by Jess Garms and Tim Hanson at http://dev2dev.bea.com/lpt/a/442
- Five Favorite Features from 5.0 by David Flanagan at http://www.onjava.com/lpt/a/5799
- First among equals by Kevlin Henney at http://www.regdeveloper.com/2005/12/29/first_among_equals/print.html
- Painting in AWT and Swing by Amy Fowler.
- A Hands-On Introduction for Developers by Randy Miller at http://bdn.borland.com/article/0,1410,31863,00.html
-

- www.javaworld.com articles.
- http://www-128.ibm.com/developerworks/java articles.
- http://www.devx.com/java articles.
- www.theserverside.com/tss articles.
- http://javaboutique.internet.com/articles articles.

Books

- Beginning Java 2 by Ivor Horton.
- Design Patterns by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (GoF) .
- UML Distilled by Martin Fowler, Kendall Scott .
- Mastering Enterprise Java Beans II by Ed Roman, Scott Ambler, Tyler Jewell, Floyd Marinescu.
- EJB Design Patterns by Floyd Marinescu .
- Sun Certified Enterprise Architect for J2EE Technology Study Guide by Mark Cade and Simon Roberts.
- Professional Java Server Programming - J2EE edition by Wrox publication.
- Design Patterns Java Companion by James W. Cooper (Free download: http://www.patterndepot.com/put/8/JavaPatterns.htm).
- Test Driven Development – By Example, by Kent Beck.
- Effective Java – programming language guide by Joshua Bloch

INDEX