

Core Java Career Essentials

Focusing on

**Java platform, language, classes, objects, OO concepts
& principles, data structures, algorithms, and pattern
matching essentials**

By

Arulkumaran Kumaraswamipillai
Sivayini Arulkumaran

Core Java Career Essentials

Focusing on platform, language, classes, objects, collections, and logic essentials

Copy Right 2011

The authors have made every effort in the preparation of this book to ensure the accuracy of the information. However, information in this book is sold without warranty either expressed or implied. The authors will not be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

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

More Java/JEE career resources are available at
<http://www.lulu.com/java-success>

First Edition : April 2011

Thanks to the reviewers: Ankit Garg, Devaka Cooray, Roberto Perillo, Rod Good, and Sean Clynes.

Table of Contents

Getting Started	5
How can this book help you?.....	6
Why is this a PERFECT companion?.....	8
What are the technical key areas?.....	9
Platform Essentials.....	15
Why use Java?	16
Java platform basics – JDK, JRE, JVM, etc.....	21
Setting up and running Java.....	30
How are your classes loaded?.....	34
Compile-time versus runtime.....	39
Knowing your way around Java.....	51
Judging Experience	52
Gauging Your Experience with UNIX	88
Exposure to tools, technologies, and frameworks.....	103
Documenting your Java applications.....	115
Ensuring code quality.	122
Language Essentials.....	133
Valid identifiers and reserved keywords.....	135
Choosing the right data types.....	144
Widening versus narrowing conversions.....	150
Understanding the operator precedence.....	159
Thinking in binary and bitwise operations	162
Initializing your data correctly.....	176
Constants and static imports.....	182
Modifiers and where are they applicable?	186
Methods and constructors.....	196
Stack versus Heap.....	209
Java is pass-by-value.....	211
Recursive functions.....	214
Class, instance, and local variables.....	217
Classes and Interfaces Essentials.....	225
Working with classes and interfaces	226
Subclassing, overriding, and hiding (aka shadowing)	227
Designing your classes and interfaces.....	233
Working with abstract classes and interfaces.....	247

Inheritance versus composition.....	260
Applying the design principles.....	268
Designing an application or a system.....	281
Class invariant and design by contract	285
Working with inner classes.....	290
Packaging your classes to avoid conflicts.....	300
Objects Essentials.....	305
Working with Objects.....	306
Cloning Objects.....	312
Casting Objects.....	314
Immutable Objects.....	318
Working with Enumerations.....	326
Understanding “==” versus equals().....	332
Working with the String class.....	338
Type safety with Generics.....	346
Serializing your objects.....	364
Garbage Collection.....	372
Logic and Data structures Essentials.....	383
Java Flow Control.....	384
Java Data structures.....	390
Sorting Elements.....	422
Algorithms.....	427
Logic Processing.....	449
Exception handling.....	471
Matching patterns with regular expressions.....	487
Matching patterns with Regular Expressions.....	488
Job Interview Tips.....	520
Interviews are not technical contests.....	521
Don't wait to be asked	523
Think out loud and brainstorm with the interviewers	524
Sometimes knowing something is better than knowing nothing	526
Interviewers place a lot of value in "I don't know" over inventing answers ...	526
An interview is a two way street	527
Open-ended questions are your opportunity	528
Books can impart knowledge, but cannot give you the much needed experience	528
Glossary & Index.....	530

Section-1:

Getting Started

- How can this book help you?
- Why is this a PERFECT companion?
- What are the technical key areas?

So you have a java interview coming up in a few days or you want to impress your peers and superiors with your technical strengths during code review sessions, team meetings, and stand-ups, and concerned about how to make a good impression? you don't need to worry if you are familiar with the fundamentals. Most Java interviews and technical challenges you face at work are structured around the fundamentals and how well you communicate those fundamentals. So regularly brushing up on these fundamentals really pays off.

Your analytical, problem solving, and coding skills will also be under scrutiny along with your ability to get the job done with the right tools. If your fundamentals are clear and know what tools to use, you can tackle any questions, and find solutions to any problems and challenges you face. Even if you don't have the exact answer for a problem, you will know how to go about solving them with a little research if you have a solid grounding in the fundamentals covered in this book.

A little preparation can make a huge difference to your career success. Preparation can help you communicate your thoughts more clearly with examples and illustrations. Preparation can make a good and lasting impression on those who talk with you during your interviews and team meetings. This impression will be partly influenced by how prepared you are and how knowledgeable you are about your industry and the challenges it faces. It will also be influenced by your appearance, attitude, enthusiasm, and confidence. Good preparation breeds confidence and it shows in the interviews and team meetings. So prepare well in advance if you just begun to see yourself in your dream company or would like to go places in your chosen field.

How can this book help you?

This unique “questions and answers approach” with tagged technical key areas, diagrams, code snippets, and examples can help you excel in your chosen Java based profession. This is more than just an interview preparation guide. Let's look at ways in which we believe this book can benefit a reader.

- Helps you brush up or be aware of the basics in software development that you must know. Preparing for the most common interview questions will increase your chances. If you fail to answer these basic questions, your chances of succeeding in interviews will be very slim. In fact, 40% - 60% of the professionals irrespective of their level of experience fail to make an impression in one or more of the following most common areas.
 - ◆ **Coding:** You may have to write some simple code, with correct syntax and logic in Java. You might be asked to explain and critique on snippets of code. You might be asked to demonstrate both recursive and iterative approaches to a given problem. You will be assessed on your ability to think about the exceptional cases, exit conditions, and algorithms.
 - ◆ **OO design:** You will be asked to define basic OO concepts and principles, and come up with the classes and interfaces to model a simple problem. A good weeder question would be to judge your ability to decide when to use attributes, inheritance, and composition. You might be asked to critique a system or platform you had worked on. A good OO design question can also test your coding skills and domain knowledge.
 - ◆ **Language fundamentals and best practices:** You must have a solid grasp of the language fundamentals. You might be asked what parts of Java you don't like and why? You must know about the common pitfalls, anti-patterns, and how to avoid them. You must know the differences between abstract classes and interfaces, how the garbage collection works?, etc and relevant best practices.
 - ◆ **Data structures:** You must demonstrate basic knowledge of the most common data structures. Most candidates know about arrays, sets, lists, and maps but fail to mention trees and graphs. You will be quizzed on real life examples of where to use a particular data structure and big-O performance

(e.g. linear, logarithmic, exponential, etc) of various operations like find, insert, and delete. It is also worth knowing the basic sorting and searching algorithms.

- ◆ **Bits, bytes, and data types:** You must demonstrate how good you are technically. You must know the logical operations like AND, OR, NOT, and XOR and where to use them in real life examples for things like setting or testing for a specific bit. Using the wrong data types in your program can bring the whole application down. You must also know the difference between signed and unsigned data types and narrowing versus widening conversions.
- ◆ **Right tools to use:** Knowing the right tools will make you more productive without having to reinvent the wheel, and get the job done more effectively and on time. These tools can vary from harnessing the power of Unix scripts with regular expressions or embedded groovy scripts to tools like Wireshark to sniff the packets.

The prospective employers generally determine if you “know it”, “knowledgeable enough to learn it on the job” or “have no clue at all”. You surely don't want to be in the category of “have no clue at all”.

- Helps you refresh your memory or learn answers to the technical questions that are frequently asked in interviews. This can build up your confidence and it really shows in interviews. Software development field is very vast and competitive, and it pays to jog your memory even if you are an experienced professional.
- All the other applicants may have much the same degrees, certifications, skills, and experience. But those who are in the know how can make it all come alive and seem real by selling their practical and technical competencies with illustrations, examples, and enthusiasm. If you don't relate your qualifications, skills, experience, and abilities to demonstrated results, most employers will tend to think that you have a great deal of raw potential – but limited immediate usefulness.
- Helps you think aloud for some of the more difficult to tackle scenario based and open-ended questions where the interviewers will be more interested in evaluating your thought process and approach than expecting a detailed solution or answer.

Getting Started

- Helps you progress in your career a lot quicker by enabling you to pro-actively learn and apply the core technical key areas that will otherwise take years to fully understand if you rely on your experience alone. Some mistakes are too expensive to learn on the job.

This book complements our previously published book in 2005, entitled “Java/J2EE Job Interview Companion”, which was well received, and the purpose of this book is to keep up with the recent developments, cover more on coding, open-ended, and scenario based interview Q&A, and to incorporate some of the constructive criticisms from the readers and reviewers. The “Java/J2EE Job Interview Companion” gives an overview of many topics from Java to J2EE, XML to UML, and frameworks to emerging technologies. The “Core Java Career Essentials” as the name implies, focuses solely on the essentials in more depth. The ensuing career essentials series will cover the topics not covered here.

“Knowledge is knowing what to do, skill is knowing how to do,
virtue is getting it done.” – Norman Vincent Peale

Why is this a PERFECT companion?

The primary objective is to make your learning or brushing up a PERFECT experience, which stands for **P**ractical, **E**njoyable, **R**ewarding, **F**ocused, **E**xamples driven, **C**oncise, and **T**hought-provoking.

- **Practical:** What is the point in learning bitwise operators without knowing where to apply and when to use them? This book is full of practical examples, workable code, best practices, and potential pitfalls.
- **Enjoyable:** Nothing is more enjoyable as a developer than cutting quality and workable code. That is why there are 100+ executable code samples as opposed to just code snippets. Feel free to experiment with the code samples.
- **Rewarding:** The basics described here with examples and sample code can bring success in job interviews, technical tests, code reviews, and performance appraisals by learning to sell yourself more effectively.

- **Focused:** Our last book entitled “Java/J2EE Job Interview Companion” was providing a bird's eye view of a vast range of technologies and frameworks. While this approach was well received and provided a road map for job interviews, it lacked some level of details. This book is focused on more details to extend its usefulness beyond just interviews by covering a limited number of topics in more depth.
- **Examples driven:** Prospective employers are not just looking for professionals with just academic qualifications. They are looking for professionals with experience. You cannot drive a car with just theoretical knowledge. Software development is no different. So use these examples to code, code, and more code.
- **Concise:** Who wants to read pages after pages about a single topic. There is always Google to perform additional research on a as needed basis.
- **Thought-provoking:** The questions and answers approach gives a different perspective to learning and brushing up. It will get you thinking and asking the right questions when you are working on a project. Asking the right questions and spotting the potential pitfalls can present you in a better light for potential promotions and pay rises.

What are the technical key areas?

Writing a quality software is a very complex process. It must not only meet all the functional requirements to satisfy the business needs, but also should be robust, maintainable, testable, and flexible enough to adapt to growing and changing business needs. It must also address non-functional aspects like,

- Adequate logging, auditing, and alarms to provide better problem diagnostics and customer support.
- Security, data integrity, data retention, fail over and disaster recovery strategies.
- Being up 24x7, performance metrics, and adherence to SLAs (Service Level Agreements) like response times, timeouts, etc. Service assurance can help improve customer trust and experience in the system.

If you just take technical design alone, there are many pros and cons for each approach

Getting Started

along with likely trade-offs to be made in your design decisions. Understanding the key areas will help you not only write quality software, but also excel in your chosen career. How do you excel in your career by understanding the key areas you may ask?

Firstly, in your regular job,

- You can earn the appreciation of your superiors and peers by pro-actively and retro-actively solving everyday problems by understanding the key areas. For example, you can resolve problems relating to performance, memory, concurrency, language fundamentals, etc. Some of the subtle issues can be hard to understand, reproduce and solve without the adequate technical skills, and it really pays to have the experience and good understanding of the basics in these key areas.
- Understanding some of the nuances and best practices can help you write less error-prone and more robust code. Writing maintainable, supportable, readable, and testable code will not go unnoticed in code reviews and can also earn you the much needed recognition. You can also come up with intelligent questions or suggestions in team/stand-up meetings and group discussions. You can pro-actively identify potential issues not only in your own code or design, but also in others' code and design to become a great contributor.
- You can build up a reputation as a “go to person” by complementing your technical skills in these key areas with soft-skills and personal attributes to mentor others and get things done.

Secondly, in your job interviews,

- You can impress your interviewers by highlighting your past achievements in these key areas. For example, you can make use of open-ended questions like “Tell me about your self?” or “Why do you like software development?” to highlight your strengths and accomplishments.
- Many software projects face performance bottlenecks, design inefficiencies, concurrency issues, inadequate software development processes, etc. Hence the prospective employers will make sure that they hire the right candidate by asking questions relating to these key areas. Even if they don't cover any particular key area you are good at like performance tuning or concurrency management, you can bring it up yourself by asking intelligent questions like “What are the challenges faced by your

current software? and do you face any performance issues or customer complaints that are hard to reproduce? ”.

- Better candidates will control the interview in a nicer way as they understand it is a two way process. They will quickly understand what the prospective interviewer is looking for and reflect back on their past experience to relate how they went about resolving problems and adding value.

Let's look at the key areas relating to software development. Please be aware that not all key areas are covered in this book.

1. **L**anguage **F**undamentals **LF**
2. **S**pecification **F**undamentals **SF**
3. **D**esign **C**oncepts **DC**
4. **D**esign **P**atterns **DP**
5. **P**erformance **C**onsiderations **PC**
6. **M**emory **C**onsiderations **MC**
7. **T**ransaction **M**anagement **TM**
8. **C**oncurrency **C**onsiderations **CC**
9. **S**Calability **SC**
10. **E**xception **H**andling **EH**
11. **B**est **P**ractices **BP**
12. **S**oftware **D**evelopment **P**rocesses **SP**
13. **S**Ecurity **SE**
14. **C**Oding for readability, maintainability, supportability, extendability, and testability **CO**
15. **Q**uality of **S**ervice **QS**
16. **P**latform **F**undamentals **PF**

Most of the questions are tagged with relevant technical key area(s) shown above. Some questions are also tagged with question type and/or its popularity as shown below:

- Open-Ended Question **OEQ**
- Scenario Based Question **SBQ**
- COding Question **COQ**
- IMpossible Question **IMQ**
- Frequently Asked Question **FAQ**

Getting Started

Speaking the same language as your prospective employers from both technical and business perspective will help you establish a level of comfort with them. Discuss things in simple terms so that anyone can understand. It is okay if you struggle a little and then figure it out with minor hints or prompting from the interviewers. It is natural to be a bit rusty as the technologies are pretty vast these days. But it is not a good sign to be completely clueless or badly confused, especially with the core fundamentals. These career companions and career essentials will boost your knowledge and confidence to succeed in your career.

Take the initiative to grasp a potential employer's problems, challenges, and requirements to marry them with your past experience and accomplishments. This is more important than just having the necessary experience or understanding in the technical key areas. The ensuing sections will elaborate on this.

Preview

Important:

- Sample code in this book is making use of static methods and other approaches in many places for illustration purpose and to keep things simple. This should not be construed as a best practice.
- A commercial standard code will make use of unit tests using either *JUnit* or *TestNG* instead of public static `main(String[] args)` methods for testing and most of the methods except a few utility methods will be non-static. You will hardly see `main` methods required except for initial bootstrapping of applications that are run stand-alone. It will also be making use of Object Oriented principles, Dependency Injection, Law of Demeter principle, Don't Repeat Yourself (DRY) principle, etc to make the code more reusable, testable, maintainable and readable.
- The *`System.out.println(...)`* statements should be replaced with a logging framework like *log4j*. For example,

```
private static final Log logger = LogFactory.getLog(MyClass.class);  
logger.info("logging text goes here .....");
```

- Comments are over used to explain concepts, and real code should not be cluttered with comments. The classes, methods, and variables should have meaningful names and the code should be self-explanatory.

Getting Started

Preview

Section-2:

Platform Essentials

- Why use Java?
- Java platform basics – JDK, JRE, JVM, etc.
- Setting up and running Java
- How are your classes loaded?
- Compile-time vs runtime

This section is mainly for a beginner level, but I am very surprised to see even some intermediate to senior level developers lack good understanding of these platform essentials. This section is a must read for a beginner level. The intermediate and senior level candidates are highly recommended to brush up on the following frequently asked questions,

Q1, Q2, Q3, Q7, Q8, Q9, Q10, Q17, Q18, Q19, Q20, Q21, Q22, Q24 and Q25.

Even if you have limited experience, it is better to have an idea as to how to go about getting it done than to have no idea at all. This will differentiate yourself from others who have similar experience as you. Good Java interview questions are designed in a way to analyze if you are capable of applying the basics of a programming language.

Q&As 1-18 are skipped

Q19 What tips would you give to someone who is experiencing a class loading or “Class Not Found” exception? **OEQ**

A19 “Class Not Found” exceptions could be quite tricky to troubleshoot. Firstly, determine the jar file that should contain the class file:

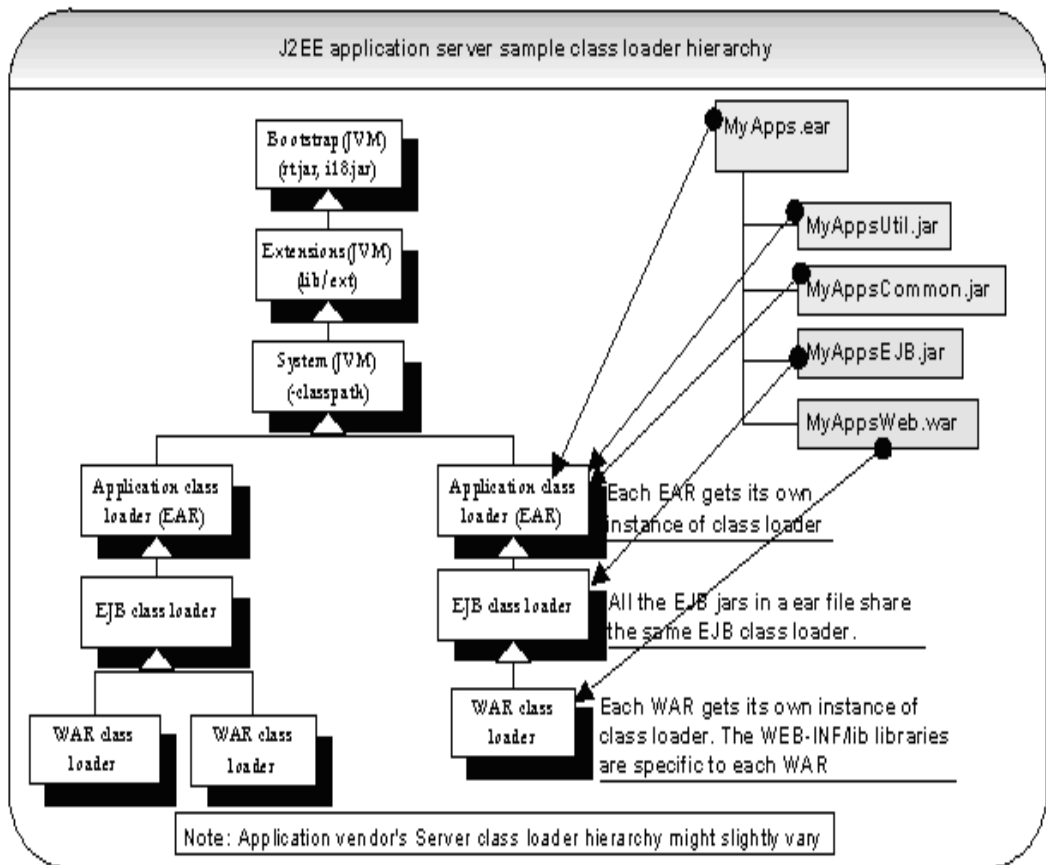
```
$ find . -name "*.jar" -print -exec jar -tf '{}' \; | grep -E "jar$|String\.class"
```

Secondly, check the version of the jar in the manifest file MANIFEST.MF, access rights (e.g. read-only) of the jar file, and any jar corruption by trying to unjar it with "*jar -xvf ...*". If the class is dynamically loaded, check if you have spelled the class name correctly.

Thirdly, check if the application is running under the right JDK? Check the `JAVA_HOME` environment property

```
$ echo $JAVA_HOME
```

Finally, you'll need to have a good understanding of your application server's class-loader hierarchy.



- Is the missing class packaged with the application (e.g., under WEB-INF/lib) and being loaded by one of the parent class-loaders? Most application servers utilize a class-loader hierarchy where WAR file's class-loader is a child of EAR class-loader which in turn is a child of the system (JVM) class-loader. Parent class-loaders can't go down to request a class loaded from a child class-loader. The problem occurs if some jars were moved to a shared library but they still depend on classes packaged with the application.
- Running different versions of the same jar loaded by the different class loaders can cause this issue.

Compile-time versus runtime

Learn to think in terms of compile-time versus runtime to solve issues quickly and identify better solutions to a given problem. Each approach has its pros and cons.

Q20 Explain compile-time versus runtime? **PF FAQ**

A20 **Compile-time** (i.e. static) is when a program text (e.g. *MyProgram.java*) is being read in, analyzed, and translated into byte code version (e.g. *MyProgram.class*) that can be run on the JVM. At compile-time you can get syntactic errors like leaving out a closing bracket or a semicolon, type safety errors like assigning a type long to type int without an explicit casting, etc.

Runtime (i.e. dynamic) is when the generated byte code is executed within the JVM. At runtime you get semantic errors like trying to examine the 4th element of a list that has only 3 elements (e.g. *ArrayIndexOutOfBoundsException*), null reference errors (e.g. *NullPointerException*), attempting to cast an object to a subclass of which it is not an instance (e.g. *ClassCastException*), etc.

Q21 Does this happen during compile-time, runtime, or both? **PF LF FAQ**

A21 **Method overloading**: Skipped....

Method overriding: Skipped....

Generics (aka type checking): Skipped....

Annotations: Skipped....

Exceptions: Skipped....

Aspect Oriented Programming (AOP): Skipped....

Q22 Can you differentiate compile-time inheritance and runtime inheritance with examples and specify which Java supports? **PF LF FAQ**

A22 Skipped....

Q23 Are marker or tag interfaces obsolete with the advent of annotations (i.e. runtime annotations)? **LF**

A23 Skipped....

Q24 What is the difference between line A & line B in the following code snippet? **COQ**
PC

```
public class ConstantFolding {  
    static final int number1 = 5;  
    static final int number2 = 6;  
    static int number3 = 5;  
    static int number4 = 6;  
  
    public static void main(String[] args) {  
        int product1 = number1 * number2;           //line A  
        int product2 = number3 * number4;           //line B  
    }  
}
```

A24 Skipped....

Q25 What is a Java debugger? What are the different ways to debug your code? **PF**

A25 Skipped....

Section-3:

Knowing your way around Java

- Judging your exposure to Java through open-ended questions.
- Gauging your exposure to Unix operating systems as most production systems are Unix based.
- Understanding your experience in regards to proper documentation.
- Obtaining a feel for your awareness and attitude towards software quality

This section is mainly for intermediate to senior level. If you are a beginner, skim through this section, and come back to it after reading the other sections. This section is mainly used to judge your experience. So where possible give practical examples by reflecting back on your past experience using the SAR (Situation-Action-Result) technique. The answers provided here are hints only. As an experienced professional, your experience and views may differ. Experienced professionals know what works and what doesn't. They know what tools to use to simplify a task at hand.

Good developers are opinionated based on their past experience. They can handle scenario based questions. They may pause a while to reflect back on their experience, but won't be stumped by the open ended questions. Reflecting back on your past experience and a little preparation prior to your interviews and meetings will really boost your performance.

This is a brush up section that will not only help you sell yourself more effectively, but also will help you reflect back on your past experience and refresh your memory. This section will add value only if you perform well relating to sections 4 – 7 on fundamentals.

This section will also provide valuable insights into getting your job done more effectively, and making yourself heard in team meetings by contributing suggestions or raising opinions. This does not mean that you will have to be inflexible. Most importantly, the good developers get things done by complimenting their technical skills with good soft-skills and right attitude.

Judging Experience

As explained earlier, open ended questions give you a great opportunity to prove your caliber and usefulness to the organization in a well-rounded manner. You have no control over what questions get asked in interviews, but you have control over what messages, skills, experience, and personal attributes you want to nicely get across to your prospective employers or interviewers without bragging, being too critical, or coming across inflexible. The open-ended questions give you the chance to make an impression. A very common open-ended question is – tell me about yourself? **OEQ**

Q1 Can you list some of the Java features you used recently? **LF**

A1 [Hint] JDK 5.0 (external version 5.0 and internal version 1.5.0) is a major feature release that largely centered on enhancing ease-of-development.

Features:

- **Generics:** provides compile-time type safety to operate on objects of various types and collections frameworks. It eliminates the tedious labor of casting. (JSR-14)
- **Enhanced for loop:** eliminates the monotonous labor and error-proneness of iterators and index variables when iterating over collections and arrays. (JSR-201)
- **Autoboxing** and **auto-unboxing:** eliminates the routine work of manual conversion between primitive data types such as int, long, etc and wrapper object types such as *Integer*, *Long*, etc. (JSR-201)
- **Type safe enums:** are flexible object oriented enumerated type facility that allows type safety without the verbosity and error-proneness of the “type safe enum pattern” that existed prior to JDK 5.0. (JSR-201)
- **Varargs:** eliminates the need of manually wrapping up argument list into an array when invoking methods that accept variable-length argument lists. (JSR-201)
- **Static import:** eliminates the need to qualify static members with class names without the shortcomings of the “constant interface” anti-pattern existed prior to JDK 5.0. (JSR-201)
- **Annotations:** let you avoid boilerplate code under many circumstances by enabling tools to generate it from annotations in the source code. This leads to “attribute oriented” (aka declarative) programming. This eliminates the need to maintain “side files” that must be kept up to date with changes in source files.

Knowing your way around Java

Instead the information can be maintained in a source file itself. This is also Java's answer to the XDoclet framework used prior to JDK 5.0 (JSR-175). Annotations allow you to add runtime metadata to classes, fields, and methods. Everything that can be done with marker or tag interfaces in earlier versions of Java can now be done with annotations at runtime using reflection.

Libraries:

- The **Collections** framework has been enhanced with support for generics, enhanced for loop, and autoboxing. Three new interfaces named *Queue*, *BlockingQueue*, and *ConcurrentMap* have been added. More efficient copy-on-write “*List*” and “*Set*” implementations have also been added.
- The Java API for XML Processing (**JAXP**) has been included in the JDK so it doesn't need to be downloaded separately. (JSR-206)
- The **concurrency utility** package `java.util.concurrent` was included as part of the standard edition 5.0, which previously had to be downloaded separately. This package provides a scalable, thread-safe, and high-performance building blocks for developing concurrent applications.

JDK 6.0 (external version 6.0 and internal version 1.6.0) release is centered on being Web Services friendly, mixing of Java with other scripting languages, and general enhancements to existing libraries.

- One of the most significant features of this release is support for the Java API for XML Web Services (JAX-WS), version 2.0. JAX-WS 2.0 includes Java Architecture for XML Binding (JAXB) 2.0 and SOAP with Attachments API for Java (SAAJ) 1.3. So Java SE 6.0 is SOA (Service Oriented Architecture) friendly, and Apache Axis framework is no longer required.
- Java SE 6.0 also includes a new framework and developer APIs to allow mixing of Java technology with scripting languages such as PHP, Python, JavaScript, and Ruby. This is very useful for prototyping, experimentation of an algorithm or function, and one off or ad hoc tasks like transforming a text file into a needed format, looking for a specific item in large log files, getting a subset of data from a database and packing it as a text file, etc. All of the above tasks can be done in a language like Java, but it can be faster and easier to do them in a scripting language like PERL, Ruby, or Python. **BP** It is a best practice to always use the right or best tool for the right job without just being overly passionate about Java alone.

- Full JDBC 4.0 (Java Data Base Connectivity) implementation providing improved XML support for Databases. A free to use Java Database based on Apache Derby is included since JDK SE 5.0 download.
- Improved desktop APIs including newly incorporated “*SwingWorker*” utility to help with multi-threading in GUI applications and much needed “*JTable*” sorting and filtering capabilities. You also have access to new things such as desktop applications through what used to be called the JDesktop Integration Components (JDIC). You can now have the ability to add applications to the system tray. The GUI performance is snappier and integrates well with the native platforms.
- Improved monitoring and management through out of the box on demand support tools and inclusion of Java heap analysis tool (Jhat) for forensic explorations of those core dumps.

Q&As 2 to 4 are skipped

Q5 Can you list the Java SE features you like to be added in the future? **LF**

A5 [Hint]

Closures: are useful for implementing abstractions that involve behavior. A closure is a named or anonymous subroutine. It looks like Java 7 will have closures. Closures can make your life as a developer much easier. Not sure of the actual syntax for Closures at this stage, but it could be something like:

For example, to run a function in a thread pool,

```
myThreadPool.submit(#() { for (int i = 1; i < 1000; i++) performTask(i) });
```

#() {...} --> is a function (i.e. an anonymous function)

#() --> empty paranthesis indicate that the function has no arguments.

Currently (up to Java 6), custom sorting of objects in a list requires an implementation of a *Comparator* interface. Closure could simplify this as follows to sort strings in a list by its length in an ascending order as shown here:

```
Collections.sort( listStrings, #(String str1, String str2) str1.length() - str2.length());
```

Knowing your way around Java

JavaScript is widely used in web development, and in JavaScript a closure is created every time you create a function within a function. When using a closure, you will have access to all the variables in the enclosing (i.e. the parent) function.

```
var calculate = function(x) {  
    var const = 2;  
    return function(y) {  
        return x + y + const; // has visibility to parent variable 'const'  
    };  
}  
  
var plus5 = calculate(5); //plus5 is now a closure  
alert(plus5(3));         //returns 10 i.e. x=5, y=3, const=2  
alert(plus5(7));         //returns 14 i.e. x=5, y=7, const=2  
alert(plus5(10));        //returns 17 i.e. x=5, y=10, const=2
```

Closures are first class objects in groovy, and in many ways it resembles anonymous inner classes in Java. The above functionality can be implemented in groovy as shown below:

```
def calculate = {x,y -> x + y + 2} //closure  
def plus5 = { calculate.call(5,it) } // "it" is like "this" in java. Here stands for  
                                     // values passed in to plus5 call like 3, 7 or 10  
  
println plus5.call(3)           //prints 10  
println plus5.call(7)           //prints 14  
println plus5.call(10)          //prints 17
```

Mixins: Java does not support multiple-inheritance, but it is possible to have the concept of "mixins inheritance", which are "partial classes" that you can bolt onto your class, giving you something very much like multiple inheritance. The concept of "mixins inheritances" are great when you want to refactor common functionality from two classes having different super classes. Currently Java does not have "mixins" like Ruby does. In many languages including Java, you cannot say,

```
class FileTreeNode extends File mixin TreeNode {...}
```

to give you access to all the operations provided by the *File* class to work with files and the *TreeNode* class to arrange files and perform binary searches at the same time.

You can achieve mixins in Java with the help of,

- Aspect Oriented Programing (AOP) language like AspectJ. AOP compliments OO.
- Using interfaces and annotations (or reflection, CGLIB, etc) to mix at build time using annotations to generate the mixin code as demonstrated below.

```
public interface TreeNode {  
    TreeNode search(String toSearch);  
    // ....other methods  
}
```

```
public class TreeNodeImpl implements TreeNode {  
    public TreeNode search(String toSearch) {... }  
    // ....other methods  
}
```

```
@compose ( type = 'TreeNode.class', field = "tree", implClass=  
                                                    "TreeNodeImpl.class"  
@parentClass (name = "java.io.File")  
public FileTreeNode extends FileTreeNodeGenerated implements TreeNode {  
    //...  
}
```

The *@compose* and *@parentClass* annotations will generate the class *FileTreeNodeGenerated* at build time as shown below, which is extended and used by the *FileTreeNode* class.

```
/**  
 * Generated class that achieves reuse of the File class through  
 * inheritance and the TreeNode class through composition. More  
 * interfaces can be composed if required.  
 */  
public class FileTreeNodeGenerated extends File {  
    private TreeNode tree; //composition  
  
    public FileTreeNodeGenerated(...) {  
        this.tree = new TreeNodeImpl(...);  
    }  
}
```

Knowing your way around Java

```
public TreeNode search(String toSearch) {  
    tree.search(toSearch);           //use 'TreeNode'  
}  
  
//... other methods  
}
```

Note: The above example is shown only to demonstrate mixins, and this may not be the ideal solution to a given problem.

You could also fake mixins in Java with the inner classes, but it would be ideal to have true mixins that are easier to use out of the box.

Java Module System: JAR files are hard to distribute, hard to version, and hard to reference in general. Existing JAR format can lead to classpath and class loading problems when:

- a developer or deployer of a Java application has accidentally made two different versions of a library available to the system.
- two different libraries (or a library and the application) require different versions of the same third library. If both versions of the third library use the same class names, there is no way to load both versions of the third library with the same classloader.
- classes loaded by different class loaders may interact in complex ways not fully understood by a developer, leading to inexplicable errors or bugs.

The Java Module System (JSR 277) was initiated to address some of the issues caused by the JAR files. The Java Module System defines a distribution format and a repository for collections of Java code and related resources. It also defines the discovery, loading, and integrity mechanisms at runtime. It looks like Java 7 will have the Java Module System. Until then the OSGi can handle all of these nicely. The OSGi (Open Services Gateway initiative), which is also known as the dynamic module system for Java, defines an architecture for modular application development. The OSGi implementations such as Apache Felix, Equinox, and Knoplerfish allow you to break your

application into multiple modules, and thus allow you to better manage the cross dependencies between those modules

Add some of the frequently used third party libraries like “Apache commons collections”, “Apache commons lang”, and more intuitive and easy to use libraries like “Joda-Time” to the Java core API (Joda-Time may become part of Java SE 7.0).

You may also think of some of the features in other languages that you liked and keen to have those features added to Java if not already present in Java.

For example, multi-line string literal in both Scala and Groovy as shown below is more readable and does not require special escape characters as in Java.

```
String multiLine = """
    Line 1
    Line 2 "no escape"
    Line 3""";
```

The syntax in Java is less readable,

```
String multiLine = "Line 1" +
    "Line 2 \"escape\"" +
    "Line 3";
```

The Java POJOs (Plain Old Java Objects) verbose and less readable with all the getter and setter methods. Groovy comes to the rescue. Groovy makes POGOs (Plain Old Groovy Objects) less verbose and more readable by creating the getters/setters during byte code generation. In a POGO, if you have an access modifier, it will be treated as a property and if you don't specify an access modifier it will be a normal field. The following code shows Groovy and Java code working together by using the -j compiler switch to enable the joint compilation.

The less verbose POGO in Groovy is as shown below:

```
public class Person {

    String name                //property
    String lastName            //property
    private int id              //a normal field
```

Knowing your way around Java

```
String toString() {  
    return "I am ${name} ${lastName}."  
}  
}
```

The invoking code in Java is as shown below

```
public class Organization {  
  
    public static void main( String args[] ) {  
        Person person = new Person();  
        person.setName( "Peter" );  
        person.setLastName( "Smith" );  
        System.out.println( person.toString() );  
    }  
}
```

Finally, you may prefer fixing or improving the existing limitations and gotchas in Java before adding any new features as it could make things worse. Most of these pitfalls are brought about by the effort to maintain the backward compatibility.

Note: Being able to compare strengths and weaknesses of Java with other programming and scripting languages will demonstrate some of the recognizable qualities of good programmers. Good programmers get excited chatting about technologies, and are passionate about diversifying on the technology stack. So learn different technologies, frameworks, and tools, and be opinionated about which are better for various scenarios.

Q&As 6 to 11 are skipped

Gauging Your Experience with UNIX

Most production systems are run in a Unix environment. The Unix environment is very robust, and offers powerful tools to automate tasks. You might be asked a scenario based question as shown below:

Q. How would you go about replacing a piece of a text or a phrase from 20,000+ web

templates residing on a Unix file system? **SBQ**

Even some candidates with 5+ years of experience will be tempted to spend a day or two to write 200+ lines of code to achieve the above requirement, when it can be achieved in 5 minutes with a simple Unix command. Some of the Unix commands like `grep`, `find`, `awk`, `sed`, etc along with the power of regular expressions can make you more productive.

Q12 Can you list some of the “UNIX” commands Java developers use very frequently?
PF

A12 Find command: allows a Unix user to process a set of files and/or directories in a file sub tree. Examples,

- Recursively delete `.svn` directories: Subversion is a widely used open-source revision control application. Every copy of source code received from subversion repository has `.svn` folders, which store metadata. However, if you want to use or distribute source code, these `.svn` folders are often not necessary. When creating a new subversion project (or folder) based on an existing subversion project, it is imperative that the `.svn` folders are deleted before checking in the copied and subsequently modified project. Otherwise you run the risk of corrupting the original subversion project with the newly copied and modified project.

```
rm -rf `find . -type d -name .svn`
```

- Search for a file with a specific name in a set of files in the current (i.e. “.”) directory.

```
find . -name "rc.conf" -print
```

- Search for “*.java” files that are using “`java.util.concurrent`” classes.

```
find . -type f -name "*.java" -exec grep "java.util.concurrent" {} \; -print  
find . -type f -name "*.java" | xargs -n1 grep "java.util.concurrent"
```

- Searching for old files. Finding files that are 7 days old or finding a file that has not been accessed for 30 days more, and files that are larger than 100k:

Knowing your way around Java

```
find . -mtime 7 -print
find . -type f -atime +30 -print
find . -type f -size +100k -ls
```

The `grep` and `egrep` commands are very handy for tearing through text files and finding exactly what you want very quickly.

- Search the log files to see if a particular user logged in.

```
cat /var/log/server.log | grep "login.jsp" | grep "userid"
```

- List all the log files that have a particular search text.

```
grep -l "Missing Characteristic Value" *.log*
```

- You can use regular expression patterns to search.

```
grep -e '^import.*util.regex' *.java
```

- Find all occurrences of `util.*` packages except the `util.regex` package.

```
grep -e '^import.*util' *.java | grep -v 'regex'
```

Tip: The Java *LinkageError* is tricky to resolve and it indicates that a class has some dependency on another class; however, the latter class has incompatibly changed after the compilation of the former class. Since plethora of third party jar files are used in commercial applications, it is not uncommon to get an error like,

```
java.lang.LinkageError: Class javax/xml/namespace/QName violates loader
constraints
```

This means the class *javax.xml.namespace.QName* is loaded by two or more different class loaders, and the versions are not compatible. For example, one might be under `WEB-INF/lib`, another might be loaded through an EJB or from a server lib folder. The “find” and “grep” Unix command can come to the rescue to identify the offending jars as shown below:

```
$ find . -name "*.jar" -print | -exec jar tvf {} \; |  
grep -E "jar$|javax/xml/namespace/QName\.class"
```

Remaining discussions are skipped ...

Q13 Can you give some examples of where you used “sed” and “awk” utilities? **PF**
A13

Q14 Can you answer the following Unix questions? **PF**

Q. Can you explain the following Unix command?

```
$ date ; ps -ef | awk '{print $1}' | sed -e '1d' | sort | uniq | wc -l >> activity.log
```

A. Skipped....

Q. How will you list the top 5 largest files or directories?

A. Skipped....

Q. How will you find out the disk free space?

A. Skipped....

More Q&A Skipped....

Q15 Can you brief on a shell script that you had worked on? **OEQ**

A15 Skipped....

Exposure to tools, technologies, and frameworks

Using the right tools, technologies, and frameworks for the right job can not only make developers more productive, but also can significantly increase the robustness and quality of the applications built. There are so many free online resources to master the tools if you know what tools to use and when to use them. Tools will not only help you become more productive, but also enable you to impress your superiors and peers by delivering your work on time without compromising on the quality.

Knowing your way around Java

Q16 Can you name some tools, technologies, or frameworks you had to use in Java for source code generation, byte code manipulation, or assembling code at runtime? **PF**

A16 Skipped....

Q17 What tools do you generally need to get your job done? **OEQ FAQ**

A17 Skipped....

Q18 What other languages have you used? **LF**

A18 Skipped....

Q19 How would you go about.....? questions like

- How would you go about performance tuning an application?
- How would you go about identifying memory leaks or thread safety issues in an application?
- How would you go about gathering requirements, designing, and documenting your applications?
- How would you go about identifying and fixing any potential transactional issues in your application?
- How would you go about determining the security requirements for your application?
- How would you go about describing the software development process you are familiar with?

A19 These questions have been addressed in the “Java/J2EE Job Interview Companion (400+ Q&A)”.

Documenting your Java applications

Good documentation is vital to any software project. As your documentation will be reviewed by your superiors and peers, a good documentation will also make a good impression and can surely help you progress in your career or get your contracts extended.

Q20 What are the different types of documents you refer to while designing or coding your system? **DC OEQ FAQ**

A20 Skipped....

Q21 In your experience, why is it important to have relevant APIs handy while coding? **COQ** **LF**

A21 Skipped....

Q22 What do you expect to see in a high level or detailed technical specification document? **DC** **OEQ**

A22 Skipped....

Q23 What do you expect to see in a high level or detailed business requirements document? **OEQ**

A23 Skipped....

Q24 What tips do you give to someone who wants to write a technical documentation?

A24 Skipped....

Ensuring code quality.

As a software engineer, you should ensure how your users will have quality experience with the system you build. You must learn not only to think from a technology perspective, but also from a user experience and business objective perspective. No point in building something with a bleeding edge technology, if it does not add real value to your business and their users. **QOS**

Q25 Do you use test driven development? Why / Why not? **COQ** **SP**

A25 [Hint:] Yes.

- Gives you a better understanding of what you're going to write. Gets you to clearly think what the inputs are and what the output is. Helps you separate the concerns by getting you to think about the single responsibility principle (SRP).
- Enforces a better test coverage. This gives you the confidence to refactor your code in the future, since you have a good coverage.

Knowing your way around Java

- You won't waste time writing features you don't need.

Q26 What tips do you give someone who is writing the unit testing? **COQ SP**

A26 Skipped....

Q27 How will you ensure quality of your design and code? **COQ BP DC DP**

A27 Skipped....

Q28 Can you list some of the key aspects you keep in mind while coding to ensure quality & robustness? **COQ BP DC DP**

A28 Skipped....

Q29 What are mock objects and when do you use them? **COQ**

A29 Skipped....

Q30 What development methodologies are you comfortable with? **SP**

A30 Skipped....

**"In theory there is no difference between theory and practice.
In practice there is."**

– YogiBerra

Section-4:

Language Essentials

- Valid identifiers and reserved keywords
- Choosing the right data types
- Widening versus narrowing conversions
- Understanding the operator precedence
- Thinking in binary and bitwise operations
- Initializing your data correctly
- Packaging your classes to avoid conflicts
- Constants and static imports
- Modifiers and where are they applicable?
- Methods and constructors
- Recursive functions
- It is always pass-by-value in Java
- Class, instance, and local variables

This section is mainly for beginner to intermediate level. It also contains useful information for senior level candidates, and worth having a quick browse through, especially if you are required to sit for a basic Java technical test. Even among experienced professionals, there are a breed of developers who know how to get things done by using various frameworks, tools, and googling without fully understanding the fundamentals. Understanding the core Java fundamentals are essential not only from the point of view of passing the technical tests used for screening potential candidates and to perform well in technical job interviews, but also from the point of view of showing off your technical capabilities within limits to your peers and superiors during code review sessions, team meetings, and situations where you were able to resolve an intermittent, critical, or obscure problem more quickly than others as you understand the nitty-gritty details.

It's important to let others know about the good things you are accomplishing. Don't think that by just working hard that you'll get noticed. You need to let people know,

Language Essentials

especially your leaders like to know these things because they can't know everything that is going on, so speak up when required to. A little preparation prior to a code review session or an important team meeting that was going to discuss “a class loading issue” can make a huge difference in making an impact. All you have to do is refresh your memory on the basics.

Preview

Valid identifiers and reserved keywords

Q1 What are identifiers in Java? What does the following code do? Can you talk us through the code highlighting some of the key language and design features? **COQ**

LF BP

```
package chapter2.com;

import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public final class ValidIdentifiers {

    private enum Validity {
        Valid, InvalidIdentifierStart, InvalidIdentifierPart, ReservedKeyWord,
        ReservedLiteral
    };

    private static final String[] RESERVED_KEYWORDS = { "abstract",
        "continue", "for", "new", "switch", "assert", "default",
        "if", "package", "synchronized", "boolean", "do", "goto",
        "private", "this", "break", "double", "implements",
        "protected", "throw", "byte", "else", "import", "public",
        "throws", "case", "enum", "instanceof", "return",
        "transient", "catch", "extends", "int", "short", "try",
        "char", "final", "interface", "static", "void", "class",
        "finally", "long", "strictfp", "volatile", "const",
        "float", "native", "super", "while" };

    private static final String[] RESERVED_LITERALS = {"true", "false", "null"};

    private static Set<String> KEYWORDS = new HashSet<String>(  
        (int)(RESERVED_KEYWORDS.length/0.75));

    private static Set<String> LITERALS = new HashSet<String>(  
        (int)(RESERVED_LITERALS.length/0.75));
```

Language Essentials

```
static {
    List<String> list = Arrays.asList(REERVED_KEYWORDS);
    KEYWORDS = new HashSet<String>(list);

    List<String> listLit = Arrays.asList(REERVED_LITERALS);
    LITERALS = new HashSet<String>(listLit);
}

public static final Validity valid(String input) {
    if (input.length() == 0
        || !Character.isJavaIdentifierStart(input.charAt(0))) {
        return Validity.InvalidIdentifierStart;
    }

    for (int i = 1; i < input.length(); i++) {
        if (!Character.isJavaIdentifierPart(input.charAt(i))) {
            return Validity.InvalidIdentifierPart;
        }
    }

    if (KEYWORDS.contains(input)) {
        return Validity.ReservedKeyWord;
    }

    if (LITERALS.contains(input)) {
        return Validity.ReservedLiteral;
    }

    return Validity.Valid;
}
```

A1 Identifiers are names you give to your variables, constants, classes, interfaces and methods.

The above code verifies if a given input is a valid identifier, complying with the following rules:

- The first character of an identifier must be a letter, an underscore(_), or a currency sign(e.g. \$).

- The rest of the characters in the identifier can be a letter, underscore, currency sign, or digit. Note that spaces are NOT allowed in identifiers.
- Identifiers are case-sensitive. This means that *age* and *Age* are considered as different identifiers.
- Identifiers cannot match any of Java's reserved words like `for`, `int`, etc or literals like `null`, `true`, and `false`..

Note: `true`, `false`, and `null` might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

Note: `const` and `goto` are reserved, but not currently used. `enum` was added in Java 5. `strictfp` allows you to have more predictable control over floating-point arithmetic.

Q. Can you talk us through the code highlighting some of the key language and design features?

A.

- Use of `enums` and `generics` indicates that this code must be using JDK version 1.5 or greater.
- **LF** Private access modifiers are used where required to encapsulate the internal details.
- **LF** The class is marked `final` so that it cannot be extended.
- **DC** It follows the “**code to interface**” design principle. For example, the `Set`, `List`, etc shown below are interfaces.

```
private static Set<String> KEYWORDS = new HashSet<String>(  
    (int)(RESERVED_KEYWORDS.length/0.75));  
//...  
List<String> list = Arrays.asList(RESERVED_KEYWORDS);
```

- **BP** Making use of the Java API methods where possible. For example, the methods shown below from the *Character* and *Arrays* classes, simplify your code. So don't memorize the Java API, but keep it handy and constantly refer to it.

Language Essentials

```
Character.isJavaIdentifierStart(input.charAt(0))
Character.isJavaIdentifierPart(input.charAt(1))
List<String> list = Arrays.asList(REERVED_KEYWORDS);
```

- **BP** If a size of a collection is known in advance, it is a best practice to set its initial size appropriately to prevent any resizing. Implementing the code as shown below would not quite work.

```
private static Set<String> KEYWORDS = new HashSet<String>(
    RESERVED_KEYWORDS.length);
```

The internal threshold for *HashSets* and *HashMaps* are calculated as $(\text{int})(\text{capacity} * \text{loadFactor})$. The default `loadFactor` is 0.75. This means the *HashSet* will resize after 75% of its capacity has been reached. The resizing and rehashing of the set can be prevented as follows,

```
private static Set<String> KEYWORDS = new HashSet<String>(
    (int)(RESERVED_KEYWORDS.length/0.75));
```

- **BP**: Checking for null and empty string as a **precondition** in the beginning of the *valid(String input)* method.

```
if (input == null || input.length() == 0 ...)
```

The above code snippet is a slight deviation of the fail fast principle, which states that check and report any possible failures as soon as possible. Testing your code for failure points will lead to better, safer, and more reliable code.

Q. Do you have any recommendations to further improve the code? **OEQ**

A. Yes.

- The Apache commons library class *StringUtils* can be introduced here. The method *isEmpty (String input)* can be used to check for both null and empty string. This library does have other useful methods that can be used elsewhere in the application code to enforce the fail fast principle.

```
StringUtils.isEmpty(input)
```


- The `RESERVED_KEYWORDS` and `RESERVED_LITERALS` constants may be loaded from a configuration file. This will ensure that if new keywords or literals are added to Java in the future, it will require only a configuration change.

Note: Even though the above recommendations are debatable depending on various other factors, it is better to know the possible options to demonstrate your experience than to have no ideas at all.

Q2 Which of these are legal Identifiers? **LF**

- a) \$Ident1
- b) _Ident1
- c) -Ident1
- d) 2Ident1
- e) private
- f) private1
- g) null
- h) Ident-1
- i) Ident\$1
- j) \u00A3Ident1
- k) \u00A5Ident1

A2 Skipped....

Q3 Which of the following are keywords in Java? **LF**

- a) friend
- b) NULL
- c) implement
- d) synchronized
- e) throws
- f) implements
- g) synchronize
- h) volatile
- i) transient
- j) native
- k) interface

Language Essentials

- l) TRUE
- m) new
- n) true
- o) strictfp
- p) instanceof
- q) then
- r) throw
- s) this

A3 The keywords are: d, e, f, h, i, j, m, o, p, r, s.

Note: *true* is a reserved literal not a reserved keyword. Also, take care with the spelling. *implements* ending with an 's' is a keyword, but *implement* is not. *synchronized* ending with 'ed' is a keyword, but *synchronize* is not. Unlike other languages, *then* is not a keyword. You only have *if* and *else* in Java. Both *throw* and *throws* are keywords.

Q4 Why is it a best practice to adhere to naming conventions and what are they? **BP**

A4 Skipped....

Choosing the right data types

Some simple mistakes like having an integer overflow can bring an application down and cost companies in thousands.

Q5 How would you go about choosing the right data types for your application? What are wrapper classes, and why do you need them? **LF BP FAQ**

A5 Skipped....

Q6 When working with floating-point data types, what are some of the key considerations? **LF BP COQ FAQ**

A6 Skipped....

Widening versus narrowing conversions

Q7 What is your understanding of widening versus narrowing conversions of primitive data types? Can you explain explicit and implicit casting? **LF COQ FAQ**

A7 Skipped....

Q8 What are the dangers of explicit casting? **LF COQ FAQ**

A8 Not knowing the MIN and MAX values can result in unexpected results due to loss of data during **narrowing**.

```
package chapter2.com;

public class DataTypes2 {
    /**
     * Trap #1 Be careful when casting explicitly.
     *
     * Not knowing the correct range (i.e Min and Max) can cause
     * unexpected results as illustrated
     */
    public static void main(String[] args) {
        int iWithinByteRange = 125;
        int iOutsideByteRangeMax = 129;
        int iOutsideByteRangeMin = -129;

        byte bGood = (byte) iWithinByteRange;
        System.out.println("bOkay=" + bGood);           // 125 – good

        byte bBad = (byte) iOutsideByteRangeMax;
        System.out.println("Trap #1 - bBad=" + bBad);     // -127 – bad

        byte bBadAgain = (byte) iOutsideByteRangeMin;
        System.out.println("Trap #1 - bBadAgain=" + bBad); // -127 – bad
    }
}
```

Q9 Can you explain why the code snippet under (a) gives a compile-time error and code snippet under (b) compiles and runs with an output of 20? **COQ LF**

a)

Language Essentials

```
byte b = 10;           // line 1  
b = b + 10;           // line 2
```

b)

```
byte b = 10;           //line 1  
b+=10;                //line 2
```

A9 Skipped....

Q10 What is wrong with the following code, and how will you fix it? **LF COQ**

```
package chapter2.com;  
  
public class IntegerMaxValue {  
  
    public int addFive(int input) {  
        if (input + 5 > Integer.MAX_VALUE) {  
            throw new IllegalArgumentException("Invalid input: " + input);  
        }  
        return input + 5;  
    }  
  
    public static void main(String[] args) {  
        IntegerMaxValue imv = new IntegerMaxValue();  
        System.out.println("value = " + imv.addFive(5));  
    }  
}
```

A10 Skipped....

Understanding the operator precedence

Q11 What do you understand by operator precedence in Java? **LF**

A11 Skipped....

Q12 Do you think the following code will throw a compile-time exception? If yes, how will you fix it? **LF**

```
float myVal = (float)3.0/2.0;
```

A12 Yes. It is cast first and then divided as casting operator has precedence over division operator as per the precedence table. So the above code is equivalent to

```
float myVal = 3.0f/2.0; // float divided by double returns a double
//as per the binary numeric promotion principle
```

To fix it, you need to get the division operator to evaluate prior to casting. You can achieve this by introducing a parenthesis around the division as parenthesis has higher precedence (in fact highest) than casting as per the precedence table.

```
float myVal = (float)(3.0/2.0); // double divided by double returns a
// double and it is then explicitly
// cast to float to return a float value.
```

Q13 What is the output of the following code snippet? **LF COQ**

```
package chapter2.com;

public class PrePostOperators {

    public static void main(String[ ] args) {
        int x = 5; // line 1
        int y = ++x; // line 2
        int z = y++; // line 3

        System.out.println("x=" + x + ", y=" + y + ", z=" + z);
    }
}
```

A13 Skipped....

Q14 What is the output of the following code snippet? **LF**

```
package chapter2.com;
```

Language Essentials

```
public class NaN {  
    public static void main(String[] args) {  
        System.out.println("Output=" + 2.0/0.0); //floating-point division by zero  
        System.out.println("Output=" + -2.0/0.0); //floating-point division by zero  
        System.out.println("Output=" + 2 /0); //integer division by zero  
    }  
}
```

A14 Skipped....

Q15 What will be the output of following operations?

LF

```
int i = 10 / 3;  
int r = 10 % 3;
```

A15 Skipped....

•

Thinking in binary and bitwise operations

Even though there is rarely a time bitwise operations seem directly necessary, the standard library uses bitwise operations indirectly for efficient processing. For example, the *StringBuffer.reverse()*, *Integer.toString()*, *BigDecimal* and *BigInteger* classes to name a few. There are number of practical examples listed where bitwise operations are very handy. Some interviewers prefer asking questions on this topic or including it in the written test to determine how technical you are. If pressed for time, either scan through or skip this topic.

Q16 Can you convert the decimal value of 127 to binary, octal, and hexadecimal values?

LF

A16 **Binary:**

Bit number	8	7	6	5	4	3	2	1
Binary has 2 possible values 0 (off) and 1(on).	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Expanded	128	64	32	16	8	4	2	1
Bits that need to be on (i.e. 1)	0	1	1	1	1	1	1	1

to add up to 127 are								
----------------------	--	--	--	--	--	--	--	--

$64 + 32 + 16 + 8 + 4 + 2 + 1 = 127$. Hence the binary value is **0111 1111**.

Octal:

Binary value for 127 is	0	1	1	1	1	1	1	1
Octal has 8 possible values 0, 1, 2, 3, 4, 5, 6, 7	8^2		8^1			8^0		
Represent the octal values in binary	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0
Expanded, and the bits that are “turned on” (i.e. 1's) highlighted. Shaded values add up to the octal values shown below.	2	1	4	2	1	4	2	1
Octal representation of 127.	1		7			7		

The octal value is 0177. (First zero denotes octal representation). You can verify your result by converting this back to decimal as follows:

$$1 \times 8^2 + 7 \times 8^1 + 7 \times 8^0 = 64 + 56 + 7 = 127.$$

Hexadecimal:

Binary value for 127 is	0	1	1	1	1	1	1	1
Hexadecimal has 16 possible values from 0 to 15 denoted by 0,1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (i.e. A is 10, B is 11 ... and F is 15)	16^1				16^0			
Represent the hexadecimal values in binary	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
Expanded, and the bits that are “turned on” (i.e. 1's) highlighted. Shaded values	8	4	2	1	8	4	2	1

Language Essentials

add up to the hexadecimal values shown below.								
Hexadecimal representation of 127.	7				F			

The hexadecimal value is 0x7F. (0x denotes hexadecimal). You can verify your result by converting this back to decimal as follows:

$$7 \times 16^1 + 15 \times 16^0 = 112 + 15 = 127.$$

Q. How will you convert 0x7F back to binary? L.F

Hexadecimal value	7				F			
Value	7				15			
Shaded values add up to the value above	8	4	2	1	8	4	2	1
	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
	0	1	1	1	1	1	1	1

The binary value for 0x7F is **0111 111**. Here is a sample Java program.

```
package chapter2.com;

public class Binary1 {

    public static void main(String[] args) {
        Byte bOctal = 0177;           // decimal value 127
        Byte bHexadecimal = 0x7F;     // decimal value 127

        System.out.println("decimal value of bOctal = "
            + bOctal.intValue());      //127
        System.out.println("decimal value of bHexadecimal = "
            + bHexadecimal.intValue()); // 127

        System.out.println("binary value of bOctal = " +
            Integer.toBinaryString(bOctal)); //1111111
        System.out.println("binary value of bHexadecimal ="
```



```

        + Integer.toBinaryString(bHexadecimal));    //1111111
    }
}

```

Q17 How do you represent negative numbers in Java? What is the negative binary value for the number 37? **LF**

A17 Skipped....

Q18 What are the common bitwise operations? **LF**

A18 Skipped....

Q19 Can you list some practical applications where the bitwise operations can be applied? **LF COQ**

A19 Skipped....

Initializing your data correctly

Q20 What would happen when the following code is compiled and run? **LF COQ**

```

package chapter2.com;

public class VariableInitialization2 {

    public static void main(String[] args) {
        int x = 10, y;
        if (x < 10) {
            y = 1;
        }
        if (x >= 10) {
            y = 2;
        }
        System.out.println("y is " + y);
    }
}

```

A20 Skipped....

Language Essentials

Q21 Can you tell me what is the use of a static block in a class with an example? Is it a good practice to use static blocks? **LF BP COQ**

A21 Skipped....

Q22 How will you provide a member variable named *dueDate* with an initial default value set to first day of the following month? **LF COQ**

A22 Skipped....

Q23 Is there anything wrong with the following code snippet? **COQ**

```
package chapter2.com;

public class VariableInitialization3 {

    private static final String CONSTANT_VALUE = null;           // line A

    static {
        // ... load values based on some condition or .properties file.
        CONSTANT_VALUE = "loaded";                               // line B
    }
}
```

A23 Yes. Compile-time error at line B.

```
CONSTANT_VALUE = "loaded";
```

indicating that the final field `VariableInitialization3.CONSTANT_VALUE` cannot be assigned.

Q. How will you fix it?

A. Remove the initial null assignment in line A. Final variables can only be assigned once.

```
private static final String CONSTANT_VALUE;           // fixed
```

Constants and static imports

Q25 What is a constant interface anti-pattern? How will you avoid it? **DP**

A25 Skipped....

Modifiers and where are they applicable?

Q26 Why use modifiers? Why can't you declare a class as both abstract and final? **LF**

FAQ

A26 Skipped....

Q27 What are the valid access modifiers? **LF FAQ**

A27 Public, protected, package-private (i.e. no modifier), and private.

Q28 Discuss the significance of the modifiers used in the following class? **LF COQ**

```
package chapter2.com;

final class MyConsts {

    private MyConsts(){ }

    static final double TRIANGLE_AREA_PREFIX = 1.0/2.0;
    static final double SPHERE_VOLUME_PREFIX = 4.0/3.0;
}
```

A28 Skipped....

Q29 If you want to extend the *java.lang.String* class, what methods will you override in your extending class? **LF**

A29 Skipped....

Q30 Are 'volatile' and 'const' valid modifiers in Java? **LF FAQ**

A30 Skipped....

Q31 How would you go about determining what access levels to use in your code? **BP**

FAQ

A31 Skipped....

Q32 If you were to give some tips on modifiers, what would they be? **LF**

A32 Skipped....

Methods and constructors

Q33 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? Are constructors inherited? How do you call a super class's constructor?

LF FAQ

A33 Skipped....

Q34 Where and how can you use a private constructor? **LF DP FAQ**

A34 Skipped....

Q35 What are static factory methods? Can you give examples from the Java API? What design patterns does it use? Can you describe the benefits of static factory methods and usefulness of private constructors? **LF DP FAQ**

A35 *Skipped....*

Q35 ... Q49 are skipped

Section-5:

Classes and Interfaces Essentials

- Working with classes and interfaces
- subclassing, overriding, and hiding
- Designing your classes and interfaces
- Working with abstract classes and interfaces
- Inheritance versus composition
- Applying the design principles
- Class invariant and design by contract
- Working with inner classes
- Packaging your classes to avoid conflicts

This section is for all. If you asked me to pick a section that is most popular with the interviewers, this is it. If you don't perform well in this section, your success rate in interviews will be very low. Good interviewers will be getting you to analyze or code for a particular scenario. They will be observing your decisions with interfaces and classes, and question your decisions to ascertain your technical skills, analytical skills, and communication skills. You can't memorize your answers. This section requires some level of experience to fully understand. It has enough examples for beginners to get some level of familiarization.

Be ready to be asked about some tricky questions around the topics discussed in this section. Java interview questions are meant to analyze your technical bent of mind. You have to tell the way you are going to solve a particular problem with Java or any other programming language of your choice. So the key is to ask the right questions and then apply what you had learned. Keep practicing the examples provided here, and experiment with them until you get a good grasp.

Working with classes and interfaces

Object oriented (i.e. OO) concepts are an important building for creating good services. They provide a common vocabulary to talk about the architecture. Without good understanding of the OO concepts, you could easily implement a procedural programming using classes. All you have to do is create a bunch of classes with nothing but data with getter and setter methods, and some other *Handler* or *Processor* classes to handle the logic with *if/else* and *instanceof* constructs sprayed all over the place. It is important to understand that objects need to not only take care of themselves with data and behavior, but also have to interact with each other.

Q1 Which class declaration is correct if *A* and *B* are classes and *C* and *D* are interfaces?

LF

- a.) `class Z extends A implements C, D { }`
- b.) `class Z extends A,B implements D { }`
- c.) `class Z extends C implements A,B { }`
- d.) `class Z extends C,D implements B { }`

A1 **a. class Z extends A implements C, D { }**

A class is a template. A class can extend only a single class (i.e. single inheritance. Java does not support **multiple implementation inheritance**), but can implement multiple interfaces to achieve **multiple interface inheritance**. An interface can also extend more than one other interfaces.

```
interface E extends C,D { //.... }
```

Q2 What is a class? What are the valid modifiers of a top level class? **LF**

A2 A class is a template for multiple objects with similar features. In another words, A class defines responsibilities (i.e. characteristics and behaviors) that are common to every object.

Modifiers: A top level class can either be public or package-private (i.e. no access modifier) scoped. It can be final concrete, abstract, or non-final concrete (i.e. no final or abstract modifiers).

```
public class A { ... } // non-final concrete class
```

Classes and Interfaces Essentials

```
class A { ... }           // non-final concrete class
public abstract class A { ... } // abstract class
public final class A { ... } // final concrete class
```

Subclassing, overriding, and hiding (aka shadowing)

Q3 What is a subclass? **LF**

A3 A subclass is a class that extends a class. A subclass inherits all of the public and protected members of its parent class, no matter what package the subclass is in. If the subclass is in the same package as its parent, it also inherits the package-private (i.e. no access modifier) scoped members of the parent.

Q4 What can you do in a subclass? **LF FAQ**

A4 You can supplement, override, inherit, and hide/shadow your superclass members.

Supplement:

- You can declare new fields in the subclass that are not in the superclass.
- You can declare new methods in the subclass that are not in the superclass.
- You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by explicitly using the keyword “super(...)”.

Override:

- You can write a new instance method in the subclass that has the same signature as the one in the superclass, thus overriding it. The ability to override methods allows you to take advantage of **runtime polymorphism**.

Inherit:

- You can inherit commonly used state and behavior in the super class. One of the benefits of **implementation inheritance** is to minimize the amount of duplicate code in an application by reusing code from the superclass. Since Java does not support multiple implementation inheritance, it becomes a card that can be played only once, and thus it should be used with caution.

Shadow/hide:

- You can declare a field in the subclass with the same name as the one in the superclass, thus hiding it (not recommended).

- You can write a new static method in the subclass that has the same signature as the one in the superclass, thus hiding it (not recommended).

Q5 What is the output of the following code snippet? Give your reasons and recommendations? **COQ**

```
package subclass1;

public abstract class Animal {

    String name = "animal";

    public String getName(){
        return this.name;
    }
}
```

```
package subclass1;

public class Cat extends Animal {

    String name = "cat";

    public String getName(){
        return this.name;
    }
}
```

```
package subclass1;

public class Example {

    public static void main(String[] args) {
        Animal animal = new Cat();
        Cat cat = new Cat();

        System.out.println(animal.name);
        System.out.println(cat.name);
        System.out.println(((Cat)animal).name);
    }
}
```

Classes and Interfaces Essentials

```
System.out.println(((Animal)cat).name);

System.out.println(animal.getName());
System.out.println(cat.getName());
}
}
```

A5 Skipped....

Q6 What happens when a parent class and a child class each have a **static** method with the same signature? **LF**

A6 Skipped....

Designing your classes and interfaces

You need to have a good understanding of the OO concepts, design principles, and certain amount of common sense to design a quality system. Any one can recite the definition of the OO concepts and design principles, but only a few can come up with good class designs for a given problem. It is important to understand when to use an attribute versus a subclass, and when to use a composition or aggregation as opposed to inheritance. For example, if a *Dog* is a class extending an abstract class *Animal*, how would you design a *Dog* with a mole? As a subclass or as an attribute? If you have a class *Animal*, how would you define its body parts? As a subclass of animal or composition? [Hint: define mole as an attribute like *special-Marks*, and define the body parts as a composition]. A mole won't make a *Dog* to be more specialized. A dog with a mole does not have a **more specialized behavior** compared to dogs without a mole. The body parts of a dog don't have an “is a” relationship with the *Dog*. You can't say that a leg “**is a**” dog. You will have to say a dog has legs. This is a “**has a**” relationship denoting a composition or aggregation. There are more to it, and you will have a better understanding after going through the following Q&As with lots of examples.

Q7 How do you know that your classes are badly designed? **DC OEQ**

A7

- If your application is **fragile** – when making a change, unexpected parts of the application can break.
- If your application is **rigid** – it is hard to change one part of the application without affecting too many other parts.
- If your application is **immobile** – it is hard to reuse the code in another

application because it cannot be separated.

Overly complex design is as bad as no design at all. Get the granularity of your classes and objects right without overly complicating them. Don't apply too many patterns and principles to a simple problem. Apply them only when they are adequate. Don't anticipate changes in requirements ahead of time. Preparing for future changes can easily lead to overly complex designs. Focus on writing code that is not only easy to understand, but also flexible enough so that it is easy to change if the requirements change.

Q8 Can you explain if the following classes are badly designed? **OEQ SBQ**

The following snippets design the classes & interfaces for the following scenario. Bob, and Jane work for a restaurant. Bob works as manager and a waiter. Jane works as a waitress. A waiter's behavior is to take customer orders and a manager's behavior is to manage employees.

```
package badrestaurant;
```

```
public interface Person {}
```

```
package badrestaurant;
```

```
public interface Manager extends Person {  
    public void managePeople();  
}
```

```
package badrestaurant;
```

```
public interface Waiter extends Person {  
    public void takeOrders();  
}
```

```
package badrestaurant;
```

```
public class Bob implements Manager, Waiter {  
  
    @Override  
    public void managePeople() {
```

Classes and Interfaces Essentials

```
//implementation goes here
}

@Override
public void takeOrders() {
    //implementation goes here
}
}
```

```
package badrestaurant;

public class Jane implements Waiter {

    @Override
    public List<String> takeOrders() {
        //implementation goes here
    }
}
```

The *Restaurant* class uses the above classes as shown below.

```
package badrestaurant;

public class Restaurant {

    public static void main(String[] args) {

        Bob bob = new Bob();
        bob.managePeople();
        bob.takeOrders();

        Jane jane = new Jane();
        jane.takeOrders();
    }
}
```

A8 Skipped....

Q9 What do you achieve through good class and interface design? **DC OEQ**

A9 Skipped....

Q10 What are the principles of class design used in regards to OOD? **DC**

A10 Skipped....

Questions 11 ... 26 are skipped....

Applying the design principles

Q27 How will you improve on the following code snippets with appropriate design principles? **DC COQ SBQ**

```
package principle_srp1;

import java.sql.Connection;
import java.sql.SQLException;
import org.apache.commons.lang.StringUtils;

public class Animal {

    private Integer id;
    private String name;
    private Connection con = null;

    //getters and setters for above attributes go here..

    public boolean validate() {
        return id != null && id > 0 && StringUtils.isNotBlank(name);
    }

    public void saveAnimal() throws SQLException {
        //save Animal to database using SQL
        //goes here ...
    }

    public Animal readAnimal() throws SQLException {
        //read Animal from database using SQL
    }
}
```

Classes and Interfaces Essentials

```
//...
Animal animal = null;
//...
return animal;
}
}
```

A27 The above class represents 3 different responsibilities.

- Uniquely identifies an animal with id and name.
- Interacts with the database to save and read an animal.
- Validates the animal details.

Hence, the above class violates the **Single Responsibility Principle** (SRP), which states that a class should have only one reason to change. This principle is based on **cohesion**. Cohesion is a measure of how strongly a class focuses on its responsibilities. It is of the following two types:

- **High cohesion:** This means that a class is designed to carry on a specific and precise task. Using high cohesion, methods are easier to understand, as they perform a single task.
- **Low cohesion:** This means that a class is designed to carry on various tasks. Using low cohesion, methods are difficult to understand and maintain.

Hence the above code suffers from low cohesion. The above code can be improved as shown below. The *Animal* class is re-factored to have only a single responsibility of uniquely identifying an animal.

```
package principle_srp1a;

public class Animal {

    private Integer id;
    private String name;

    public Integer getId() {
        return id;
    }
}
```

```

    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

The responsibility of interacting with the database is shifted to a data access object (i.e. DAO) class. The data access object class takes an animal object or any of its attributes as input.

```

package principle_srp1a;

public interface AnimalDao {
    public void saveAnimal(Animal animal);
    public Animal readAnimal(Integer id);
}

```

```

package principle_srp1a;

import java.sql.Connection;

public class AnimalDaoImpl implements AnimalDao {

    private Connection con = null;

    // getters and setters for above attributes
    // go here..

    @Override
    public void saveAnimal(Animal animal) {
        // save Animal to database using SQL
        // goes here ...
    }
}

```

Classes and Interfaces Essentials

```
@Override
public Animal readAnimal(Integer id) {
    // read Animal from database using SQL
    // goes here ...
    Animal animal = null;
    // ...
    return animal;
}
}
```

Finally, the responsibility of validating an animal is re-factored to a separate class that takes an animal object as input.

```
package principle_srp1a;

public interface Validator {
    public boolean validate(Animal animal);
}
```

```
package principle_srp1a;

import org.apache.commons.lang.StringUtils;

public class AnimalValidator implements Validator {

    @Override
    public boolean validate(Animal animal) {
        return animal.getId() != null && animal.getId() > 0
            && StringUtils.isNotBlank(animal.getName());
    }
}
```

You now have 3 classes that have clear separation of concerns. The Animal class has been decoupled from database concern and validation concern. The above code is also well encapsulated and highly cohesive.

The challenge with SRP is getting the granularity of a responsibility right. One of the most common complaints about SRP is object explosion. This is a valid complaint,

but when things are broken down by concern as shown above, it is far easier to consider the concern in isolation and come up with a better design for that single concern. The art of design is all about striking a good balance by asking the right questions and not blindly following a principle.

Q28 Is there anything wrong with the following code snippet? **DC COQ SBQ**

```
package principle_ocp1;

public interface Animal {
    //methods are left out for brevity
}
```

```
package principle_ocp1;

public class Cat implements Animal {
    //methods are left out for brevity
}
```

```
package principle_ocp1;

public class Spider implements Animal {
    //methods are left out for brevity
}
```

```
package principle_ocp1;

public class Ostritch implements Animal {
    //methods are left out for brevity
}
```

```
package principle_ocp1;

import java.util.List;

public class AnimalLegsCounter {

    public int count(List<Animal> animals) {
        int count = 0;
```

Classes and Interfaces Essentials

```
for (Animal animal : animals) {  
    if (animal instanceof Cat) {  
        count += 4;  
    } else if (animal instanceof Spider) {  
        count += 8;  
    } else if (animal instanceof Ostritch) {  
        count += 2;  
    }  
}  
return count;  
}
```

```
package principle_ocp1;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Example {  
  
    public static void main(String[] args) {  
        List<Animal> list = new ArrayList<Animal>();  
        Animal animal = new Cat();  
        list.add(animal);  
        animal = new Spider();  
        list.add(animal);  
        animal = new Ostritch();  
        list.add(animal);  
  
        int count = new AnimalLegsCounter().count(list);  
        System.out.println("Total count for " + list.size() + " animals = " + count);  
    }  
}
```

A28 Skipped....

Questions 29 to 40 are skipped

Section-6:

Objects Essentials

- Working with objects
- Cloning objects
- Casting objects
- Immutable objects
- Working with enumerations
- Understanding “==” versus equals()
- Working with the String class
- Type safety with generics
- Serializing your objects
- Garbage collecting your objects

An object (or instance) is an executable copy of a class. In the last section, you looked at the object oriented concepts, design principles, working with classes, and interfaces. It is a must to have a good understanding in working with the objects. You will be performing various operations like creating, cloning, casting, comparing, modifying, serializing, and garbage collecting your objects.

The general preparation for a Java job interview can be summed up in four steps: research, predict questions, prepare answers and practice. The step 2 to predict questions is often not an easy task, but you can always prepare for most common interview questions relating to OO concepts, classes, interfaces, working with objects, and language fundamentals. Still more than 50% of the candidates don't answer these questions right. If you can demonstrate that your fundamentals are solid by answering these questions with examples and illustrations, you will stand a good chance of succeeding. This will be valued a lot more than happening to know or having experience with a sought-after framework the very moment. In other words, if you don't get these fundamental questions right, knowing the sought-after frameworks is not going to do you any good.

Working with Objects

The *java.lang.Object* class is the ultimate super class of all objects. If a class does not explicitly extend a class, then the compiler assumes it extends *java.lang.Object*.

Questions & Answers 1 ... 34 are skipped.

Type safety with Generics

Generics are a big step forward for Java, making it more readable and robust. Having said that, there are a number of limitations and can be tricky to use it correctly in your code. Remembering some of the rules will help you better understand generics.

Q34 Why use generics? **LF**

A34 Generics was introduced in JDK 5.0, and allows you to abstract over types. Without generics, you could put any object into a collection. This can encourage developers to write programs that are harder to read and maintain. For example,

```
List list = new ArrayList();
list.add(new Integer());
list.add("A String");
list.add(new Mango());
```

Since you can add any object, you would also not only have to use “instanceof” operator, but also have to explicitly cast any objects returned from this list.

```
package generics;

import java.util.ArrayList;
import java.util.List;

public class WithoutGenerics {

    public static void main(String[] args) {
        //Bad practice to add any object
        List list = new ArrayList();
        list.add(new Integer(5));           //index 0
        list.add("A String");               //index 1
        list.add(new Float(3.0));           //index 2
    }
}
```

```

//too many unsightly casts & instanceof constructs
Integer i = (Integer)list.get(0);
String s1 = null;
Object o1 = list.get(1);
if(o1 instanceof String){
    s1 = (String)o1;
}

//if you use the wrong cast, you can get a ClassCastException at runtime
String s2 = (String)list.get(2);    //index 2 is a Float, not a String .
}
}

```

Now with generics, your code becomes more robust and readable. **Rule 1:** The type safety is checked during **compile-time**.

```

package generics;

import java.util.ArrayList;
import java.util.List;

public class WithGenerics {

    public static void main(String[] args) {

        //can only add Integers
        List<Integer> list1 = new ArrayList<Integer>();
        list1.add(new Integer(5));

        //can only add Strings
        List<String> list2 = new ArrayList<String>();
        list2.add("A String");

        //can only add Floats
        List<Float> list3 = new ArrayList<Float>();
        list3.add(new Float(3.0));

        //compile error if you try to add a wrong type
    }
}

```

Objects Essentials

```
//list1.add("NOT ALLOWED");    //list1 contains Integers NOT String

//no casting is needed
Integer i = list1.get(0);
String s1 = list2.get(0);
Float f1 = list3.get(0);
}
}
```

The for-each loop that was added in JDK 1.5 works well with generics.

```
for(String aString : list2) {
    System.out.println(aString);
}
```

Q35 What do you understand by the term “type erasure” with regards to generics? **LF**

FAQ

A35 **Rule 1:** Java generics differ from C++ templates. Java generics (at least until JDK 7), generate only one compiled version of a generic class or method regardless of the number of types used. During **compile-time**, all the parametrized type information within the angle brackets are erased and the compiled class file will look similar to code written prior to JDK 5.0. In other words, Java does not support runtime generics.

Q. Why was it done this way?

A. This was done this way to achieve backward compatibility.

```
package generics;

import java.util.HashMap;
import java.util.Map;

public class TypeErasure {

    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<String, Integer>();
        map.put("Key1", new Integer(1));
        Integer val1 = map.get("Key1");           //casting is not required
        System.out.println(val1);
    }
}
```

```
}
```

If you run the compiled class file (i.e. *TypeErasure.class*) with the help of a Java decompiler discussed under platform essentials, you will get the following source code.

```
package generics;

import java.util.HashMap;
import java.util.Map;

public class TypeErasure
{
    public static void main(String[ ] args)
    {
        Map map = new HashMap();
        map.put("Key1", new Integer(1));
        Integer val1 = (Integer)map.get("Key1");
        System.out.println(val1);
    }
}
```

As you can see, all the angle brackets have disappeared, and casting has been added.

Q36 What does the following code fragment print? **LF COQ**

```
List<String> list1 = new ArrayList<String>();
List<Integer> list2 = new ArrayList<Integer>();
System.out.println(list1.getClass() == list2.getClass());
```

A36 Skipped....

Q37 What are the differences among **LF FAQ**

- raw or plain old collection type e.g. **Collection**
- Collection of unknown e.g. **Collection<?>**
- Collection of type object e.g. **Collection<Object>**

A37 Skipped....

Objects Essentials

Q&As 38 to 61 are skipped.

Preview

Section-7:

Logic and Data structures Essentials

- Java Flow Control
- Java Data structures and algorithms
- Logic Processing
- Exception handling

The data structures and logic processing are prevalently used in programming. It is important to have a basic understanding of the most common data structures like arrays, lists, sets, maps, trees, and graphs, and the basics of the “big-O” algorithmic complexity analysis. You should know why a particular data structure or an algorithm needs to be used for a given usage pattern. You should know the abstract data types such as *List*, *Stack*, or *Set* and the corresponding concrete implementations like *ArrayList*, *HashSet*, *HashMap*, etc.

It is also imperative to understand the performance versus memory tradeoffs between two different concrete implementations of an abstract data structure. The data structures can be traversed iteratively(e.g. arrays, lists, sets) or recursively (e.g. tree). Various sorting, partitioning, and searching algorithms can be applied to various data structures. Most data structures internally use an array or a list to provide a more specialized functionality. Thread-safety and immutability are other two considerations in using different data structures. The following are some common questions answered in this section.

“If Java did not have a *Map* or *Set* implementation, how would you go about implementing your own?”

“What are the common data structures, and where would you use them?”

Java Flow Control

Q1 Why would you prefer a short circuit “&&, ||” operators over logical “& , |” operators? **LF COQ**

A1 Firstly, *NullPointerException* is by far the most common runtime exception. If you use a logical operator, you can get a *NullPointerException*. This can be avoided easily by using a short circuit operator as shown below. There are other ways to check for null, but short circuit && operator can simplify your code by not having to declare separate if clauses.

```
if((obj != null) & obj.equals(newObj)) { // can cause a NullPointerException
    ....                               // obj.equals(newObj) is executed
                                     // even if obj != null returns false.
}
```

Short-circuiting means that an operator only evaluates as far as it has to, not as far as it can. If the variable 'obj' is null, it won't even try to evaluate the '*obj.equals(newObj)*' clause as shown below. This protects the potential *NullPointerException*.

```
if((obj != null) && obj.equals(newObj)) { // cannot get a NullPointerException
    ...                                   // because obj.equals(newObj) is
                                     // executed only if obj != null returns true
}
```

PC Secondly, short-circuit “&&” and “||” operators can improve performance in certain situations. For example:

```
//the CPU intensive method in bold is executed only if number is > 7
if((number <= 7) || (doComputeIntensiveAnalysis (number) <= 13)) {
    ...
}
```

Q2 What can prevent the execution of code in a finally block? **LF COQ**

A2

- An end-less loop.

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

```

        System.out.println("This line is printed .....");
        //endless loop
        while(true){
            ...
        }
    }
    finally{
        System.out.println("Finally block is reached.");    // won't reach
    }
}

```

- *System.exit(1)* statement.


```

public class Temp {

    public static void main(String[] args) {
        try {
            System.out.println("This line is printed .....");
            System.exit(1);
        }
        finally{
            System.out.println("Finally block is reached.");    // won't reach
        }
    }
}

```

- Thread death or turning off the power to CPU.
- An exception arising in a finally block itself.
- Process p = Runtime.getRuntime().exec("<o/s kill command>");

Q3 What will be the output of the following code snippet? 

```

package flow;

public class Finally1 {

    public static void main(String[] args) {
        System.out.println("Entering main()");
        System.out.println(new Finally1().getValue());
    }
}

```

Logic and Data structures Essentials

```
}

public String getValue() {
    try {
        System.out.println("Entering getValue()");
        return "Returning the value";
    } finally {
        System.out.println("executing the finally block");
    }
}
```

A3 Skipped ...

Q4 What will be the output of the following code snippet? **LF COQ**

```
package flow;

public class Finally2 {

    public static void main(String[] args) throws Exception {
        try {
            System.out.println("Entered the try block");
            if(1==1){
                throw new Exception("Some exception");
            }
        }
        finally{
            System.out.println("Executing the finally block.");
        }
    }
}
```

A4 Skipped....

Questions and Answers for 5 to 10 skipped

Java Data structures

Q11 What are the core interfaces of the Java collection framework? **LF**

A11 Skipped....

Q12 What are the common data structures, and where would you use them? **LF COQ**
OEQ

A12 Skipped....

Q12 What do you know about the big-O notation and can you give some examples with respect to different data structures? **LF OEQ**

A12 The Big-O notation simply describes how well an algorithm scales or performs in the worst case scenario as the number of elements in a data structure increases. The Big-O notation can also be used to describe other behavior such as memory consumption. At times you may need to choose a slower algorithm because it also consumes less memory. Big-o notation can give a good indication about performance for large amounts of data, but the only real way to know for sure is to have a performance benchmark with large data sets to take into account things that are not considered in Big-O notation like paging as virtual memory usage grows, etc. Although benchmarks are better, they aren't feasible during the design process, so Big-O complexity analysis is the choice.

The algorithms used by various data structures for different operations like search, insert and delete fall into the following performance groups like constant-time $O(1)$, linear $O(n)$, logarithmic $O(\log n)$, exponential $O(c^n)$, polynomial $O(n^c)$, quadratic $O(n^2)$ and factorial $O(n!)$ where n is the number of elements in the data structure. It is generally a tradeoff between performance and memory usage. Here are some examples.

Example 1: Finding an element in a *HashMap* is usually a constant-time, which is $O(1)$. This is a constant time because a hashing function is used to find an element, and computing a hash value does not depend on the number of elements in the *HashMap*.

Example 2: Linear search of an array, list, and *LinkedList* is linear, which is $O(n)$. This is linear because you will have to search the entire list. This means that if a list is twice as big, searching it will take twice as long.

Logic and Data structures Essentials

Example 3: An algorithm that needs to compare every element in an array to sort the array has polynomial complexity, which is $O(n^2)$. A nested for loop is $O(n^2)$. An example is shown under sorting algorithms.

Example 4: Binary search of a sorted array or *ArrayList* is logarithmic, which is $O(\log n)$. Searching an element in a *LinkedList* normally requires $O(n)$. This is one of the disadvantages of *LinkedList* over the other data structures like an *ArrayList* or array offering a $O(\log n)$ performance, which offers better performance than $O(n)$ as the number of elements increases. A logarithmic running times mean, if 10 items take at most x amount of time, 100 items will take say at most $2x$ amount of time, and 10,000 items will take at most $4x$. If you plot this on a graph, the time decreases as n (i.e. number of items) increases.

Preview

The decision as to using an unordered collection like a *HashSet* or *HashMap* versus using a sorted data structure like a *TreeSet* or *TreeMap* depends mainly on the usage pattern, and to some extent on the data size and the environment you run it on. The practical reason for keeping the elements in sorted order is for frequent and faster retrieval of sorted data if the inserts and updates are frequent. If the need for a sorted result is infrequent like prior to producing a report or running a batch process, then maintaining an unordered collection and sorting them only when it is really required with *Collections.sort(...)* could sometimes be more efficient than maintaining the ordered elements. This is only an opinion, and no one can offer you a correct answer. Even the complexity theories like Big-O notation like $O(n)$ assume possibly large values of n . In practice, a $O(n)$ algorithm can be much faster than a $O(\log n)$ algorithm, provided the data set that is handled is sufficiently small. One algorithm might perform better on an AMD processor than on an Intel. If your system is set up to swap, disk performance need to be considered. The only way to confirm the efficient usage is to test and measure both performance and memory usage with the **right data size**. Measure both the approaches on your chosen hardware to determine, which is more appropriate.

Q13 What is the tradeoff between using an unordered array versus an ordered array? **OEQ**

A13 Skipped....

Q&As 14 to 19 skipped

Q20 If Java did not have a *HashMap* implementation, how would you go about writing your own one? **SBQ COQ**

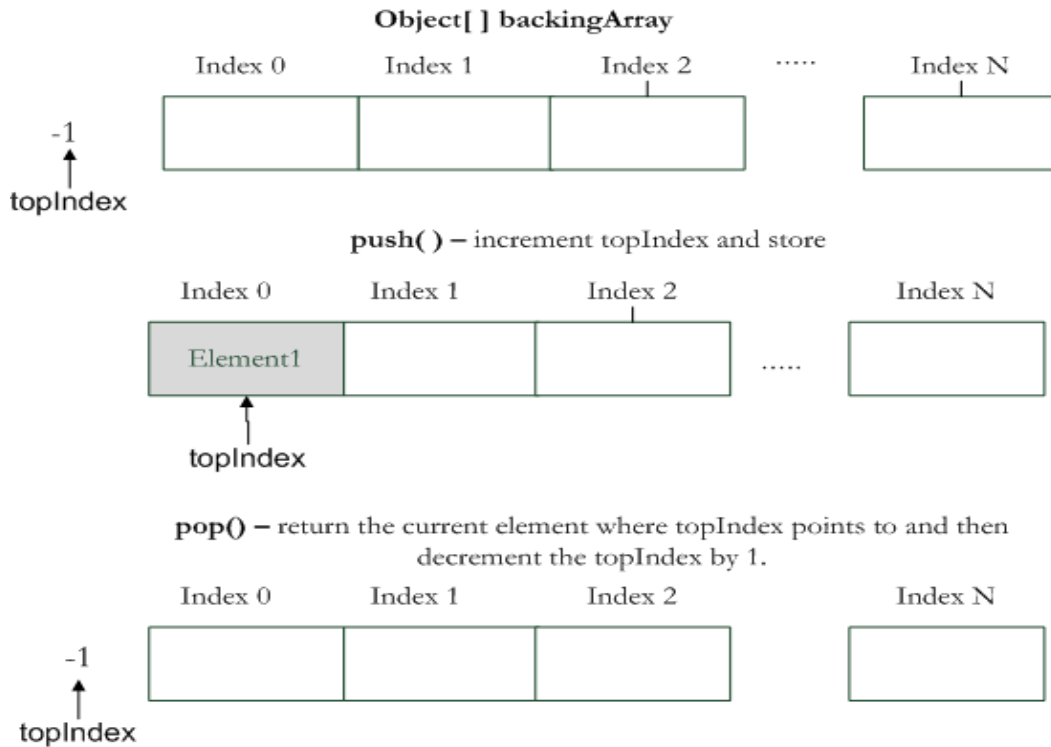
A20

Q20 If Java did not have a *Stack* implementation, how would you go about implementing your own? **SBQ COQ**

A20

- Determine the backing data structure (e.g. array, linked list, etc).
- Determine the methods that need to be implemented like *pop()*, *push()*, *peek()*, *clear()*, *empty()*, etc.
- Determine how you are going to keep track of the last item added. For example, keeping track with an index variable “*topIndex*”.
- Determine what to do if the capacity is reached? Double the capacity of the backing array.
- Drawing a diagram as shown below could make things clearer.

Logic and Data structures Essentials



Here are some code snippets to get started. Define the interface first.

```
package bigo;

public interface Stack<E> {
    E push(E item);
    E pop();
    boolean empty();
    //other methods like peek(), etc omitted for brevity
}
```

Now provide the implementation class.

```
package bigo;

public class MyStack<E> implements Stack<E> {
```



```

private Object[] backingArray;
private int topIndex;
private static final int DEFAULT_CAPACITY = 10;

public MyStack() {
    this.backingArray = new Object[DEFAULT_CAPACITY];
    topIndex = -1;
}

@Override
@SuppressWarnings("unchecked")
public E pop() {
    if (empty()) {
        throw new RuntimeException("No data is found");
    }
    // return the index element and reset the index by -1
    return (E) backingArray[topIndex--];
}

@Override
public E push(E item) {
    doubleTheArrayIfLimitIsReached();
    //increment the index and store the item
    this.backingArray[++topIndex] = item;
    return item;
}

@Override
public boolean empty() {
    return topIndex == -1;
}

/**
 * If the array capacity is reached, double it
 */
private void doubleTheArrayIfLimitIsReached() {
    if (topIndex + 1 == this.backingArray.length) {
        Object[] newArray = new Object[backingArray.length * 2];
    }
}

```

Logic and Data structures Essentials

```
// copy the existing elements to the new expanded array
System.arraycopy(backingArray, 0, newArray, 0,
    backingArray.length);
// set the backingArray to the expanded array
backingArray = newArray;
System.out.println(
    "The backing array has been expanded to " + backingArray.length);
}
}
}
```

Note: A queue can be implemented in a similar fashion by declaring a *backingArray*, a *frontIndex*, a *backIndex*, and a *currentSize*. Initialize the *frontIndex* to 0 and the *backIndex* to -1. You will have to provide the relevant methods like *enqueue()*, *dequeue()*, etc.

enqueue():

- If the *currentSize* == *backingArray.length* extend the backing array capacity by doubling it.
- Increment the *backIndex* by 1. If the incremented index is equal to the array length (*backingArray.length*), set the *backIndex* to 0 (i.e. imagine it as an endless circular array).
- Set the element to the *backIndex* (i.e. *backingArray[backIndex] = element*).
- Increment the *currentSize* by 1 (i.e. *currentSize++*).

dequeue():

- Decrement the *currentSize* by 1 (i.e. *currentSize--*).
- Return the value that *frontIndex* points to (i.e. *backingArray[frontIndex]*).
- Increment the *frontIndex* by 1. If the *frontIndex* == *backingArray.length*, set the *frontIndex* to 0 (i.e. imagine it as an endless circular array).

Note: Use the LIFO operations provided by the *Deque* interface and its implementations as opposed to the *Stack* implantation, which extends the legacy *Vector* class.

Q&As 21-24 skipped

Logic Processing

The interviewers are not only looking for your logic processing and coding capability, but also your reasoning ability and engineering skills. Think aloud where possible with the logical steps and key considerations like readability, maintainability, performance, memory utilization, possible test scenarios – both positive and negative unit tests, alternative approaches, pros, and cons for each alternative, etc. Don't compromise on readability, extendability, and maintainability for a small performance gain.

Your approach is as important if not more important than getting it right. The interviewers will also be looking to see if you are willing to change your code without the “I know it all” attitude and your willingness to learn. Time limitation and nervousness can play a part in arriving at the most efficient solution, but your approach and engineering skills can give interviewers the confidence that you are on the right track and can get the job done if they know how you are thinking.

Q25 Write a program that allows you to create an integer array of 5 elements with the following values: `int numbers[] = {5,2,4,3,1}`. The program computes the sum of first 5 elements and stores them at element 6, computes the average and stores it at element 7 and identifies the smallest value from the array and stores it at element 8. **COQ**

A25 Remember that the arrays are zero based.

```
package chapter2.com;

public class Numbers {

    public static void main(String args[] ) {
        // last three zeros are for the result
        int numbers[] = { 5, 2, 4, 3, 1, 0, 0, 0 };

        //set the minimum to a max value
        numbers[7] = Integer.MAX_VALUE;

        for (int i = 0; i < 5; i++) {
            // sum the numbers
```

Logic and Data structures Essentials

```
    numbers[5] += numbers[i];  
    // track the lowest  
    if (numbers[i] < numbers[7]) {  
        numbers[7] = numbers[i];  
    }  
}  
// average the numbers  
numbers[6] = numbers[5] / 5;  
System.out.println("Total is " + numbers[5]);  
System.out.println("Average is " + numbers[6]);  
System.out.println("Minimum is " + numbers[7]);  
}  
}
```

Output:

```
Total is 15  
Average is 3  
Minimum is 1
```

Q26 Write a program that will return whichever value is nearest to the value of 100 from two given int numbers. **COQ**

A26 Skipped....

Remaining Q&As for this section are skipped

Section-8:

Matching patterns with regular expressions

- The reason regular expressions are so useful is that they can be used to target what you need in any string with the flexibility of being able to use your matches to process the given string. Regular Expressions are integrated into many programmer tools and programming languages.
- Regular expressions are used to search, replace, filter, and validate data. Each time you use Regular Expressions, you are invoking a powerful search engine. This search engine is one that searches through text looking for specific patterns.
- Many find regular expressions to be a bit tricky, but if you can learn how to handle regular expressions you will have one of the most simplest, cleanest and efficient tools in your arsenal to take on a variety of problems.
- Regular Expressions can be used to match just about anything. Regular expressions are a language study all to themselves.

Matching patterns with Regular Expressions

The regular expressions are very powerful and it can be used in your Java code, shell scripts, UNIX commands, build scripts like ANT, XSD restrictions, IDEs, TEXT editors, and other tools. Here are a few reasons to motivate yourself to learn regular expressions (aka **regex**)

- Regex are everywhere. Even many experienced programmers are intimidated by it, and knowing how to use regex effectively will help you stand out from the crowd.
- Regex help you write short code.
- Regex saves time.
- Regex are fast if written with performance in mind.

Note: Please refer to the `java.util.regex.Pattern` class API for the summary of regular expression constructs.

Q01 How will you go about implementing the following validation rules for a user name?

- user name must be between 2 and 17 characters long.
- valid characters are A to Z, a to z, 0 to 9, . (full-stop), _ (underscore) and - (hyphen)
- user name must begin with an alphabetic character.
- user name must not end with a . (full stop) or _ (underscore) or - (hyphen).

A01 The above rules can be implemented with a regular expression as shown below:

```
package regex;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class UserNameRegex {

    public static final String PATTERN =
        "^[a-zA-Z][a-zA-Z0-9._-]{0,15}[a-zA-Z0-9]$";
    public static final Pattern p = Pattern.compile(PATTERN);
```

Matching patterns with regular expressions

```
public static boolean apply(String userName) {  
    Matcher matcher = p.matcher(userName);  
    return matcher.find();  
}  
}
```

PC Not compiling the regular expression can be costly if `Pattern.matches()` is used over and over again with the same expression in a loop or frequent method calls because the `matches()` method will re-compile the expression every time it is used. You can also re-use the `Matcher` object for different input strings by calling the method `reset()`.

What does this pattern mean?

<code>^</code>	<code>[a-zA-Z]</code>	<code>[a-zA-Z0-9._-]{0,15}</code>	<code>[a-zA-Z0-9]</code>	<code>\$</code>
Beginning of a line	Valid start characters. Should start with an alphabet. Must occur once.	Valid characters in the middle. Minimum occurrences 0 and max occurrences 15.	Valid end characters. Should not end with "." "-" or "_". Must occur once.	End of a line.

How will you test this? Test with JUnit. The `junit.jar` file must be in the classpath.

```
package regex;  
  
import org.junit.Assert;  
import org.junit.Test;  
  
public class UserNameRegexTest {  
  
    @Test  
    public void testMinLength() {  
        Assert.assertFalse("can't be <2", UserNameRegex.apply("P"));  
    }  
  
    @Test  
    public void testMaxLength() {  
        Assert.assertFalse("Can't be >17", UserNameRegex
```



```

        .apply("Jonathon-Christopher"));
    }

    @Test
    public void testCantEndWith() {
        Assert.assertFalse("can't end with . - _", UserNameRegex
            .apply("s.g.r."));
    }

    @Test
    public void testMustStartWith() {
        Assert.assertFalse("Must start with an alphabet",
            UserNameRegex.apply("23Lucky"));
    }

    @Test
    public void validNames() {
        Assert.assertTrue("Min Length 2", UserNameRegex.apply("Jo"));
        Assert.assertTrue("Max Length 17", UserNameRegex
            .apply("Peter-Christopher"));
        Assert.assertTrue(". - _ allowed in the middle",
            UserNameRegex.apply("s.g-h_k.r"));
        Assert.assertTrue("end with numeric", UserNameRegex
            .apply("Lucky23"));
    }
}

```

Q02 How would you go about validating a supplied password in your code that must adhere to the following conditions?

- Must contain a digit from 0-9.
- Must contain one uppercase character.
- Must contain one lowercase character.
- The length must be between 6 to 15 characters long.

A02 Skipped....

Q03 What does .*, +, and ? mean with respect to regular expressions? Where do you look for the summary of Java regular expression constructs?

Matching patterns with regular expressions

A03 *, +, and ? are known as (greedy) quantifiers as they quantify the preceding character(s). For example,

- . → matches any character.
- * → matches any character repeated 0 or more times.
- + → matches any character repeated 1 or more times.
- ? → matches any character repeated 0 or 1 time. This means optional.

- [a-zA-Z]* → Any alphabet repeated 0 or more times.
- [a-zA-Z]+ → Any alphabet repeated 1 or more times.
- [a-zA-Z]? → Any alphabet repeated 0 or 1.

Note: These are not wild characters. *, +, and ? are **regex repetitions**. The {x,y} is also used for repetition. For example, [a]{3,5} means the letter “a” repeated at least 3 times or a maximum of 5 times. In fact, internally the *Pattaren.java* class implements,

- **a*** as a {0, 0x7FFFFFFF}
- **a+** as a {1, 0x7FFFFFFF}

The values in square brackets (i.e. []) are known as character sets. The ., *, ?, etc are escaped and used as literal values within the square brackets. Here are some examples of the character sets.

- [aeiou] → matches exactly one lowercase vowel.
- [^aeiou] → matches a character that ISN'T a lowercase vowel (^ inside [] means NOT).
- ^[a-z&&[^aeiou]]*\$ → matches any character other than a vowel anchored between start (^) and end (\$). This is a character class **subtraction regex**. The “&&” means intersection.
- [a-d[m-p]] → Matches characters a to d and m to p. This is a **union regex**.
- [a-z&&[d-f]] → Matches only d, e, and f. This is an **intersection regex**.
- [x-z[\p{Digit}]] → matches x-z and 0-9. Similar to [x-z0-9] or [x-z[\d]]. The “\p” stands for POSIX character classes.
- ^[aeiou] - matches a lowercase vowel anchored at the beginning of a line
- [^ ^] → matches a character that isn't a caret '^'.
- ^[^] → matches a character that isn't a caret at the beginning of a line.
- ^[^\.] → matches anything but a literal period, followed by "any" character, at

the beginning of a line

- `[.*]*` → matches a contiguous sequence of optional dots (.) and asterisks (*)
- `[aeiou]{3}` - matches 3 consecutive lowercase vowels (all not necessarily the same vowel)
- `\[aeiou\]` → matches the string "[aeiou]". "\" means escape.

Remaining Q&As are skipped

Section-9

Job Interview Tips

- Interviews are not technical contests...
- Don't wait to be asked
- Think out loud and brainstorm with the interviewers
- Sometimes knowing something is better than knowing nothing ...
- Interviewers place a lot of value in "I don't know" over inventing answers
- An interview is a two way street ...
- Open-ended questions are your opportunity to
- Books can impart knowledge, but cannot give you the much needed experience ...

Interviews are not technical contests...

to see who gets the most technical questions right. It is all about hiring the right person who can get the job done. To get the job done, relevant technical skills must be complemented with good soft-skills, right attitude, and passion. Open ended questions are your ideal opportunity for you to make a statement about your technical and non-technical abilities to get the job done.

Q. Why do you like software development?

A. [Hint:]

- Have special interest in pro-actively and re-actively resolving thread-safety issues, performance issues, design gaps, and general development issues, where I get to apply and expand technical, analytical, researching, problem-solving, communication, and people skills.
- Enjoy analyzing the pros, cons, and trade-offs of each design and development alternatives to arrive at an effective and workable solution that can adapt to growing and changing business needs.
- Software development can be complex as there are so many moving parts, and it is quite satisfying to get your work through the SDLC (Software Development Life Cycle) phases like design, development, deployment, testing, documentation, support, and hand-over. There is always a variety of tasks and responsibilities to be performed. Opportunity to applying agile practices where appropriate, makes your journey even more enjoyable by interacting more closely with the multidisciplinary teams.
- Motivated by having something new to be learned in every project in terms of domain knowledge, technologies, frameworks, tools, best practices, anti-patterns, business processes, development processes, and people. For example, Java 5.0 features like annotations, generics, enhanced for loop, auto-boxing, enums, and `java.util.concurrent` packages take ease of development to a new level.
- It involves both the technical side as well as the people side to have your project completed successfully.

Job Interview Tips

Remaining Q&As are skipped

Preview

<i>Glossary & Index</i>	
Abbreviation	Description
aka	Also Known As
ANTLR	ANother Tool for Language Recognition
AOP	Aspect Oriented Programming
API	Application Programming Interfaces
APT	Annotation Processing Tool
BCEL	Byte Code Engineering Library
EE	Enterprise Edition
ETL	Extract Transform Load
FIFO	First In First Out
i.e.	That is
IDE	Integrated Development Environment
JAR	Java ARchive
JAX-WS	Java API for XML Web Services
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JDK	Java Development Kit
JIT	Just In Time
JMX	Java Management eXtension
JPA	Java Persistence API
JRE	Java Runtime Environment
JSR	Java Specification Request
JVM	Java Virtual Machine
LIFO	Last In First Out
MBean	Managed Bean
ME	Micro Edition

Glossary & Index

[illegible]

Index

Classes and Interfaces Essentials.....	
Applying the design principles.....	
Can you list some of the key principles to remember while designing your classes?.....	281
How can you improve on the following code snippet?.....	276
How will you improve on the following code snippets with appropriate design principles?.....	268
Is there anything wrong with the following code snippet?.....	272
What is your understanding of some of the following most talked terms like Dependency Inversion Principle (DIP), Dependency Injection (DI) and Inversion of Control (IoC)?.....	279
Class invariant and design by contract.....	
What do you understand by the term “design by contract”?.....	289
What is a class invariant?.....	285
Designing an application or a system.....	
How would you go about designing a car parking station?.....	284
How would you go about designing a system as described below.....	281
What is a class invariant?.....	285
Designing your classes and interfaces.....	
Can you explain if the following classes are badly designed?.....	234
How do you know that your classes are badly designed?.....	234
What are the 3 main concepts of OOP?.....	241
What are the principles of class design used in regards to OOD?.....	240
What do you achieve through good class and interface design?.....	240
What problem(s) does abstraction and encapsulation solve?.....	244
Inheritance versus composition.....	
Can you give an example of the Java API that favors composition?.....	267
Can you give an example where composition is favored over implementation inheritance?.....	267
How do you achieve code reuse in your application?.....	260
How do you achieve polymorphic behavior?	260
What is the difference between implementation inheritance and composition?.....	260
What questions do you ask yourself to choose composition (i.e. has-a relationship) for code reuse over implementation inheritance (i.e. is-a relationship)?.....	261
Packaging your classes to avoid conflicts.....	
How do you go about designing good packages?.....	300
What are the usages of Java packages? How do you go about designing good packages?.....	300
Subclassing, overriding, and hiding (aka shadowing).....	
What can you do in a subclass?.....	227
What happens when a parent class and a child class each have a static method with the same signature?	233
What is a subclass?.....	227
What is the output of the following code snippet? Give your reasons and recommendations?.....	228
Working with abstract classes and interfaces.....	
Can interfaces have constant declarations?.....	259
Can you give an example of an interface driven framework in the Java API?.....	259
What are marker interfaces in Java? Why to have marker interfaces?.....	259

Glossary & Index

What can an interface do that an abstract class cannot? What are the differences between abstract classes and interfaces?.....	256
What do you understand by abstract classes and interfaces?.....	247
What is the significance of abstract classes & interfaces with respect to OO design?.....	256
What modifiers can be used in an interface?.....	259
When would you prefer one over the other?.....	258
Working with classes and interfaces.....	
What is a class? What are the valid modifiers of a top level class?.....	226
Which class declaration is correct if A and B are classes and C and D are interfaces?	226
Working with inner classes.....	
Can you have an inner class inside a method and what variables can you access?.....	295
What is a callback method?.....	296
What is an inner class? Can you provide an example from the Java API? When to use inner classes?..	290
Knowing your way around Java.....	
Documenting your Java applications.....	
In your experience, why is it important to have relevant APIs handy while coding?.....	117
What are the different types of documents you refer to while designing or coding your system?	115
What do you expect to see in a high level or detailed business requirements document?.....	121
What do you expect to see in a high level or detailed technical specification document?.....	120
What tips do you give to someone who wants to write a technical documentation?.....	122
Ensuring code quality.....	
Can you list some of the key aspects you keep in mind while coding to ensure quality & robustness?..	129
Do you use test driven development? Why / Why not?.....	122
How will you ensure quality of your design and code?.....	125
What are mock objects and when do you use them?.....	131
What development methodologies are you comfortable with?.....	132
What tips do you give someone who is writing the unit testing?.....	122
Exposure to tools, technologies, and frameworks.....	
Can you name some tools, technologies, or frameworks you had to use in Java for source code generation, byte code manipulation, or assembling code at runtime?.....	103
How would you go about.....? questions like	114
What other languages have you used?.....	112
What tools do you generally need to get your job done?.....	108
Gauging Your Experience with Unix.....	
Can you answer the following Unix questions?.....	98
Can you brief on a shell script that you had worked on?.....	102
Can you give some examples of where you used “sed” and “awk” utilities?.....	93
Can you list some of the “UNIX” commands Java developers use very frequently?.....	88
Judging Experience.....	
Can you explain thread-safety and atomicity with an example? What do you understand by optimistic versus pessimistic locking?.....	74
Can you list some of the Java features you used recently?.....	52
Can you list the Java SE features you dislike and why?.....	58
Can you list the Java SE features you like to be added in the future?.....	62
Can you list the Java SE features you really love and why?.....	55
Give me a high level description of your experience with the Java platform?.....	67

In your experience, what are some of the most common errors Java programmers make?.....	82
What are some of the very common runtime (or unchecked) exceptions you have come across in Java?.....	73
When is a ConcurrentModificationException thrown?.....	69
What are your favorite areas of the Java APIs and why?.....	78
What open source libraries/frameworks do you have experience with?.....	54
Who are your role models? What books really inspired you? What was the recent Java book you read?.....	
What websites do you use to keep your knowledge current?.....	
Language Essentials.....	
Choosing the right data types.....	
How would you go about choosing the right data types for your application? What are wrapper classes, and why do you need them?.....	144
When working with floating-point data types, what are some of the key considerations?.....	147
Class, instance, and local variables.....	
What are the different kinds of variables in Java? What are some of the best practices relating to usage of these types of variables?.....	217
What will be the output for the following code?.....	221
Constants and static imports.....	
What is a constant interface anti-pattern? How will you avoid it?.....	182
Initializing your data correctly.....	
Can you tell me what is the use of a static block in a class with an example? Is it a good practice to use static blocks?.....	179
How will you provide a member variable named dueDate with an initial default value set to first day of the following month?.....	181
Is there anything wrong with the following code snippet?.....	181
What would happen when the following code is compiled and run?.....	176
Java is pass-by-value.....	
Explain the statement Java is always pass by value?.....	211
The value of Point p before the following method calls is (10,20). What will be the value of Point p after executing the following method calls?.....	213
Methods and constructors.....	
Can you extend a class with a private constructor? For example, a singleton class may have a private constructor. What would you do, if you want to extend a singleton class?.....	201
How many ways can an argument be passed to a subroutine and explain them?.....	207
There is a class X.java and it has a public method. Class Y extends class X. How would you prevent a method in class X from being accessed in class Y using Y's instance?.....	205
What are some of the do's and don'ts with respect to constructors in order to write more testable code?.....	203
What tip would you give your fellow developers to make their code more testable?.....	199
What are static factory methods? Can you give examples from the Java API? What design patterns does it use? Can you describe the benefits of static factory methods and usefulness of private constructors?.....	207
What are varargs?.....	206
What do you understand by the term "co-variant" in Java?.....	209
What is the difference between an argument and a parameter?.....	
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? Are constructors inherited? How do you call a super class's constructor?.....	196
When is a method said to be overloaded and when is a method said to be overridden? What are their	

Glossary & Index

differences? What is a co-variant return type?.....	204
Where and how can you use a private constructor?	198
Modifiers and where are they applicable?.....	
Are 'volatile' and 'const' valid modifiers in Java?.....	192
Discuss the significance of the modifiers used in the following class?.....	186
How would you go about determining what access levels to use in your code?.....	194
If you want to extend the java.lang.String class, what methods will you override in your extending class?	192
If you were to give some tips on modifiers, what would they be?.....	195
What are the valid access modifiers?.....	186
Why use modifiers? Why can't you declare a class as both abstract and final?.....	186
Recursive functions.....	
How would you take advantage of Java being a stack based language? What is a re-entrant method?..	214
What are idempotent methods?.....	216
Thinking in binary and bitwise operations.....	
Can you convert the decimal value of 127 to binary, octal, and hexadecimal values?.....	163
Can you list some practical applications where the bitwise operations can be applied?.....	171
How do you represent negative numbers in Java? What is the negative binary value for the number 37?	165
What are the common bitwise operations?.....	166
Understanding the operator precedence.....	
Do you think the following code will throw a compile-time exception? If yes, how will you fix it?.....	159
What do you understand by operator precedence in Java?.....	159
What is the output of the following code snippet?.....	160p.
What will be the output of following operations?.....	162
Valid identifiers and reserved keywords.....	
What are identifiers in Java? What does the following code do? Can you talk us through the code highlighting some of the key language and design features?.....	135
Which of the following are keywords in Java?.....	142
Which of these are legal Identifiers?.....	139
Why is it a best practice to adhere to naming conventions and what are they?.....	143
Widening versus narrowing conversions.....	
Can you explain why the code snippet under (a) gives a compile-time error and code snippet under (b) compiles and runs with an output of 20?.....	153
What are the dangers of explicit casting?.....	153
What is wrong with the following code, and how will you fix it?.....	157
What is your understanding of widening versus narrowing conversions of primitive data types? Can you explain explicit and implicit casting?.....	150
Logic and Data structures Essentials.....	
Algorithms.....	
Can you write an algorithm to.....?.....	427
What are the different binary tree traversal mechanisms?.....	439
What are the different searching methodologies for trees that may not have any particular ordering?	
Which approach will you use if you are likely to find the element near the top of the tree? Which approach will you be using if you are likely to find the element near the bottom of the tree?.....	447
Java Data structures.....	

Describe typical use cases or examples of common uses for various data structures provided by the Collection framework in Java?.....	404
How do you get an immutable collection?.....	408
If Java did not have a HashMap implementation, how would you go about writing your own one?.....	414
If Java did not have a Stack implementation, how would you go about implementing your own?.....	418
What are some of the best practices relating to the Java Collection framework?.....	410
What are the common data structures, and where would you use them?.....	391
What are the core interfaces of the Java collection framework?.....	390
What do you know about the big-O notation and can you give some examples with respect to different data structures?.....	399
What does the following code do? Can the LinkedHashSet be replaced with a HashSet?.....	410
What is the difference between Collection and Collections?.....	408
What is the tradeoff between using an unordered array versus an ordered array?.....	402
What method(s) should you override for objects you plan to add to sets or maps?.....	408
Java Flow Control	
Can you write a method to print out a multiplication table?.....	388
If you have large if-else or switch-case statements, what would come to your mind?.....	388
What can prevent the execution of code in a finally block?.....	384
What is a conditional operator?.....	389
What is a fibonacci sequence? Write a method to compute the Nth fibonacci number without using nested loops?.....	390
What is the difference between while and do while loop?.....	387
What will be the output of the following code snippet?.....	385, 387
Why would you prefer a short circuit “&&, ” operators over logical “&, ” operators?.....	384
Write a method to print the odd numbers from 1 to 99?.....	388
Sorting Elements	
If I mention the interface names Comparable or Comparator, what does come to your mind? Why do we need these interfaces?.....	422
Matching patterns with Regular Expressions	
Can you discuss what the following regular expression does?.....	502
Can you give me some real life scenarios where you used regular expressions?	507
Can your write a function that will replace all tokens delimited by @ with a given String?	501
How will you go about implementing the following validation rules for a user name?.....	488
How will you go about matching comments in an XML or HTML file?.....	504
How would you configure the Java regex engine to be case insensitive?.....	506
How would you go about validating a supplied password in your code that must adhere to the following conditions?.....	490
If you had a special requirement to strip out currencies from a specific report, how would you go about stripping it out?.....	518
What do you understand by greedy, reluctant, and possessive quantifiers? What do you understand by the term “backtracking”?	498
What does .*, +, and ? mean with respect to regular expressions? Where do you look for the summary of Java regular expression constructs?.....	493
What does grouping mean with regards to regular expressions? Would you prefer capturing or non-capturing group?.....	494

Glossary & Index

Which of the following will be matched by the regex “.*[^ a-zA-Z0-9].*” ?.....	518
Which of the following will be matched by the regex “.*[^ 0-9].*” ?.....	518
Write a regular expression for the following scenarios?.....	515
Objects Essentials	
Casting Objects.....	
If you have a reference variable of a parent class type, and you assign a child class object to that variable, and then you invoke a static method that is present in both parent and child classes, which method will be invoked?.....	316
What do you understand by variable shadowing?.....	315
What is type casting? Explain up casting vs. down casting? When do you get ClassCastException?.....	314
Cloning Objects.....	
What is the main difference between shallow cloning and deep cloning of objects?.....	312
Garbage Collection.....	
Assume that the following utility method is very frequently accessed by parallel threads, and profiling indicates that this method is causing GC to over work. How would you go about improving the following implementation?.....	381
Explain different types of references in Java?.....	374
How do you know if your garbage collector is under heavy use? What are the best practices to minimize heavy use of garbage collection?.....	378
If you have a circular reference of objects, but you no longer reference it from an execution thread (e.g. main thread), will this object be a potential candidate for garbage collection?.....	377
What are the different ways to make an object eligible for Garbage Collection when it is no longer needed?.....	377
What do you know about the Java garbage collector? When does the garbage collection occur?.....	372
What is the difference between final, finally and finalize() in Java?.....	372
When does the GC stop? Who controls the GC? Can GC be forced?.....	378
Which part of the memory is involved in Garbage Collection? Stack or Heap?.....	378
Immutable Objects.....	
How will you prevent the caller from adding or removing elements from pets? How will you make this code fully immutable?.....	324
How would you defensively copy a Date field in your immutable class?.....	323
What about data that changes sometimes? Is there any way to obtain the benefits of immutability with the added benefit of thread-safety for data that changes less frequently?.....	325
What is an immutable object? How do you create an immutable type? What are the advantages of immutable objects?.....	318
Why is it a best practice to implement the user defined key class as an immutable object?.....	319
Serializing your objects.....	
Are there any disadvantages in using serialization?.....	370
How would you exclude a field of a class from serialization?.....	369
What are the common uses of serialization? Can you give me an instance where you used serialization?.....	369
What happens to static fields during serialization?.....	369
What is a serial version id?.....	370
What is serialization? How would you exclude a field of a class from serialization or what is a transient variable? What happens to static fields during serialization? What are the common uses of serialization?.....	
What is a serial version id? Are there any disadvantages in using serialization?.....	365

What is the difference between Serializable and Externalizable interfaces? How can you customize the serialization process?.....	370
Which Java interface must be implemented by a class whose instances are transported via a Web service?	364
Type safety with Generics.....	
Can you identify any issues with the following code?.....	363
Does the following code snippet compile? What does it demonstrate?.....	363
Is it possible to generify methods in Java?.....	361
Is it possible to generify your own Java class?.....	356
Is the following code snippet legal? If yes why and if not why not?.....	360
Is the following code snippet legal? If yes why and if not why not?	360
What are the differences among	350
What do you understand by the term “type erasure” with regards to generics?.....	348
What does the following code fragment print?.....	350
Why use generics?.....	346
Understanding “==” versus equals().....	
Can you discuss the output of the following code?.....	335
Can you discuss what gets printed?.....	336
What happens when you run the following code?.....	334
What is the difference between “==” and equals(.) method? What is the difference between shallow comparison and deep comparison of objects?.....	332
Working with Enumerations.....	
Are enums immutable?.....	330
Can enums be used in switch statements?.....	328
Do you have to override equals() and hashCode() methods for enum?.....	330
How will you convert a String value to an enum value?.....	329
How will you get an integer equivalent of an enum value?.....	329
Why is it a best practice to use enums over static final int or String constants?.....	326
Would you use equals() or == to compare enum values?.....	329
Working with Objects.....	
Can you override clone() and finalize() methods in the Object class? How do you disable a clone() method?.....	310
If a class overrides the equals() method, what other method it should override? What are some of the key considerations while implementing these methods?.....	306
What are the non-final methods in Java Object class, which are meant primarily for extension? Are these methods easy to implement correctly? What are the implications of implementing them incorrectly?..	306
When should you override a toString() method?.....	309
Working with the String class.....	
Can you explain how Strings are interned in Java?.....	342
Can you list some escape characters in Java? Where are they generally used?.....	343
Can you talk through the output of the following code snippet?.....	341
Can you write a method that reverses a given String?.....	339
How will you split the following string of text into individual vehicle types?.....	344
What are the different ways to concatenate strings? Which approach is most efficient?.....	345
When do you use a String and when do you use a StringBuffer or StringBuilder?.....	338
Why String has been made immutable in Java?.....	343
Platform Essentials.....	

Glossary & Index

Compile-time versus runtime	
Are marker or tag interfaces obsolete with the advent of annotations (i.e. runtime annotations)?.....	44
Can you differentiate compile-time inheritance and runtime inheritance with examples and specify which Java supports?.....	43
Does this happen during compile-time, runtime, or both?.....	39
Explain compile-time versus runtime?.....	39
What is a Java debugger? What are the different ways to debug your code?.....	48
What is the difference between line A & line B in the following code snippet?.....	47
Handling Exceptions	
What are the best practices in regards to exception handling?.....	483
What is a user defined exception? When would you define your own exception?.....	481
What is the difference between exception and error?.....	481
When do you want to use checked exceptions and when do you want to use unchecked exceptions?..	476
Why does Java have exceptions? What is the difference between checked and unchecked exceptions?.	471
Will the following code compile?.....	474
How are your classes loaded?	
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?.....	34
Explain static vs. dynamic class loading?.....	36
What tips would you give to someone who is experiencing a class loading or “Class Not Found” exception?.....	37
Java platform basics – JDK, JRE, JVM, etc	
Can you briefly describe the JDK and JRE file structures?.....	28
How do you monitor the JVMs?.....	25
How would you differentiate JDK, JRE, JVM, and JIT?.....	22
What are some of the JVM arguments you have used in your projects?.....	27
What are the two different bits of JVM? What is the major limitation of 32 bit JVM?.....	24
What are the two flavors of JVM?.....	24
What does jar stand for? How does it differ from a zip file?.....	29
What is the main difference between the Java platform and the other software platforms?.....	21
Logic Processing	
Can you write a code to evaluate() a simple mathematical calculation as shown below using a binary tree?.....	461
Write a method which takes the parameters (int[] inputNumbers, int sum) and checks input to find the pair of integer values which totals to the sum. If found returns true, else returns false?.....	455
Write a program that allows you to create an integer array of 5 elements with the following values: int numbers[] = {5,2,4,3,1}. The program computes the sum of first 5 elements and stores them at element 6, computes the average and stores it at element 7 and identifies the smallest value from the array and stores it at element 8.....	450
Write a program that will return factorial of n (usually written as n!), which is the product of all integers up to and including n (1 * 2 * 3 * ... * n). Can you explain both recursive and iterative approaches?	
Which approach would you prefer?.....	452
Write a program that will return whichever value is nearest to the value of 100 from two given int numbers.....	451
Setting up and running Java	
How do you create a class under a package in Java?.....	32
What do you need to develop and run Java programs? How would you go about getting started?.....	30

What do you need to do to run a class with a main() method in a package?.....	32
Why use Java?	
Give a few reasons for using Java?.....	16
Is Java 100% object oriented? If yes, why? And if not, why not?.....	18
What does Java comprise of?.....	20
What is the difference between C++ and Java?.....	17

Glossary & Index

Please email any suggestions or errors to java-interview@hotmail.com. For other career making resources, please visit <http://www.lulu.com/java-success>. The career companions and essentials will compliment each other to lift your career a few notches.

"Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius -- and a lot of courage -- to **move in the opposite direction.**"

- Albert Einstein

"If you can't explain it to a six year old, you don't understand it yourself."

- Albert Einstein

