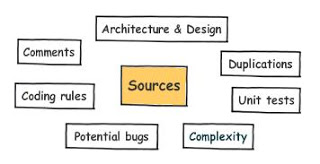
**Q.** How do you ensure code quality in your application?  
**A.**code quality means writing **readable**and **robust**code, that conforms as much as possible to the style-guideline that is used, and that has as little as possible defects. It also means writing **maintainable**code with proper automated and manual tests.   
  
**1. Write a number of automated tests**

* **Unit tests** using**JUnit**or**TestNG**. For unit tests use mock objects to ensure that your tests don't fail due to volatility of the data changes. There are mocking frameworks like **EasyMock**, **Mockito**, and **PowerMock**.
* **Integration testing** of your services with JUnit or TestNG. Your integration tests are only as good as the quality of the data. You could either use dedicated test databases or use frameworks like**DBUnit** to manage extraction and insertion of data.
* **Web testing**  using **Selenium + WebDriver**. Selenium + WebDriver ( [Selenium interview questions and answers](http://java-success.blogspot.com/2011/10/selenium-web-driver-interview-questions.html)) allows you to reenact web user experience and run it as an automated unit test using JUnit or TestNG.Your tests are only as good as the quality of the data. You could either use dedicated system test databases or use frameworks like **DBUnit**. DBUnit allows you to extract the data from databases into flat XML files, and then refresh (i.e. insert or update) the data into the database during setup phase of running the unit tests. There are handy proxy JDBC driver tool called **P6SPY**, which logs the SQL queries that are executed against the database by the DBUnit. This P6SPY also very handy in debugging Hibernate’s generated SQL  by acting as a proxy driver between JDBC and the real driver so that all generated SQL will be logged. There are other Web testing tools like **Badboy**.
* **Load testing your application**with tools like **JMeter**, **OpenSTA**, etc.  The Badboy compliments JMeter by allowing you to record scripts and then exporting the scripts as a JMeter file to be used in JMeter.[JMeter Interview Questions and Answers](http://java-success.blogspot.com/2011/10/jmeter-interview-questions-and-answers.html)

**2. Have regular code reviews.** There are tools like **Crucible** from Atlassian that gives your team an efficient way to benefit from the power of constant code review with features like inline commenting, simple workflow, asynchronous reviews, email and RSS notifications, JIRA integration and much more.   
  
**3. Using a number of code quality tools.**

* **Checkstyle**ensures the style of your Java code is standardized and "nice". It checks white spaces, new lines, formatting, etc. (i.e. it looks on the code line by line).  This only ensure style of your code.
* On the other hand there is **PMD**which not necessarily checks the style of your code but it checks the structure of the whole code. PMD scans Java source code and looks for potential problems like possible bugs, dead code, suboptimal code, overcomplicated expressions, duplicate code, etc.
* **FindBugs**is a static analysis tool to look for bugs in Java code. It discovers possible*NullPointerExceptions*and a lot more bugs.
* **Sonar** is a very powerful tool covering 7 axes of code quality as shown below.

[](http://2.bp.blogspot.com/-Tfolrz8o5II/TpZL4nd1K2I/AAAAAAAAAHg/0mgJ5dCgWdE/s1600/code_quality_sonar.JPG)

(The diagram is from the Sonar website)

**4. Using continuous integration servers** (on a clean separate machine) like **Bamboo**, **Hudson**,**CruiseControl**, etc to continuously integrate and test your code.  
  
**5. Not stopping to code once the code works.** Too many developers feel their job stops at making something happen. It is a best practice to constantly **refactor code**with proper unit tests in place.   
  
  
**Q.** Do you use test driven development? Why / Why not?   
**A.** [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.
* You won't waste time writing features you don't need.

**Q**. How do you keep your knowledge up to date about writing quality code?  
**A**. Mainly through good books and online resources.  
  
**Through good books**

* **Code Complete**: A Practical Handbook of Software Construction by Steve McConnel
* **Clean Code**:  A Handbook of Agile Software Craftsmanship by Robert. C. Martin

**Through good articles and  quick tips at online resources like**

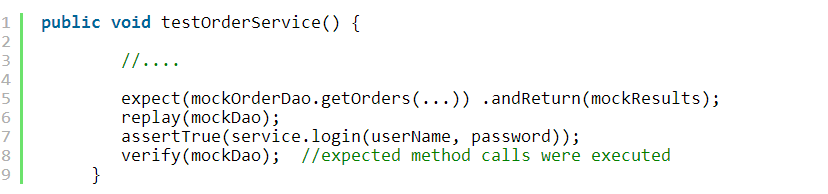
* JavaLobby (java.dzone.com)
* TheServerside.com
* InfoQ.com
* handy blogs on Mockito, PowerMock, Easy mock, DBUnit, Selenium, JUnit, TestNG, etc.

**Q.** What do you look for when you are reviewing others' code?  
**A.** Firstly, and most importantly look to see if the following key areas are properly adhered to avoid any potential issues relating to thread-safety, performance, memory leak, scalability, transaction management, exception handling, etc. Also, look for the key areas like best practices, design concepts, and design patterns. [Key Areas](http://java-success.blogspot.com/2006/10/learn-javaj2ee-core-conceptskey-areas.html).  
  
Secondly, ensure that proper test cases are written. Also, ensure that it has a good code coverage.  
  
Here are a few finer points:

* Naming conventions.
* Existence of unused code and commenting code out. Delete unused code. Source control is there for maintaining the history of your code.
* Unnecessary comments.  The method and variable names must be self explanatory without cluttering the code with excessive comments. The methods should do only one thing with fewer lines of code. More than 15-20 lines of code is questionable. The number of parameters passed in must also be small. The public methods must fail fast with proper validations.
* Repeated code due to copy-paste. For example, same logic or hard coded values repeated in a number of places.
* Favoring trivial performance optimization over readability and maintainability.
* Tightly copuled code. For example, not coding to interfaces, favoring inheritance over composition, etc.
* Badly defined variable scopes or variable types. For example, using a data type double to represent monetary values instead of BigDecimal. The variable scopes must be as narrow as possible.
* Using mutable objects and read only varaibles where immutable objects make more sense.
* Proper implementation of language contracts. For example, not properly implemented equals and hashCode methods.
* Deeply nested loops or conditionals. Nested loops can be replaced with changing the logic or through recursion. Nested if-else conditionals are a good candidate for applying polymorphism.
* Not properly handling exceptions. For example, burying exceptions, exposing internal exception details to the users without proper friendly messages at the Web  layer, etc.
* Badly written unit tests.
* Not designing the classes and interfaces with proper design concepts and principles. For example, strongly coupled classes, classes trying to do more things than it should, modelling it as a class when it should be an attribute, etc.
* Not handling and testing non-functional scenarios. For example, not properly handling service timeouts or using incorrect timeout values.
* Reinventing the wheel by writing your own implementation, when there is already a proven and tested  implementation provided by the API.

**Note:**The Core Java Career Essentials cover all the above in detail with examples.  
  
  
**Q.** There are lots of advantages in writing tests, but in your experience, are there any disadvantages?  
**A**. [**Hint**: Everything has pros and cons. Discussing the cons demonstrates your experience, but make it a point to mention how you overcome the cons to exemplify your problem solving, researching, and analytical skills.   
  
Tests need to be maintained. If a particular method signature or logic is changed, then all the relevant tests need to be fixed. Many developers don't put lots of thoughts into writing quality test cases through proper **what if** analysis. For example, what if new records are added to the database? What if the data is missing?, etc. So, this continuous tweaking of test cases will require time. Writing quality tests require experience and lots of  analysis.you need to have an enthusiastic team and at least one experienced developer who knows how to write good tests, and also knows a few thing about good architecture  
  
Requires some**initial upfront investment** in time and cost. But, this initial up front investment will pay-off  significantly in a long run through quality code and robust application.  
  
**Q.** What is the difference between fake objects, mock objects, and stubs?  
**A.**

* **Fake**objects build a very lightweight implementation of the same functionality as provided by a component that you are faking. Since they take some shortcut, they are not suitable for production.
* **Mocks**are objects pre-programmed with expectations which form a specification of the calls they are expected to receive. You can use mocking frameworks like *EasyMock*, *Mockito*, *PowerMock*, etc to achieve this.When an actual service is invoked, a mock object is executed with a known outcome instead of the actual service. With mock objects, you can verify if expected method calls were made and how many times. The verify(mockOrderDao) shown below tells EasyMock to validate that all of the expected method calls were executed and in the correct order.



* **Stubs**are like a mock class, except that they don't provide the ability to verify that methods have been called or not called. Generally services that are not ready or currently not stable are stubbed to make the test code more stable.

You use a Mock when it's an object that returns values that you set to the tested class. You use a Stub to mimic an Interface or Abstract class to be tested. In fact, the difference is very subtle and it doesn't really matter what you call it, fake, mock, or stub, they are all objects that aren't used in production, and used for managing complexity to write quality tests.   
  
  
**Q.** Can you list some of the key principles to remember while designing your classes?  
**A.** Use design principles and patterns, but use them judiciously. Design for current requirements without anticipating future requirements, and over complicating things. A well designed (i.e. loosely coupled and  more cohesive) system can easily lend itself to future changes, whilst keeping the system less complex. 

* Favor composition over inheritance.
* Don't Repeat Yourself (DRY principle). Code once and only once.
* Find what varies and encapsulate it.
* Code to an interface, and not to an implementation.
* Strive for loose coupling and high cohesion.

**Q.** In your experience, what are some of the common dilemmas you face when writing unit tests?  
**A.** [Hint: provide the dilemma and the tips to overcome the dilemma]

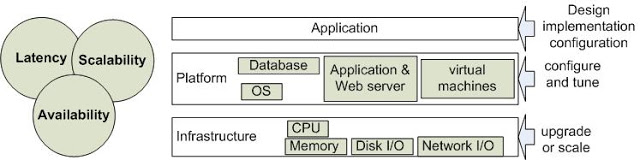
* **Whether to fix the code or the test.**When you write unit tests, sometimes you feel compelled to change your code just to facilitate the test. For example, when you need to test a private method or attribute. Doing so is a bad idea. If you ever feel tempted to make a private method public purely for testing purposes, don't do it. Testing is meant to improve the quality of your code, not decrease it. Having said this, in most cases, thinking about the test first in a TDD (.i.e. Test Driven Development) will help you refactor and write better code. Also, mock frameworks like **PowerMock**, let you mock private methods, static methods, constructors, final classes and methods, etc. Care must be taken in testing private methods. Inside a class you can end up with a lot of private methods manipulating instance or class variables without having them passed as parameters. This leads to high method coupling, and something not recommended. Instead use private methods that take explicit parameters and provide explicit return values.
* **Whether to use mocks or not.** The quality unit tests need to be repeatable. Some non-deterministic or environmental conditions can make your code fragile. Mocking these scenarios can make your code more robust. For example, a *CircusService*class can use a mock *CircusDao*to avoid test cases failing due to data related issues. Alternatively, the test cases can use frameworks like **DBUnit**to insert data  during test set up and  remove data during the test tear down to provide stable data for the test cases. Some test cases rely on Web service calls, and availability of these services are non-deterministic, and it makes more sense to mock those services.

* **How to test non-functional requirements?** For example, a non-functional requirement like 95 percent of all Web-based transactions should complete within 2 seconds. Your approach here would be to simply run the test code many times in a loop and compare the transaction times with target time, and keeping track of the number of passes and fails. If at the end of the test less than 95 percent of the transactions fail, then fail the test too. There are scenarios where @Test(timeout=5)  becomes handy to fail a test case that takes longer than 5 seconds to execute.

* **Testing multi-thread code**. This can be quite challenging  and sometimes an impossible task. If the tests are too complex, then review your code and design. There are ways to program for multi-threading that are more conducive for testing. For example, making  your objects immutable where possible, reducing the number of instances where multiple threads interact with the same instance, and using the *ExecutorService*(e.g. having a *SerialExecutorService*which is part of the **jconch**framework to execute the threads serially)  instead of the Thread class, etc. There are frameworks like **GroboUtils**and **MultithreadedTC**aiming to expand the testing possibilities of Java with support for multi-threading and hierarchical unit testing. The goal of  **jconch**project is to provide safe set of implementations for common tasks in multi-threaded Java applications. The *SerialExecutorService*class is one of them.

**Q.** When somebody says an application is running "very slow, what does he/she mean?  
**A.** He/she is generally referring to one of two performance attributes –  **latency**or **scalability**. Latency describes how long it takes for a given task to complete, whereas scalability describes how a program's performance gets impacted under increasing load. Building high performance applications require  

* Low latency –  for example,  low page loading times.
* High scalability – for example, serving increasing number of users without adversely impacting performance.
* High availability – for example, staying up 24x7, without going down due to memory leak or running out of connections to the database or LDAP server.

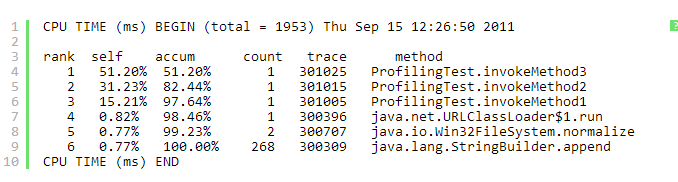
[](http://1.bp.blogspot.com/-nYhM7QD7UoU/T6hrKX3dG9I/AAAAAAAAAig/Hvhd4hUcraM/s1600/performance-overview.JPG)

As described in the above diagram, performance issues can occur due to a variety of reasons.  It can vary from a bad application, database, or infrastructure design that does not scale well to poorly tuned load balancer, virtual machine, application server, and from badly constructed SQL statements (e.g. a Cartesian join) and frequently back tracking regular expressions to thread contention issues caused by multiple threads simultaneously waiting for a long running locked method, block of code, or  database records. Memory and non-memory (e.g. sockets, connections, etc) leaks can also degrade performance by depriving your application of these scarce resources by either consuming more CPU cycles due to over working the garbage collection or by running out of memory or connections.    
  
**Q.**What tools do you need to profile a Java application?  
**A.** For troubleshooting Java applications, there are **basic tools**like vmstat, hprof, JConsole, JAMon, PerfAnal, etc to **more feature packed profiling tools** like VisualVM, Netbeans profiler, eclipse TPTP (i.e. Test & Performance Tools Platform), etc to**tools that can be used in production environment**like YourKit for Java, JProfiler for Java, etc to**tools for larger distributed and clustered systems**with large number of nodes like CA Wiley Introscope for Java, HP Sitescope and ClearStone for Java.  
  
**Q.**How would you go about profiling a Java application?  
**A.**The following example uses an **hprof**tool that comes with Java to determine the cpu times. The profiles are dumped to “java.hprof.txt” file when the program exits or a control character**Ctrl-\**or **Ctrl-Break** (WIN32) depending on platform is pressed. On Solaris OS and Linux a profile is also generated when a QUIT signal is received (kill -QUIT ). The process id can be determined with the “**jps**” command. The command shown below executes the "*ProfilingTest*" class with the hprof agent for measuring cpu times.

[?](http://java-success.blogspot.com.au/2012/05/java-performance-questions-and-answers.html)

|  |  |
| --- | --- |
| 1 | java -agentlib:hprof=cpu=times test.ProfilingTest |

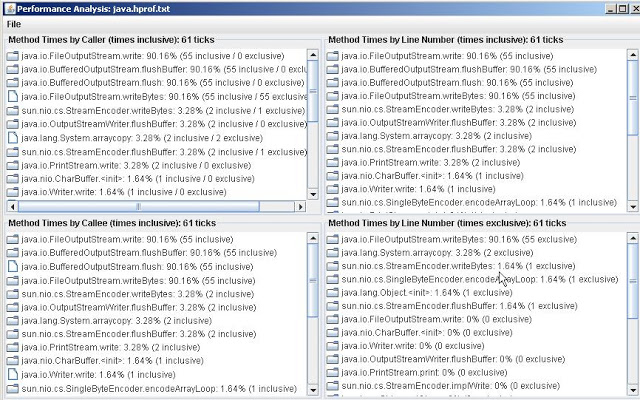
The profiled results will be dumped to a file named “java.hprof.txt”.



**PerfAnal**is a GUI based analysis tool to analyze the above results.

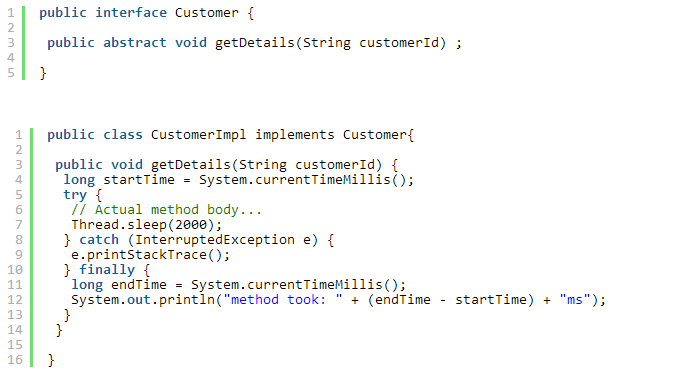
[?](http://java-success.blogspot.com.au/2012/05/java-performance-questions-and-answers.html)

|  |  |
| --- | --- |
| 1 | java -jar PerfAnal.jar  java.hprof.txt |

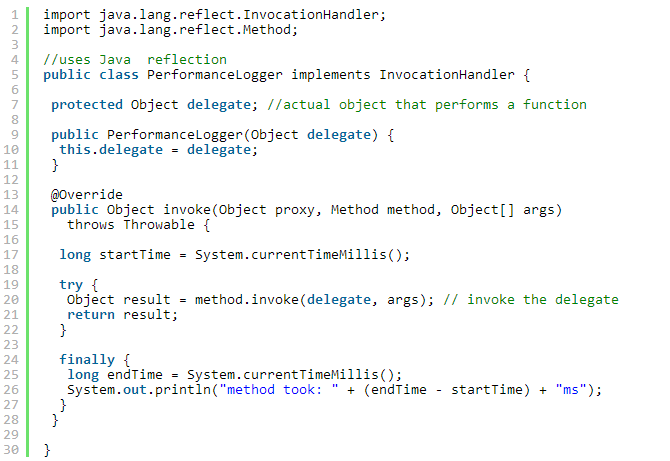
[](http://1.bp.blogspot.com/-r_ZvwQ-Va38/T54vJgq81LI/AAAAAAAAAYg/i0GxL3pxHtk/s1600/hprof-perf-analyzer.JPG)

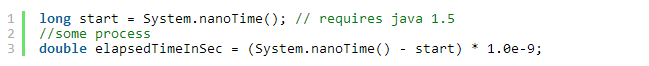
Similar approach can be used for profiling for memory usage.

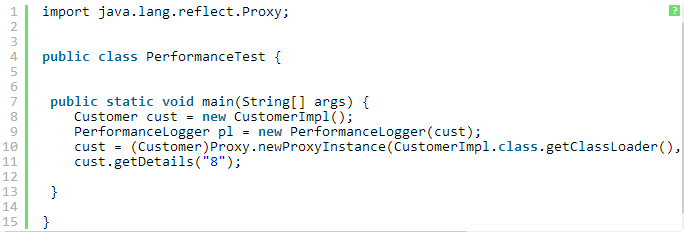
**Q.** How would you go about performance testing your Java application?  
**A.**  
  
**1.** Using a profiler on your running code. It should help you identify the bottlenecks. For example, jprofiler or Netbeans profiler. A profiler is a program that examines your application as it runs. It provides you with useful run time information such as time spent in particular code blocks, memory / heap, etc. In Java 6, you could use the JConsole.  
  
  
**2.** You also need to set up performance test scripts with JMeter to simulate the load. Most issues relating to performance bottlenecks, memory leaks, thread-safety, deadlocks, etc only surface under certian load. The performance testing scripts can be recorded while the application is being used and then manually refined. The tools like JMeter HTTP Proxy or Badboy software can be used to trace the script.   
  
**3.**You could provide a custom solution with the help of **AOP**(aspect oriented programming) or **dynamic proxies**to intercept your method calls. Dynamic proxies allow you to intercept method calls so you can interpose additional behavior between a class caller and its "callee".   
  
For example,**without AOP or dynamix proxy**



The above approach is**very intrusive**. Imagine if you have to add this to 30 to 40 other methods. How do you turn this metrics monitoring on and off without commenting and uncommenting your actual method. With dynamic proxy or Spring based AOP, you can alleviate this problem. Here is an example of the dynamic proxy class.



**Note**: If you are using Java 5 or higher use the**System.nanoTime( )**instead of System.currentTimeMillis(). For example,  
  
  
The test class that tests the above code



cust = (Customer)Proxy.newProxyInstance(CustomerImpl.class.getClassLoader(), new Class[] {Customer.class}, pl);

With this approach, you don't need to put the System.currentTimeMillis( ) in your delegate classes. It only needs to be in PerformanceLogger, and it will intercept actual calls to the delegates to print performance metrics. If you want to make changes as to how the metrics are collected or want to turn off System.currentTimeMillis( ), you will only have to do it in one class, that is the dynamic proxy class *PerformanceLogger*.  
  
If you are using Spring, you could use the AOP based interceptors. The [AOP based deadlock retry using Spring](http://java-success.blogspot.com.au/2012/02/spring-and-hibernate-interview.html)is using a very similar approach for dead lock retry.  
  
**Note**:I have been in interviews where the interviewer was more interested in the custom solution by asking  
  
**Q.** If you did not have a profiling tool, how would you go about gathering performance metrics?  
**A**. **AOP**or **dynamic proxy** based solution as discussed above. You could also mention the built-in command-line and graphical tools that come with Java  like JConsole, jstack, jmap, hprof, vmstat, etc. But the main focus must be via interceptors.  
  
**Design pattern**: If you are asked to describe or talk about a design pattern, you could mention this dynamic proxy class as a **proxy design pattern**. Many pick either singleton or factory design pattern. It would be nicer to pick something other than these two common patterns. Some interviewers specifically ask you to pick anything except factory and singleton design pattern.

**Q.** What tips do you give someone regarding Java performance?  
**A.**  
  
  
**1. Don't compromise on design:** You should not compromise on architectural principles for just performance. You should make effort to write architecturally sound programs as opposed to writing only fast programs. If your architecture is sound enough then it would allow your program not only to scale better but also allows it to be optimized for performance if it is not fast enough. If you write applications with poor architecture but performs well for the current requirements, what will happen if the requirements grow and your architecture is not flexible enough to extend and creates a maintenance nightmare where fixing a code in one area would break your code in another area. This will cause your application to be re-written. So you should think about extendibility (i.e. ability to evolve with additional requirements), maintainability, ease of use, performance and scalability (i.e. ability to run in multiple servers or machines) during the design phase. List all possible design alternatives and pick the one which is conducive to sound design architecturally (i.e. scalable, easy to use, maintain and extend) and will allow it to be optimized later if not fast enough. Once you get the correct design, then measure with a profiler and optimize it.  
  
**2**. **Be aware of the death by thousand-cuts:**Having said**not to compromise on the design**,  one needs to be mindful of  performance inefficiencies that can creep in throughout the software development. For example, an inefficient method being called 50-100 times can adversely impact performance. A real-life example would be a JSF application that invokes its life-cycle methods many times. So, having a long-running subroutine within the life-cycle method might end up calling it more than once. So, know your best practices and potential pitfalls. Here are a few things to keep in mind. 

* **Use immutable objects where applicable**. Immutable objects are inherently thread-safe and can also be reused. A good candidate for implementing the flyweight design pattern. The following Java method is an example of flyweight.

[?](http://java-success.blogspot.com.au/2012/03/java-interview-questions-and-answers.html)

|  |  |
| --- | --- |
| 1 | Integer.valueOf(String s) |

        It keeps some amount of the created Integers internally, so, when you pass the String that you have  
        passed before - it returns you an existing instance.

* **Check your regexes and SQL** queries for backtracking and Cartesian joins respectively.

* **Define your objects and variables with the right scopes.** The variables need to be defined at the lowest scope possible (e.g local --> instance --> static)
* **Use proven libraries, frameworks, built-in algorithms, and data structures** as opposed to creating your own. For example, when handing concurrency use java.util.concurrent package.

**3**. Always have a performance focus by developing proper load testing scripts and benchmarks. The non-functional requirements should cover the relevant SLAs (i.e. Service Level Agreements). **Tune your application server, database server, application, etc where required** with proper bench marking and load testing scripts. The mission critical applications must have run time metrics gathering in place commercial tools. There are tools that can be used in production environment like YourKit for Java, JProfiler for Java, etc and for larger distributed and clustered systems with large number of nodes there are profilers like CA Wiley Introscope for Java, ClearStone for Java, and HP Performance managers. These tools are handy for proactive detection and isolation of server/application bottlenecks, historical performance trend tracking for capacity planning, and real-time monitoring of system performance.  
  
**Q.**When designing your new code, what level of importance would you give to the following attributes? Rank the above attributes in order of importance?     
  
**--** **Performance  
-- Maintainability  
-- Extendibility   
-- Ease of use  
-- Scalability**  
  
**Rank the above attributes in order of importance?**  
  
**A.** This is an open-ended question. There is no one correct order for this question. The order can vary from application to application, but typically if you write **1** - extendable, **2** - maintainable and **3** – ease of use code with some high level performance considerations, then it should allow you to  optimize/tune for **4** - performance and **5**- scalability. But if you write some code, which only perform fast but not flexible enough to grow with the additional requirements, then you may end up re-writing or carrying out a major revamp to your code.   
  
**Note**: These types of questions have no right or wrong answers, but can reveal a lot about your experience, passion, and attitude towards building a quality software. Good software developers and architects are opinionated based on their past experiences.