







**Caching in Hibernate**

Its got two levels of caching. The first level the implicit Session caching - you access the same object in the same session you'll see a cached version (which is safe, because a Session is transaction-scoped). Forcing Session caching to be ignored can be done by calling flush() on the Session.

The second level cache requires you pick a third party implementation of a clusterable cache - EHCache, [JBoss](http://www.coderanch.com/forums/f-63/JBoss" \t "_new" \o "JBoss Forum) Cache etc. or write your own. You implement it by marking objects as cacheable in the mapping file and adding the hibernate.cache.provider\_class attribute to hibernate.properties (or however you choose to configure the SessionFactory).

**Improve performance in Web portal applications by caching objects**

Web applications are typically accessed by many concurrent users. Usually, the application's data is stored in a relational database or filesystem, and it takes time and costs overhead to access these data sources. Database-access bottlenecks can slow down or even crash the application if it receives too many simultaneous requests. Object caching is one technique that overcomes this problem.

# [Design Documents (High Level and Low Level Design Documents)](http://stackoverflow.com/questions/10297869/design-documents-high-level-and-low-level-design-documents)

High level design involves decomposing system into modules, and representating the interfaces and invocation relationships among modules. A HLD is referred to as software architecture.

LLD also known as detailed design is used to design internals of the individual modules identified during HLD i.e Data structure and algorithms of the modules are designed and documented.

Now, HLD and LLD are actually used in traditional Approach (Function-Oriented Software Design) whereas in OOAD, System is seen as a set of objects interacting with each other.

going by the above definitions, A high-level design document will usually include a high-level architecture diagram depicting the components, interfaces and networks that need to be further specified or developed. The document may also depict or otherwise refer to work flows and/or data flows between component systems.

Class diagrams with all the methods and relation between classes comes under LLD. Programs specs are covered under LLD.LLD describes each and every module in an elaborate manner so that the programmer can directly code the program based on this.There will be at least 1 document for each module and there may be more for a module. The LLD will contain: - detailed functional logic of the module in pseudocode - database tables with all elements including their type and size - all interface details with complete API references(both requests and responses) - all dependency issues -error message listings - complete input and outputs for a module.

## The basics

As mentioned earlier, the purpose of the class diagram is to show the types being modeled within the system. In most UML models these types include:

* a class
* an interface
* a data type
* a component.

Application profiling in java

# [Performance profiler for a java application](http://stackoverflow.com/questions/3035601/performance-profiler-for-a-java-application)

# <http://stackoverflow.com/questions/3035601/performance-profiler-for-a-java-application>

I need to optimise a Java application. It makes some 3rd party calls. I need some good tool to accurately measure the time taken by individual API calls. To give an idea of complexity- the application takes a data source file containing 10 lakh rows, and it takes around one hour to complete the processing. As a part of processing , it makes some 3rd party calls (including some network calls). I need to identify which calls are taking more time then others, and based on that, find out a way to optimise the application.

I can recommend [JVisualVM](https://visualvm.dev.java.net/). It's a great monitoring / profiling tool that is bundled with the Oracle/Sun JDK. Just fire it up, connect to your application and start the CPU-profiling. You should get great histograms over where the time is spent.

[Getting Started with VisualVM](https://visualvm.dev.java.net/gettingstarted.html) has a great screen-cast showing you how to work with it.

Screen shot:

I use JVMMonitor which is completely integrated with Eclipse, it's simple to use and it's up-to-date (latest version is 3.7.1 compatible with Eclipse Indigo)

It's available from the Eclipse Marketplace, or at the following site: <http://www.jvmmonitor.org/index.html>

I recommend it.

JVM Monitor is a Java profiler integrated with Eclipse to monitor CPU, threads and memory usage of Java applications.

**Enterprise Architecture**

An enterprise architecture (EA) is a conceptual blueprint that defines the structure and operation of an organization.

Enterprise architecture as containing four points-of-view, called the

1. **Business** perspective,
2. **Application** perspective,
3. **Information** perspective,
4. **Technology** perspective.

**The business perspective** defines the processes and standards by which the business operates on a day-to-day basis. **The application perspective** defines the interactions among the processes and standards used by the organization. The **information perspective** defines and classifies the raw data (such as document files, databases, images, presentations, and spreadsheets) that the organization requires in order to efficiently operate. **The technology perspective** defines the hardware, operating systems, programming, and networking solutions used by the organization.