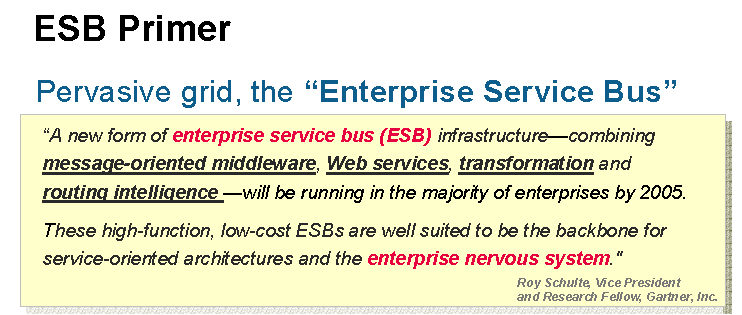
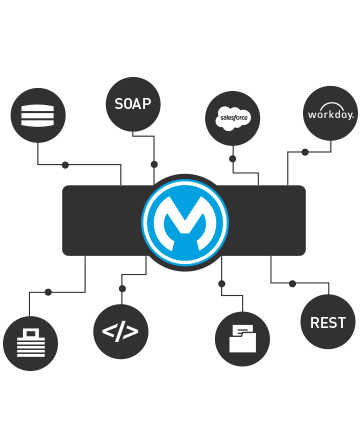
1. What is an ESB?

An Enterprise Service Bus (ESB) is fundamentally an architecture. It is a set of rules and principles for integrating numerous applications together over a bus-like infrastructure. [ESB](https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb) products enable users to build this type of architecture, but vary in the way that they do it and the capabilities that they offer. The core concept of the [ESB architecture](https://www.mulesoft.com/resources/esb/why-use-esb) is that you integrate different applications by putting a communication bus between them and then enable each application to talk to the bus. This decouples systems from each other, allowing them to communicate without dependency on or knowledge of other systems on the bus. The concept of ESB was born out of the need to move away from point-to-point integration, which becomes brittle and hard to manage over time. Point-to-point integration results in custom integration code being spread among applications with no central way to monitor or troubleshoot. This is often referred to as "spaghetti code" and does not scale because it creates tight dependencies between applications.



## Why use an ESB?



Increasing organizational agility by reducing time to market for new initiatives is one of the most common reasons that companies implement an ESB as the backbone of their IT infrastructure. An ESB architecture facilitates this by providing a simple, well defined, "pluggable" system that scales really well. Additionally, an ESB provides a way to leverage your existing systems and expose them to new applications using its communication and transformation capabilities.

## Implementation

The ESB architecture has some key principles that allow for business agility and scale. The key focus is to decouple systems from each other while allowing them to communicate in a consistent and manageable way.

* The "bus" concept decouples applications from each other. This is usually acheived using a messaging server like JMS or AMQP.
* The data that travels on the bus is a canonical format and is almost always XML.
* There is an "adapter" between the application and the bus that marshals data between the two parties.
* The adapter is responsible for talking to the backend application and transforming data from the application format to the bus format. The adapter can also perform a host of other activities such as message routing transaction management, [security, monitoring](https://www.mulesoft.com/platform/soa/mule-enterprise-security), error handling, etc.
* ESBs are generally stateless; the state is embedded in the messages passing through the bus.
* The canonical message format is the contract between systems. The canonical format means that there is one consistent message format traveling on the bus and that every application on the bus can communicate with each other

## Integration core principles

Let's take a look at how an ESB architecture maps to our five core integration principles:

* **Orchestration**: Composing several existing fine-grained components into a single higher order composite service. This can be done to achieve appropriate "granularity" of services and promote reuse and manageability of the underlying components.
* **Transformation**: Data transformation between canonical data formats and specific data formats required by each ESB connector. An example of this would be transforming between CSV, Cobol copybook or EDI formats to either SOAP/XML or JSON. Canoncial data formats can greatly simplify the transformation requirements associated with a large ESB implementation where there are many consumers and providers, each with their own data formats and definitions.
* **Transportation**: Transport protocol negotiation between multiple formats (such as HTTP, JMS, JDBC). Note: Mule treats databases like another "service" by making JDBC just another transport (or endpoint) where data can be accessed.
* **Mediation**: Providing multiple interfaces for the purpose of a) supporting multiple versions of a service for backwards compatibility or alternatively, b) to allow for multiple channels to the same underlying component implementation. This second requirement may involve providing multiple interfaces to the same component, one legacy interface (flat file) and one standards compliant (SOAP/XML) interface.
* **Non-functional consistency**: For a typical ESB initiative, this can include consistency around the way security and monitoring policies are applied and implemented. Additionally the goals of scalability and availability can be achieved by using multiple instances of an ESB to provide increased throughput (scalability) and eliminate single-points-of-failure (SPOFs), which is the key objective for highly available systems.

## Choosing an ESB platform

There are many [ESB platforms](https://www.mulesoft.com/resources/esb/mule-esb-integration-platform) out there, from big proprietary vendors to niche and open source vendors. On paper, there are lots of similarities. Here are some points to consider when making an ESB selection.

### Lightweight

Mule is the most [lightweight integration platform](https://www.mulesoft.com/platform/api) available, with the fully loaded distribution weighing in at 40 MB. It is modular by design so you can strip out unwanted modules if you need to reduce the footprint. We don't see "lightweight" as just about size either; it is also the cost of making changes to existing integrations and the amount of heavy lifting you need to do to make changes. The Mule run-time offers modularization and super-fast hot deployment as well as a configuration model that makes it easy to re-order and add/change functionality.

### Not just mediation

Most vendors think of an ESB as purely mediation between systems and have separate products for hosting business logic and publishing services. We see this as unnecessary complexity. Mule provides a light and scalable service container for publishing REST and SOAP services. Since Mule integrates tightly with Spring, it means developers can also leverage the capabilities of Spring to implement business logic.

### Accessible - any developer can learn Mule

Mule uses common tools that all Java developers are familiar with, such as Maven, Eclipse, JUnit and Spring. Mule uses an XML configuration model (similar to Spring) to define logic, and custom code can be written in a variety of languages, including Java, Groovy, JavaScript, Ruby or Python. Also, MuleStudio helps new developers get up to speed quickly with a graphical development environment.

### Scaling up, scaling down

Mule was designed for horizontal scale on commodity hardware - no need for big iron. Mule's runtime is easily embeddable into an application. It can also be embedded in your application server such as Tomcat, JBoss or WAS or directly in your application. More importantly, Mule provides JUnit support so that it can be embedded in a JUnit test case. This is powerful because it means you can create repeatable unit tests for integrations that will run on a developer laptop and can be incorporated into a continuous build.

### Message agnostic

A powerful feature of Mule is that the container is message agnostic. This means it does not force XML messages on its users. While XML is common, there are many scenarios where you will want to use JSON, flat files, Cobol Copybooks, binary and file attachments, streams and Java objects. Our graphical Data Mapper is equally not fussy about the data that can be mapped. What's more, Mule streaming allows developers to process large messages efficiently.

### Cloud ready

If you'd rather leave the application architecture, hosting and monitoring of your integration to the integration experts then CloudHub™ is for you. CloudHub is an [integration Platform as a Service](https://www.mulesoft.com/platform/enterprise-integration) (iPaaS) that gets you up and running in minutes. CloudHub offers a multi-tenanted, elastic platform with [connectivity to 150+ SaaS](https://www.mulesoft.com/platform/saas), Social Media and infrastructure services and the ability to connect to your on-premise applications. CloudHub applications run on Mule standalone and vice versa. This means that whether you are deploying on-premise or the cloud, there are no new concepts to learn and the developer experience is the same. No need to learn a new way of doing things.

## Summary

Most organizations want to increase agility by reducing time to market for new initiatives. ESBs promote this objective by implementing a simple, well defined, "pluggable" system that scales really well. Here at MuleSoft we understand that an ESB architecture is exactly that: an architecture and not simply a product you can buy off the shelf. It encompasses not only infrastructure but application design as well. Mule [ESB Enterprise](https://www.mulesoft.com/platform/soa/mule-esb-enterprise) serves as a great foundation upon which to build such an architecture.

1. JMS Basic :

JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication.JMS is also known as a messaging service.

Understanding Messaging

Messaging is a technique to communicate applications or software components. JMS is mainly used to send and receive message from one application to another.

Requirement of JMS

Generally, user sends message to application. But, if we want to send message from one application to another, we need to use JMS API.

Consider a scenario, one application A is running in INDIA and another application B is running in USA. To send message from A application to B, we need to use JMS.

Advantage of JMS

1) **Asynchronous:** To receive the message, client is not required to send request. Message will arrive automatically to the client.

2) **Reliable:** It provides assurance that message is delivered.

Messaging Domains

There are two types of messaging domains in JMS.

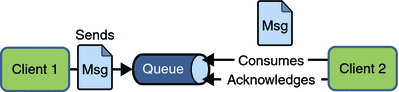
1. Point-to-Point Messaging Domain
2. Publisher/Subscriber Messaging Domain

1) Point-to-Point (PTP) Messaging Domain

In PTP model, one message is **delivered to one receiver** only. Here, **Queue** is used as a message oriented middleware (MOM).

The Queue is responsible to hold the message until receiver is ready.

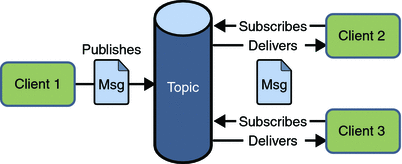
In PTP model, there is **no timing dependency** between sender and receiver.



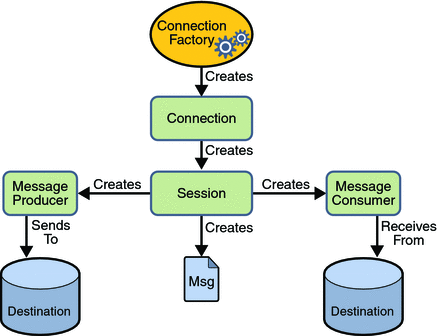
2) Publisher/Subscriber (Pub/Sub) Messaging Domain

In Pub/Sub model, one message is **delivered to all the subscribers**. It is like broadcasting. Here, **Topic** is used as a message oriented middleware that is responsible to hold and deliver messages.

In PTP model, there is **timing dependency** between publisher and subscriber.



JMS Programming Model



32

Point to Point means message(s) is sent from one application(producer or sender) to another application(consumer/receiver) via a queue. There can be more than one consumer listening on a queue but only one of them will be get the message. Hence it is Point to Point or One to One.

On the other hand Publish/Subscribe is another messaging model where a message(or publication as it is commonly called) is sent to multiple consumers(or subscribers) through a topic. The topic is the link between publisher and subscriber. The subscribers may or may not acknowledge the published message. Implementations like JMS acknowledge the message the messaging providers but not the sender of the message. Publications will be received by all subscribers, durable and non-durable. Any new subscribers on the same topic will not get the publication unless it is a Retained publication.

JMS Queue Example

To develop JMS queue example, you need to install any application server. Here, we are using **glassfish3** server where we are creating two JNDI.

1. Create connection factory named **myQueueConnectionFactory**
2. Create destination resource named **myQueue**

After creating JNDI, create server and receiver application. You need to run server and receiver in different console. Here, we are using eclipse IDE, it is opened in different console by default.

#### 1) Create connection factory and destination resource

Open server admin console by the URL **http://localhost:4848**

Login with the username and password.

Click on the **JMS Resource -> Connection Factories -> New**, now write the pool name and select the Resource Type as QueueConnectionFactory then click on ok button.

**Steps for sending Message over Queue**

//Create and start connection using connectionFactory from JNDI lookup

//Create queue session using connection object.

//get the Queue object from lookup

//Create QueueSender object  from session.

//Create TextMessage object using session.

//Write message on above

//send message using QueueSender.

//Connection close

**Steps for receiving Message over Queue**

//Create and start connection using connectionFactory from JNDI lookup

//Create queue session using connection object.

//get the Queue object from lookup

//Create QueueReceiver  object  from session.

//Create listener object that implements MessageListener

// register the listener object with receiver

JMS Topic Example

It is same as JMS Queue, but you need to change Queue to Topic, Sender to Publisher and Receiver to Subscriber.

You need to create 2 JNDI named **myTopicConnectionFactory** and **myTopic**.

**Steps for sending Message over Topic**

//Create and start connection using TopicconnectionFactory from JNDI lookup

//Create Topic session using connection object.

//get the Topic object from lookup

//Create TopicPublisher object  .

//Create TextMessage object using session.

//Write message on above

//send message using Publisher.

//Connection close

**Steps for receiving Message over Topic**

//Create and start connection using TopicConnectionFactory from JNDI lookup

//Create Topic session using connection object.

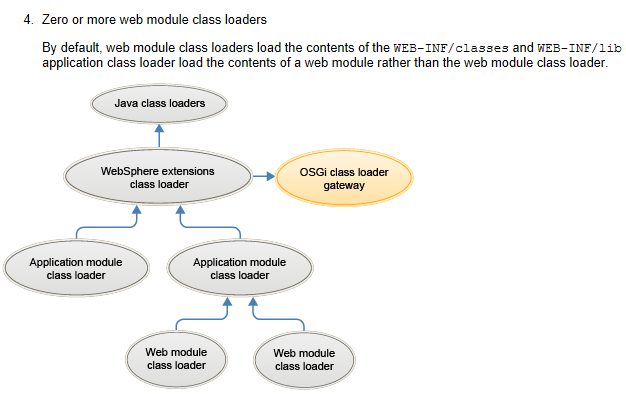
//get the Topic object from lookup

//Create TopicSubscriber object  from session.

//Create listener object that implements MessageListener

// register the listener object with subscriber

1. Class Loader : <http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.iseries.doc/ae/crun_classload.html>



### What is Java ClassLoader?

We know that Java Program runs on [Java Virtual Machine](https://www.journaldev.com/546/difference-jdk-vs-jre-vs-jvm) (JVM). When we compile a Java Class, it transforms it in the form of bytecode that is platform and machine independent compiled program and store it as a .class file. After that when we try to use a Class, Java ClassLoader loads that class into memory.

There are three types of built-in ClassLoader in Java:

1. **Bootstrap Class Loader** – It loads JDK internal classes, typically loads rt.jar and other core classes for example java.lang.\* package classes
2. **Extensions Class Loader** – It loads classes from the JDK extensions directory, usually $JAVA\_HOME/lib/ext directory.
3. **System Class Loader** – It loads classes from the current classpath that can be set while invoking a program using -cp or -classpath command line options.

Java ClassLoader are hierarchical and whenever a request is raised to load a class, it delegates it to its parent and in this way uniqueness is maintained in the runtime environment. If the parent class loader doesn’t find the class then the class loader itself tries to load the class.

### Why write a Custom ClassLoader in Java?

Java default ClassLoader can load files from local file system that is good enough for most of the cases. But if you are expecting a class at the runtime or from FTP server or via third party web service at the time of loading the class then you have to extend the existing class loader. For example, AppletViewers load the classes from remote web server.

### How does Java ClassLoader Work?

When JVM requests for a class, it invokes function of the ClassLoader by passing the fully classified name of the Class.

loadClass function calls for method to check that the class has been already loaded or not. It’s required to avoid loading the class multiple times.

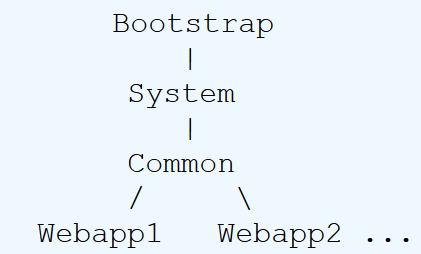
If the Class is not already loaded then it will delegate the request to parent ClassLoader to load the class.

If the parent ClassLoader is not finding the Class then it will invoke findClass() method to look for the classes in the file system.

**Tomcat Class Loader**

In normal Java applications when a classloader is asked to load a class it forst delegates the request to it's parent class loader and then loads it if parent class loaders cannot find the requested class.

For web application servers this slightly differs. There are generally different class loader for each web app deployed in a web application server like tomcat. For Tomcat it looks like below -



So for web apps class loading resource happens in following order -

1. Bootstrap classes of your JVM (Core java classes)
2. /WEB-INF/classes of your web application
3. /WEB-INF/lib/\*.jar of your web application
4. System class loader classes (Tomcat / Classpath specific classes)
5. Common class loader classes (classes common to all web apps)

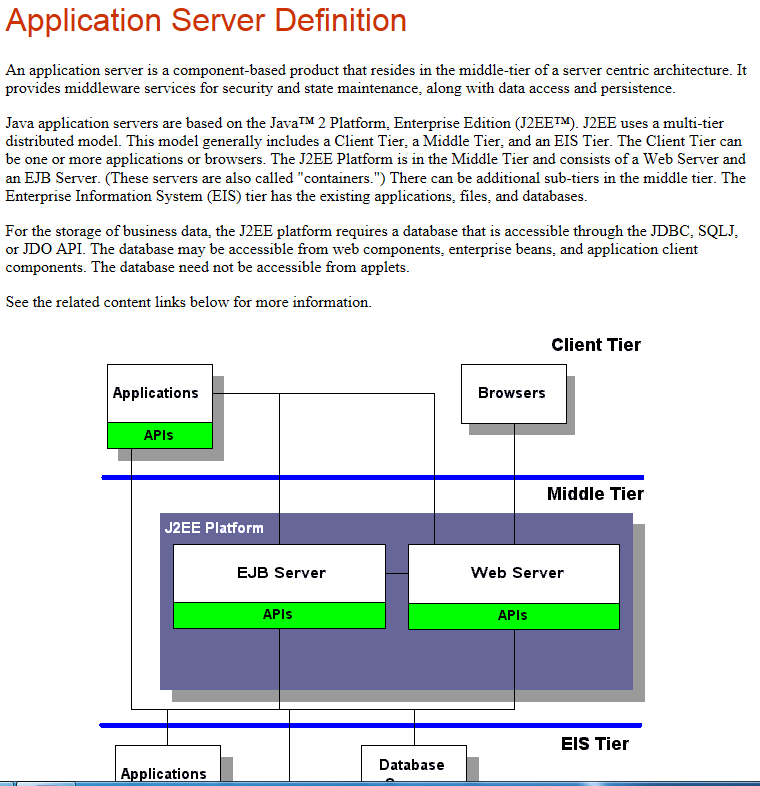
But note if web application class loader is configured with delegate="true" then order is changed -

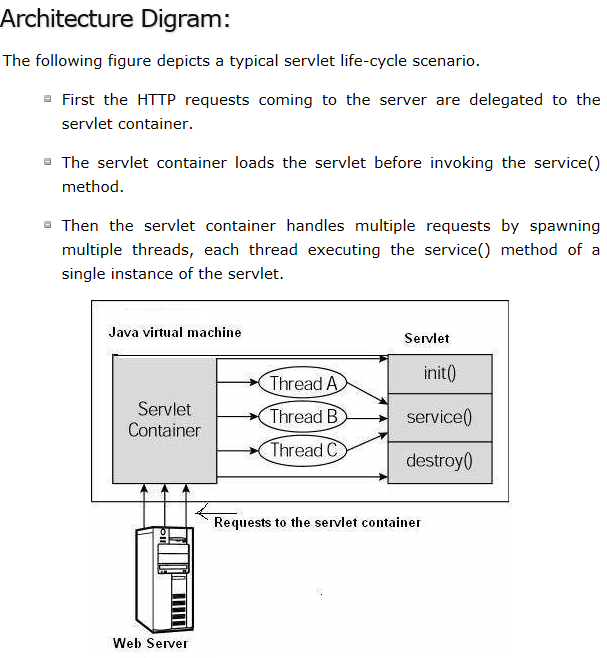
1. Bootstrap classes of your JVM (Core java classes)
2. System class loader classes (Tomcat / Classpath specific classes)
3. Common class loader classes (classes common to all web apps)
4. /WEB-INF/classes of your web application
5. /WEB-INF/lib/\*.jar of your web application

For more details you can check Apache Tomcat's [Class Loader HOW-TO](http://tomcat.apache.org/tomcat-7.0-doc/class-loader-howto.html) page.

<https://www.journaldev.com/349/java-classloader>

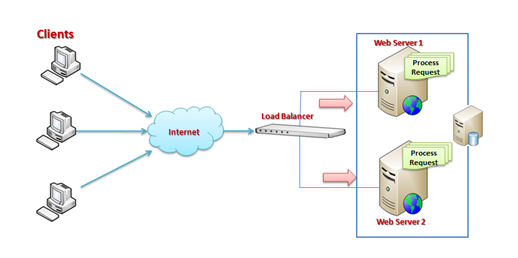
<https://www.baeldung.com/java-classloaders>



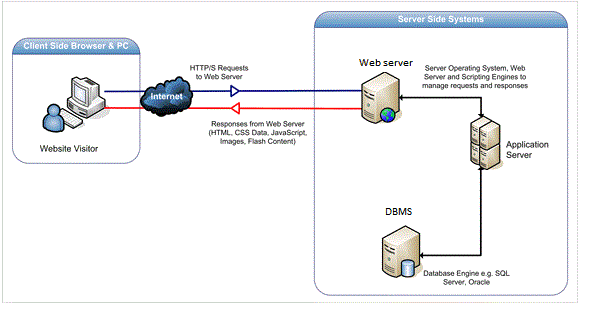


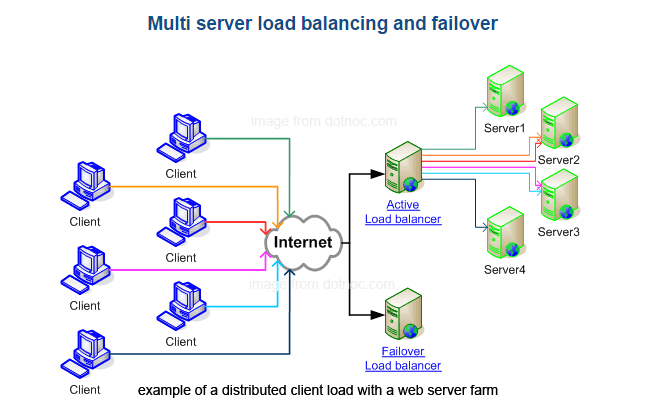
1. SDLC : <https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm>
2. HTTP Server, Application Server and Load Balancer.

http://stackoverflow.com/questions/936197/what-is-the-difference-between-application-server-and-web-server



IBM HTTP **Server** (**IHS**) is a web **server** based on the Apache Software Foundation's HTTP **Server** that runs on AIX, HP-UX, Linux, Solaris, JADE, Windows NT, IBM i and z/OS. It is available for download and use free of charge but without IBM suppor





http server is used for routing web requests to a WAS in a typical enterprise scenario. If you are building small apps, and do not need ejbs, just use the http server. if you need ejbs, just use WAS. However, if you need functionality like routing requests to WAS because you do not want the user to hit the WAS directly for security reasons, then use HTTP in front of WAS

All the answers are good so let me give my cents , Websphere as other java applications, was made to work with dinamic contents . apache or IHS that is apache IBM plugins can handle with all kinds of http/https requests , static , dinamic , proxy ,cache at the client tags, balance and so on ... just using apache over java aplications is not the best , the best is allow apache handle all static downloads without call it from application server , like images and fixed htmls and left application server handle the just dinamic contents . at this way we can optimize the environment , so development should create a pattern to dispose static and dinamic at different virtual direct like /jsp for dinamic and the rest as static , if not apache cache can be used instead for not overload the application server . to make the story short , apache is so powerful ans can help a lot the application server sparing the work , a simple and effective usage is done to forward the port using default port 80/443 for http/https instead usual high ports on java application servers , make the apache deliver https is a best option too , instead left java application server care of it , so there is a lot of reasons to use Apache or IHS in front of application servers

1. Difference between Abstraction and Encapsulation in Java - OOPS

**Abstraction vs Encapsulation – Java OOPS**

Abstraction and Encapsulation in Java are two important [Object oriented programming concept](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html) and they are completely different to each other. Only similarity between Abstraction and Encapsulation is that they are OOPS concept, other than that they mean two different things. Abstraction represent taking out the behavior from How exactly its implemented, one example of [abstraction in Java](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html) is interface while Encapsulation means hiding details of implementation from outside world so that when things change no body gets affected. One example of [Encapsulation in Java](http://javarevisited.blogspot.sg/2012/03/what-is-encapsulation-in-java-and-oops.html) is private methods; clients  don't care about it, You can change, amend or even remove that method  if that method is not encapsulated and it were public all your clients would have been affected. Apart from this main difference in behavior here are couple of more *differences between Abstraction and Encapsulation in Java*.

## Abstraction vs Encapsulation in Java

Here are some of the main differences between Abstraction vs Encapsulation in Java and OOPS(Object Oriented programming) concept. Abstraction and Encapsulation along with [Inheritance](http://java67.blogspot.sg/2012/08/what-is-inheritance-in-java-oops-programming-example.html) and [polymorphism](http://javarevisited.blogspot.sg/2011/08/what-is-polymorphism-in-java-example.html) forms basis of Object oriented programming in Java.

a) First difference between Abstraction and Encapsulation is that, Abstraction is implemented in Java using [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) and abstract class while Encapsulation is implemented using [private](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html), package-private and protected access modifier.

b) Encapsulation is also called data hiding.

c) Design principles "[programming for interface than implementation](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html)" is based on abstraction and "encapsulate whatever changes" is based upon Encapsulation.

What is Need of abstract class? Any real time example?

A concrete example of an abstract class would be a class called Animal. You see many animals in real life, but there are only kinds of animals. That is, you never look at something purple and furry and say "that is an animal and there is no more specific way of defining it". Instead, you see a dog or a cat or a pig,all animals. The point is, that you can never see an animal walking around that isn't more specifically something else (duck, pig, etc.). The Animal is the abstract class and Duck/Pig/Cat are all classes that derive from that base class. Animals might provide a function called "Age" that adds 1 year of life to the animals. It might also provide an abstract method called "IsDead" that, when called, will tell you if the animal has died. Since IsDead is abstract, each animal must implement it.So, a Cat might decide it is dead after it reaches 14 years of age, but a Duck might decide it dies after 5 years of age.

Source: Stackoverflow(http://stackoverflow.com/questions/1894453)