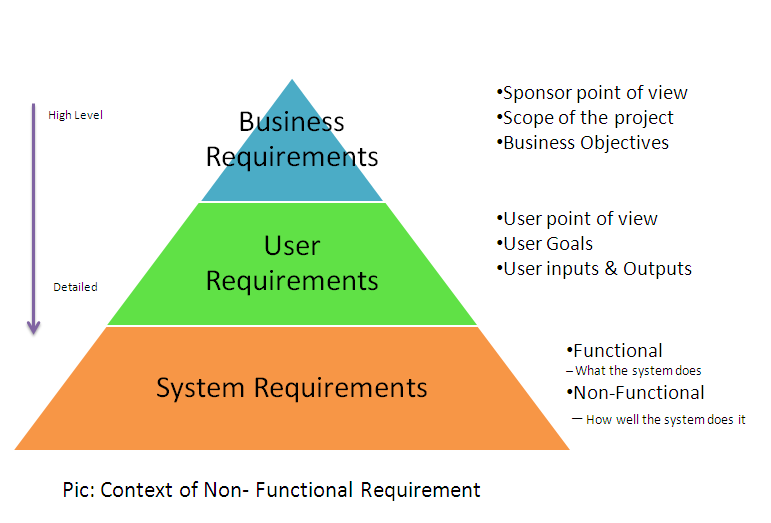
**2015 Revisited**

**Difference between Functional and Non-Functional Requirements in Software development**Gathering the correct and complete requirement is one of the most important thing in software development. Incorrect and incomplete requirement are main reason of why project fails. If you are in software development, you may have come across terms like *functional*and *non-functional* requirement. If you are wondering why a prototype take a 2 weeks but actual applicationdevelopment requires around 3 to 4 months of development; think of non-functional requirement. When someone told you to build a software, what they tell you is what that software should do e.g. allow you to trade on certain market, but they don't tell you about security, performance, load and other stuff, this is what I called non functional requirement. Obvious *difference between functional and non-functional requirement* is, former is specified by its users, business analyst and its part of software's feature list, for example functional requirement of a trading application is to receive order, enrich, transform and send it to Stock Exchange, but non-functional requirement is not specified by user, its rather thought by Software architect, Subject Matter Experts, Technical lead and Support peoples. For example, for this same trading application non-functional requirement could be Failover and Recovery, [logging](http://javarevisited.blogspot.sg/2011/05/top-10-tips-on-logging-in-java.html), auditing, latency and other performance characteristic that, application should be able to run continuous, can process 5K orders per second etc. Support people can also ask functionality require to add user, give access, revoke access, monitoring etc. Every application in software development has one or other kind of non-functional requirements. In this article, we will learn more about difference between functional and non-functional requirements in software and product development.  
Functional vs Non-functional Requirements

[](http://4.bp.blogspot.com/-gxzK76qurDU/VL-2lTvGkPI/AAAAAAAACZs/s9ZW-U_7Z9s/s1600/Difference+between+Functional+and+Non-Functional+Requirement.PNG)

Here are couple of differences between functional and non-functional requirement in software development process :  
  
1) Functional requirement is specified by User, while non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.  
  
2) Functional requirement is also the activity System must perform, on other hand non-functional are depending upon criticality of application. For example, if your application is not critical and you can live with downtime, you may not need to develop complex failover and disaster recovery code, reducing your application total development time.  
  
3) Functional requirements defines a software’s functionality i.e. what can they do, while non-functional requirements defines, other things which is not required by user but requirement by service provider or software itself e.g. logging is a non-functional requirement to support an application, not directly used by user but essential to troubleshoot any issue in production environment.  
  
4) Non-functional requirements are sometimes defined in terms of metrics (something that can be measured about the system) to make them more tangible.  
  
5) Non-functional requirements may also describe aspects of the system that don't relate to its execution, but rather to its evolution over time (e.g. maintainability, extensibility, documentation, etc.).  
  
That's all about difference between functional and non-functional requirement in field of software development. While estimating total development time, always think about non-functional requirements and highlight them as soon as possible. They are used to set expectation correctly from Managers and Users perspective. Most of the time, your manager think a feature can be done in days, without giving any thought to non functional requirement attached to it, it’s your, a software programmer's job to bring that up.

**Java Clone Tutorial Part 2 - Example to Override with Mutable field**This is the second part of Java tutorial on Cloning, In [first part](http://javarevisited.blogspot.com/2013/09/how-clone-method-works-in-java.html) we have seen *how clone method works in Java* with a simple example of cloning object, with primitives and Immutable. In this tutorial, we will take one step further and override clone method for creating clone of object with *mutable field*. In our case mutable field is a Collection here, to be precise a List. Since default implementation of clone() method only does shallow copy of objects, it can create issue, if original object contains mutable object or Collection classes. In our example, we have a class called Programmer, with String name, int age and List of Certifications. When we override clone() method inside Programmer class, we need to explicitly take care of this List, otherwise, both original and cloned object will point to same Collection in Java heap, which means, any change e.g. adding a new Certification in original object will also reflect in cloned object or vice-versa. Since an object should be independent of it's clone, we need to fix this issue by applying deep cloning techniques. Along with *this example of overriding clone in Java*, In this Java clone tutorial part 2, we will also take a look at some [Java best practices](http://javarevisited.blogspot.sg/2013/05/ava-tips-and-best-practices-to-avoid-nullpointerexception-program-application.html) for implementing correct clone method, disadvantages and shortcomings of cloning in Java and in particular clone method and finally, when to use clone in Java.  
**How to Override Clone method in Java**

There are some guidelines mentioned about overriding clone method in Java, e.g. call should be delegated to super.clone(), by keeping that in mind, following steps should be followed while overriding clone() in Java :  
  
1) Let the class, which supports cloning implements Cloneable interface. (failure to do this will result in CloneNotSupportedException).  
  
2) Override protected clone() method from java.lang.Object class.  
  
3) In overridden clone(), first call super.clone() to get the shallow copy of object.  
  
4) If your class contains any Collection or Mutable object, than [deep copy](http://java67.blogspot.sg/2013/05/difference-between-deep-copy-vs-shallow-cloning-java.html) of those field. Like in our example, Programmer class contains List in it's certification field, when super.clone() will return, both original and cloned object will point to same object. To fix this, we reassign certification fields of clone object by explicitly copying data, as shown in following line :

clone.certifications = new ArrayList(certifications); //deep copying

5) Depending upon object, you may call it's clone method e.g. in case of java.util.Date or recursively copy it's data into new field. This whole process is known as deep copying. See here to know [how to deep copy Java Collection](http://javarevisited.blogspot.sg/2014/03/how-to-clone-collection-in-java-deep-copy-vs-shallow.html).  
  
**Java Program to override Clone method with Mutable Field**

Here is our sample Java program which will teach you how to implement clone method for a class which contains a mutable field. Remember, clone should be exactly same as original object but they must be different object i.e. if one reference change value of mutable member than cloned object should not be affected by that.

import java.util.ArrayList;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

/\*

\* Java program to show how to override clone method for deep copying.

\* This example includes a mutable filed in class to be cloned to show how you deal with

\* practical classes which contains both immutable and mutable fields.

\*

\* @author Javin

\*/

public class CloneTest {

private static final Logger logger = LoggerFactory.getLogger(Cloneclass);

public static void main(String args[]) {

Programmer javaguru = new Programmer("John", 31);

javaguru.addCertificates("OCPJP");

javaguru.addCertificates("OCMJD");

javaguru.addCertificates("PMP");

javaguru.addCertificates("CISM");

logger.debug("Real Java Guru : {}", javaguru);

Programmer clone = javaguru.clone();

logger.debug("Clone of Java Guru : {}", clone);

//let's add another certification to java guru

javaguru.addCertificates("Oracle DBA");

logger.debug("Real Java Guru : {}", javaguru);

logger.debug("Clone of Java Guru : {}", clone);

}

}

class Programmer implements Cloneable{

private static final Logger logger = LoggerFactory.getLogger(Programmer.class);

private String name;

private int age;

private List certifications ;

public Programmer(String name, int age) {

this.name = name;

this.age = age;

this.certifications = new ArrayList();

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public void addCertificates(String certs){

certifications.add(certs);

}

@Override

public String toString() {

return String.format("%s, %d, Certifications: %s", name, age, certifications.toString());

}

@Override

protected Programmer clone() {

Programmer clone = null;

try{

clone = (Programmer) super.clone();

clone.certifications = new ArrayList(certifications); //deep copying

}catch(CloneNotSupportedException cns){

logger.error("Error while cloning programmer", cns);

}

return clone;

}

}

Output of Shallow copying :

[main] DEBUG CloneTest - Real Java Guru : John, 31, Certifications: [OCPJP, OCMJD, PMP, CISM]

[main] DEBUG CloneTest - Clone of Java Guru : John, 31, Certifications: [OCPJP, OCMJD, PMP, CISM]

[main] DEBUG CloneTest - Real Java Guru : John, 31, Certifications: [OCPJP, OCMJD, PMP, CISM, Oracle DBA]

[main] DEBUG CloneTest - Clone of Java Guru : John, 31, Certifications: [OCPJP, OCMJD, PMP, CISM, Oracle DBA]

After deep copying collection:

[main] DEBUG CloneTest - Real Java Guru : John, 31, Certifications: [OCPJP, OCMJD, PMP, CISM]

[main] DEBUG CloneTest - Clone of Java Guru : John, 31, Certifications: [OCPJP, OCMJD, PMP, CISM]

[main] DEBUG CloneTest - Real Java Guru : John, 31, Certifications: [OCPJP, OCMJD, PMP, CISM, Oracle DBA]

[main] DEBUG CloneTest - Clone of Java Guru : John, 31, Certifications: [OCPJP, OCMJD, PMP, CISM]

Remember, this code will work because [String is immutable](http://java67.blogspot.sg/2014/01/why-string-class-has-made-immutable-or-final-java.html), otherwise you can not reply on copy Constructor provided by Collection classes, they only provide shallow copy and not deep copy. If you need to deep copy a collection, you need to iterate over it and clone each object separately.

**Best Practices to follow while overriding clone method in Java**

Now you know how to override clone method, it's time to sharper your knowledge and learn some best practices related to clone() method in Java. We will also look, what things to avoid, because best practices will not yield any result if you don't stop following bad practices.  
**1) Return Sub class from clone() method, instead of returning java.lang.Object**

One shortcoming of clone() method is that it return Object, which means user of clone() method must do type casting to get correct type of object. From Java 1.5 onwards an overriding method can return subclass of return type declared in original method, which means you can return sub class from clone method. It is known as [co-variant method overriding](http://javarevisited.blogspot.sg/2014/03/covariant-method-overriding-of-java-5.html). This will prevent lot of type casting at client side. Unfortunately clone() method of java.util.Date is not updated to take advantage of this change made in Java 5. Which means you need to cast cloned object into Date, before using it.  
**2) Use deep copy, if your class contains any mutable field.**

Remember to perform deep cloning before returning deep copy of object, if it contains any [mutable field](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html). Java documentation of clone method says that, you may need to modify certain fields before returning them, to make cloned object completely independent from original object. For example in java.util.Date method's clone method, we are explicitly cloning cdate field, as shown below.

d = (Date)super.clone();

if (cdate != null) {

d.cdate = (BaseCalendar.Date) cdate.clone();

}

This step conflict with use of final field, had cdate would be final, you can not perform this step, because final fields can not be reassigned once initialized.  
  
  
**3) Don't throw CloneNotSupportedException.**

One of the most annoying thing, while using clone() method to copy object is [handling of checked exception](http://java67.blogspot.sg/2012/12/difference-between-runtimeexception-and-checked-exception.html), which is often necessary, because most of the time, client only call clone() method on object, they know supports cloning of objects. By catching CloneNotSupportedExcpetion internally in overridden clone() method, you do a lot of favor to your clients. You can take a look at java.util.Date's clone() method for this example, which doesn't throw this exception.

/\*

\* Return a copy of this object.

\*/

public Object clone() {

Date d = null;

try {

d = (Date)super.clone();

if (cdate != null) {

d.cdate = (BaseCalendar.Date) cdate.clone();

}

} catch (CloneNotSupportedException e) {} // Won't happen

return d;

}**Shortcomings of Cloneable and clone method in Java**

[](http://2.bp.blogspot.com/-5yyfWKRz6hY/VL5m2TbdmSI/AAAAAAAACZc/5THYBYr3K7g/s1600/Clone+method+with+Mutable+field+in+Java.jpg)

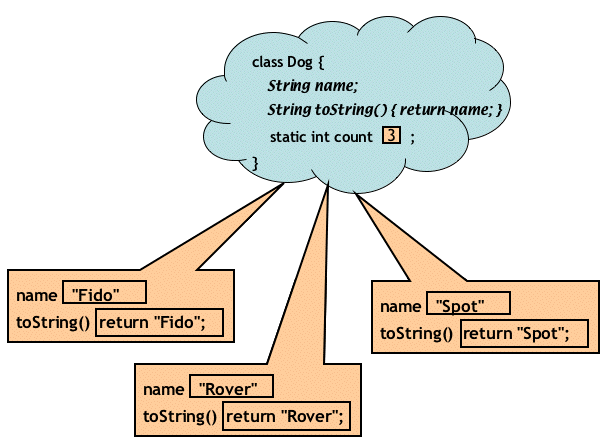
Cloneable and clone() method has failed to meet expectation of creating copy of object, it's one of the most criticized design decision made by Java API designers, along with checked exception and possibly introducing NullPointerException. Apart from not providing deep copy of Object, clone()method has several other problems and because of that many Java programmers doesn't even override clone() method as often they [override equals(), hashCode() and compareTo()](http://java67.blogspot.sg/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html) methods.  
  
  
1) Well behaved implementation of clone() method is based on convention suggested in Java documentation, rather than enforced by design. Which means it's fragile, as it makes it easy for Java programmer to forget those convention.  
  
2) Your ability to correctly override clone() method strongly depends upon, how super class overrides it. If your super class is part of any library, which is not in your control, it decisively limit your ability to create a well behaved clone() method in Java.  
  
3) Process of deep cloning in Java, conflicts with proper use of final fields, because if your Mutable object is a [final field](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html), you can not reassign copied values inside clone method.  
  
4) Creating copies using clone() method is not natural, because it doesn't call constructor, which means it doesn't leverage invariant enforced by constructor of object, and demand extra care while copying, similar to deserializing an object.  
  
5) Another annoying part of clone() method is that it throws unnecessary checked exception in form of CloneNotSupportedExcpetion, which reduces readability of code.  
  
6) Prior to Java 1.5, every caller of clone() method must need to type cast cloned object into required type, though this can be avoided by returning correct type from overridden clone() method from java 5 onwards.  
  
  
**When to use Clone method in Java**

Given all these problem associated with clone() method, it's risky to use it for creating copy, until you are absolutely sure that corresponding class provides well behaved implementation of clone() method. I personally prefer to use clone() method for creating copies of objects like java.util.Date, which provides correctly overridden clone() method but also doesn't throwCloneNotSupportedException. Similarly, if your class only contains primitives and Immutable object you can rely on shallow copy created by Object's clone() method. Another place, where you want to use clone() method in Java is for cloning arrays. Apart from that, I would suggest to prefer Copy Constructor and [Static Factory methods](http://javarevisited.blogspot.ca/2013/07/when-to-make-method-static-in-java.html) for creating objects from another object. They don't come with all problems, presented by clone() method and does the job well by creating exact copies.

**Why Override equals, hashcode and toString method in Java**Couple of questions, which are often asked to me was why do we need to override equals() and hashcode() method, Why should I implement toString(), What will happen if I don't override them or in a different way, I have never overridden equals and hashcode and not faced any problem, why should I override them now. You guessed it correct, almost all of these questions comes from beginners, who have either taken some Java programming classes, or started learning Java by their own. Though concept of equality is something you cannot ignore, correct understanding of [equals()](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html), [hashcode()](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html), [toString()](http://javarevisited.blogspot.sg/2012/09/override-tostring-method-java-tips-example-code.html) and some other method from java.lang.Object class, often goes unnoticed, at least until they self-realize it or Interviewer force them to explore that part. I have already written couple of articles on equals and hashcode, e.g. [equals vs ==](http://java67.blogspot.sg/2012/11/difference-between-operator-and-equals-method-in.html),  hashcode tips, [equals and hashcode interview questions](http://javarevisited.blogspot.sg/2013/08/10-equals-and-hashcode-interview.html) and one of my personal favorite, 5 tips to override equals in Java, which already touches this subject but I thought to explain it clearly here for once and all.  
**Why you should override equals or hashcode**

From the face, you can guess that equals() is used to check if two objects are equal or not. Now this equality can be defined in two ways, identity equality and logical equality, as I explained in [equals vs == post](http://javarevisited.blogspot.sg/2012/12/difference-between-equals-method-and-equality-operator-java.html), it's the logical equality, which is taken care by equals method. If you are doing Java programming than you probably know that every class in Java implicitly inherit fromjava.lang.Object, and from there every object inherit equals() and hashcode(). There default implementation is in line with == operator, i.e. equals() provide identity equality and return true if reference variable pointing to same object. Now, if you don't need logical equality, then you don't need to override equals, but the problem is you will need it. All your domain object e.g. Order, Trade, Message can be compared to each other and you need logical comparison. One of the popular example is java.lang.String class, which needs logical comparison i.e. character based comparison. If two String object contains same characters in same order they are considered equals, which is what you need in many programming task. Similarly, all domain object has equality defined, but true need of equals and hashcode arise, when you use them as key in hash based collection e.g. [Hashtable or HashMap](http://javarevisited.blogspot.sg/2012/12/how-to-sort-hashmap-java-by-key-and-value.html). These collection classes relies on rules of  Java programming around equals and hashcode to work according to their specification, popularly known as equals-hashcode contract. According to which, *you must override hashcode*, *if you are overriding equals*and vice-versa. Problem is that this is not enforced by compiler, and if you make such mistake, your program will not work properly.  
  
For example, any object which doesn't follows equals and hashcode contract, if used as key in HashMap, you may not be able to retrieve object again, see [how HashMap works internally in Java](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) for more details. In short, you need to override equals and hashcode, if you are writing a domain object, or you want to store them in hash based collection. Once you understand why you should override equals and hashcode, and when you should do that, it's easy to actually do that. See my post 5 tips to override equals and hashcode in Java for more details.  
**Why you need to override toString method**

You should override toString() method for all domain object, because whenever you print them using logger or [System.out.println() statements](http://javarevisited.blogspot.sg/2012/10/eclipse-shortcut-to-systemoutprintln-in-java.html), there toString() method is called. Since default implementation of toString() is not very helpful, and only print classname@hashcode e.g. com.test.User@1033203. If you print some useful information, e.g. Arun, 1022320,  it will only help you during debugging and troubleshooting. Now there are multiple ways to override toString() in Java, see that link for some easy and productive way. For  example, if you print array in Java you will not see any meaningful value, because it doesn't override toString() method, but you can still [print arrays by using Arrays.toString() method](http://javarevisited.blogspot.sg/2012/12/3-example-to-print-array-values-in-java.html). This will now show you elements stored in array, instead of just type of array and it's hashcode.

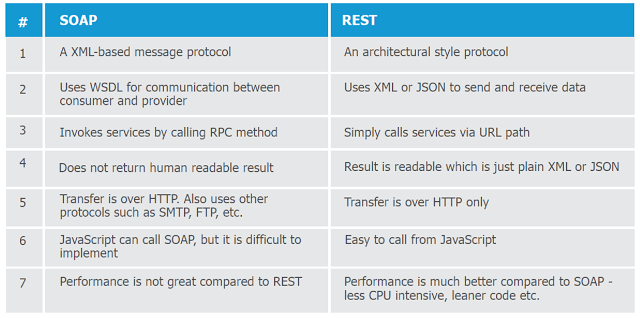
[](http://3.bp.blogspot.com/-1uQ70MeLtn4/VLPGu_hXpbI/AAAAAAAACWI/NzvZr7W4_TM/s1600/toString+in+Java.gif)

I hope this help you to understand **significance of equals, hashcode and toString method in Java**. In fact all methods of java.lang.Object are worth reading. These are fundamental concept and solid knowledge of this will only going to help you, both during Interview and on Job. Time spent on understanding java.lang, java.util and java.io are best investment in learning Java. Always remember to override hashcode() if you are overriding equals() method and vice-versa. Failing to do so is not a compile time error but can create really subtle bugs which can take hours to debug and solve, for example your HashMap reduced to linked list due to frequent collision, you not able to retrieve object put on HashMap etc.  
  
**Difference between SOAP and RESTful Web Service in Java**  
Though both **SOAP and RESTful web services** allows a client to query server for some information, but the way they are implemented and used is quite different. Main *difference between SOAP and REST* is that former provides an standard of communication between client, server and other parties and has restricted set of rules and format, while REST leverages ubiquity of HTTP protocol, in both client and servers, to allow them to communicate with each other regardless of their implementation. In short, getting data from a RESTful web service requires less headache then getting data from a SOAP web service. Since everybody is familiar with HTTP requests like [GET or POST](http://java67.blogspot.sg/2014/08/difference-between-post-and-get-request.html), its easy to understand and correlated how RESTful webservice are working and which URL of REST web service provides what kind of information. In SOAP, you need to understand lengthy WSDL document to find out right methods and right way to call them.  
  
For example, suppose you want to retrieve today's weather for a particular city from a server which is providing weather information, your RESTful URL will look something like http://weatherdata.org/data/weather/uk/london, which is very similar to HTTP request like http://weatherdata.org/data/weather?q=uk,London. On the other hand, in order to get the same data using SOAP, you need to create an XML message with header and body and send it http://www.webservicex.net/globalweather.asmx?op=GetWeather. In short, **RESTfull web services** are much more simpler, flexible and expressive than SOAP web services in Java. Apart from the obvious, let's find out some more *differences between SOAP vs RESTFul web services*.  
**Difference between REST and SOAP in Java**

Here are some fundamental differences between REST, RESTful and SOAP Web Services, which will help you not only to understand these two key technologies better but also to answer some [tricky Java web services questions](http://java67.blogspot.sg/2012/09/top-10-java-web-service-interview-question-answer-soap-rest.html) based upon these two technologies :  
  
**Short Form**  
REST stands for REpresntational State Transfer (REST) while SOAP Stands for Simple Object Access Protocol (SOAP).  
  
**Architecture style vs Protocol**  
REST is an architectural style, on which RESTFul web services are built, while SOAP is a standard devised to streamline communication between client and server in terms of format, structure and method.  
  
**Use of HTTP Protocol**  
REST takes full advantage of HTTP protocol, including methods e.g. [GET, POST, PUT, and DELETE](http://javarevisited.blogspot.sg/2012/03/get-post-method-in-http-and-https.html) to represent action e.g. from an application which provides data related to books, GET request can be used to retrieve books, POST can be used to upload data of a new book, and DELETE can be used to remove a book from library. On the other hand SOAP uses XML messages to communicate with server.  
  
**Supported Format**  
RESTful web service can return response in various format e.g. JSON, XML and HTML, while by using SOAP web service you tie your response with XML because actual response is bundled inside a SOAP message which is always in XML format.  
  
**Speed**  
Processing a RESTful web service request is much faster than processing a SOAP message because you need to less parsing. Because of this reason RESTful web services are ***faster***than SOAP web service.  
  
**Bandwidth**  
SOAP messages consumes more bandwidth than RESTFul messages for same type of operation because [XML](http://javarevisited.blogspot.sg/2015/07/how-to-read-xml-file-as-string-in-java-example.html) is more verbose than [JSON](http://javarevisited.blogspot.sg/2015/03/parsing-large-json-files-using-jackson.html), standard way to send RESTFul messages and SOAP has additional header for every message, while RESTFul services utilizes HTTP header.  
  
**Transport Independence**  
Since SOAP messages are wrapped inside a SOAP envelop it can be sent over to any transport mechanism e.g. TCP, FTP, SMTP or any other protocol. On the other hand *RESTful web services* are heavily dependent upon HTTP protocol. They used HTTP commands their operation and depends upon on HTTP for transmitting content to server. Though in real world, SOAP is mostly over HTTP so this advantage of transport independence is not really utilized.  
  
  
**Resource Identification**  
RESTful web services utilizes URL to identify the desired resources to be accessed, while SOAP uses XML messages to identify the desired web procedure or resource to be invoked.  
  
  
**Security**  
Security in RESTful web service can be implemented using standard and traditional solutions for authorized access to certain web resources. While to implement security in SOAP based web services you need additional infrastructure in web to enable message or transport level security concerns.   
  
**Caching**  
RESTful web service take full advantage of web caching mechanism because they are basically [URL based](http://java67.blogspot.sg/2013/01/difference-between-url-uri-and-urn.html). On the other hand, SOAP web services totally ignore web caching mechanism.  
  
**Approach**  
In REST based web-services every entity is centered around resources, while in case of SOAP web service, every entity is centered around interfaces and messages.  
  
**An Example**  
In first paragraph we have seen one example of requesting same web service using both SOAP and RESTFul style, you can see that REST web service is easy to understand, can be cached and required little effort to understand as compared to SOAP.

**Summary - RESTful vs SOAP Web Service**

Here is a nice summary of key differences between REST and SOAP style web services :

[](http://3.bp.blogspot.com/-zg3xuzcWTXg/Vaj0gLvGabI/AAAAAAAADZU/fhE-v_AXJFA/s1600/SOAP+vs+REST+in+Java.png)

**Some Technical details about SOAP Web Services**

1) SOAP stands for Simple Object Access Protocol but nothing is so simple about it :)  
  
2) It's only possible to send [XML messages](http://java67.blogspot.sg/2012/10/jdom-example-reading-and-parsing-xml-java.html) to server using SOAP web service because your request is embedded inside SOAP envelope which is in XML format, on the other hand RESTful web services allows you to send request in various formats e.g. JSON and flexible enough to send [JSON responses](http://javarevisited.blogspot.sg/2013/02/how-to-convert-json-string-to-java-object-jackson-example-tutorial.html).  
**Why REST is better than SOAP?**

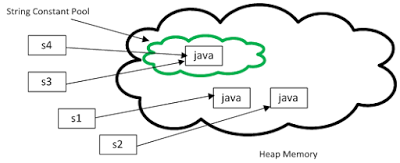
Now that you know some differences between REST and SOAP web services, let's summarize our reasons of *why REST is better choice* for modern day web service requirement :  
1. REST can be consumed by any client  e.g. Java, C++, Python client and even a web browser with Ajax and JavaScript.  
2. REST is lightweight as compared to SOAP, it doesn't require CPU consuming XML parsing and it also consumes less bandwidth because unlike SOAP, REST doesn't require a SOAP header for every message.  
3. SOAP is an old technologies, all modern technical giant are using REST e.g. Google, Twitter, and Flickr.  
4. REST is easy to learn, its just nouns and verbs. If you already know HTTP methods then its even easier.  
5. Java has excellent support for RESTFul web services, well it also has good support for SOAP web services but you have lots of choices here e.g. Jersey, RESTLet etc.  
  
That's all about **difference between REST and SOAP Web Service in Java**. Its's one of the most frequently asked questions on Java web service topic. Since REST is the technology which is right now dominating web service space, its also important to know the pros and cons REST style of web service provides over good old secure SOAP web services.  
  
**Difference between Direct, Non Direct and Mapped ByteBuffer in Java**  
ByteBuffer is one of the important class of Java NIO API. It was introduced in java.nio package on JDK 1.4, it not only allows you to operate on on heap byte arrays but also with direct memory, which resides outside the JVM. There are mainly three types f ByteBuffer, Direct, Non Direct and mapped byte buffers. You can create both direct and non direct buffers usingjava.nio.ByteBuffer class, while MappedByteBuffer is a subclass of ByteBuffer, which is created by FileChannel.map() method, to operate on [memory mapped file](http://javarevisited.blogspot.sg/2012/01/memorymapped-file-and-io-in-java.html). Main difference between direct and non direct byte buffers are there memory location, non-direct byte buffers are just a wrapper around byte array and they reside in Java Heap memory, while direct byte buffer are outside of JVM and memory is not allocated from heap. You can remember this fact by there name, *Direct* indicates working with memory directly. Due to this reason, direct byte buffers are also not affected by Garbage Collection. *MappedByteBuffer* is also a type of direct byte buffer, which represent memory mapped region of a file. In this Java NIO tutorial, you will see couple of more differences between direct, non direct and mapped byte buffers, which will help you to understand the concept and there usage better. If you love books like me and wants to learn advanced concept e.g. high performance and low latency application development, performance tuning and JVM internals, I suggest to take a look at [Definitive guide of Java Performance](http://www.amazon.com/Java-Performance-The-Definitive-Guide/dp/1449358454?tag=javamysqlanta-20), one of the must read book for Java programmers.  
**Direct vs Non direct vs MappedByteBuffer in Java**

As I said ByteBuffer is one of the very important class in high performance application. It is widely used in high frequency trading application, which strives for very very low latency, mostly in sub micro second level. When I first mentioned about memory mapped file in Java, I have outlined some benefits of using those files, and ByteBuffer class is key to operate them. Most of the differences between direct and non direct ByteBuffer derived from the fact that one is [inside heap memory](http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html), while other is outside heap.  
  
1) First difference between non-direct and direct byte buffer comes from the fact, how you create them. You can create non-direct byte buffer either by allocating space for buffer's content or by wrapping an existing byte array into buffer. While a Direct byte buffer may be created by calling factory method allocateDirect() or by mapping a region of a file directly into memory , known  as MappedByteBuffer.  
  
2) In case of Direct byte buffer, JVM performs **native IO operation** directly into buffer, without copying them into any intermediate buffer, this makes it very attractive for performing high speed IO operation on them, but this facility comes with care. If a memory mapped file is shared between multiple process than you need to ensure that it won't get corrupted i.e. some regions of memory mapped file not becoming unavailable.  
  
3) One more difference between direct and non-direct byte buffers are that former's memory footprint may not be obvious because they are allocated outside of Java heap, while non-direct buffers [consumes heap space](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) and are subject to garbage collection.  
  
4) You can check whether a byte buffer is direct or non-direct by calling isDirect() method from java.nio.ByteBuffer class. It returns true, if byte buffer is direct.

[](http://1.bp.blogspot.com/-Qn6w5VIdKXM/VcIXD5LhTLI/AAAAAAAADjU/bT-9Rc2PtIc/s1600/Difference+between+Direct+and+Mapped+ByteBuffer+in+Java.jpg)

These were some differences between direct, non-direct and mapped byte buffers in Java. If you are working in high volume low latency systems than  most of the  cases  you will work with either direct or mapped byte buffers. Since [ByteBuffer indexes are integer based](http://java67.blogspot.sg/2015/06/how-to-convert-bytebuffer-to-string-in-java-example.html), which effectively limits there addressable space up-to 2GB, you may want to check **BigByteBuffer** class from Java 1.7 nio package, which provides long indexes, alternatively you can also use offsets to map different regions of memory mapped file.  
  
That's all on **difference between direct, non-direct and mapped byte buffer in Java**. Just remember that, Direct buffers are allocated outside heap and they are not in control of Garbage Collection, while non-direct buffers are simply a wrapper around byte arrays, located inside heap. Memory mapped files can be accessed by using MappedByteBuffer, which is also a direct buffer. One more thing to remember is that default order of bytes in ByteBuffer is BIG\_ENDIAN, which means the bytes of a multi-byte value are ordered from most significant to least significant.  
  
**When to use intern() method of String in Java?**  
**String.intern()** method can be used to to deal with String duplication problem in Java. By carefully using the intern() method you can save a lot of memories consumed by duplicate String instances. A string is duplicate if it contains the same content as another string but occupied different memory location e.g. str1 != str2 but str1.equals(str2) is true. Since String object consumes a large amount of heap memory in average Java application, it makes sense to use intern() method to reduce duplication and take advantage of [String pool feature](http://javarevisited.blogspot.com/2013/07/java-string-tutorial-and-examples-beginners-programming.html) provided by Java. You can use intern() method to intern a String object and store them into String pool for further reuse. For example, when you create a [String literal](http://java67.blogspot.com/2014/08/difference-between-string-literal-and-new-String-object-Java.html) e.g. "abc", it's automatically stored inside String pool, but when you create a new String object e.g. new String("abc"), even though it's same String, a new object at a different memory location is created. This is a duplicate String. By calling  the intern() method on this object, you can instruct JVM to put this String in the pool and whenever someone else creates "abc", this object will be returned instead of creating a new object. This way, you can save a lot of memory in Java, depending upon how many Strings are duplicated in your program. You can also read [Java Performance by Charlie hunt](http://www.amazon.com/dp/0137142528/?tag=javamysqlanta-20) and [Java Performance, The Definitive Guide](http://www.amazon.com/Java-Performance-The-Definitive-Guide/dp/1449358454?tag=javamysqlanta-20) By Scott Oaks to learn more about troubleshooting Java Performance issue and optimizing the performance of Javaapplications.  
Some important things about String.intern() method

Here are some of the important points about the intern() method from java.lang.String class which are worth remembering:  
  
1) String.intern() method is there in String class from JDK 1.1. It returns a canonical representation of String object. When the intern method is invoked, if the String pool already contains that String object such that [equals()](http://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html) return true, it will return the String object from the pool, otherwise it will add that object into the pool of unique String.  
  
2) After calling intern() method on s1 and s2, **s1.intern() == s2.intern()**, if s1.equals(s2) because both will be pointing same String constant in pool.  
  
3) Prior to Java 6, uncontrolled usage of String.intern() method can cause [java.lang.OutOfMemory: PermGen space](http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html)because String pool was physically located on PermGen area of Java heap, which is quite small in many JVM (32M to 96M) and fixed. From Java 7 onward, the intern()  method has become even more useful because String pool is relocated to main [heap space of JVM](http://javarevisited.blogspot.com/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html). This will help to further reduce String duplication by using String.intern() method.

[](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html)

4) The intern() method is a non-static method and should be called using a String literal or String object in Java.  
  
That's all about **when to use String.intern() method in Java**. Java 8 update 20 also introduced a new feature called *String deduplication*, which can reduce memory footprint caused by duplicate String without writing a single line of code, but, unfortunately, it's only available for G1 garbage collector and you cannot use it if you are using ConcurrentMarkSweep garbage collector.