General:

**1. Multi-Tenancy**

Software Multitenancy refers to a software architecture in which a single instance of a software runs on a server and serves multiple tenants.

A tenant is a group of users who share a common access with specific privileges to the software instance. With a multitenant architecture,

a software application is designed to provide every tenant a dedicated share of the instance including its data, configuration, user management,

tenant individual functionality and non-functional properties. Multitenancy contrasts with multi-instance architectures,

where separate software instances operate on behalf of different tenants.

Commentators regard multitenancy as an important feature of cloud computing

ОДин сервис(код один) - много пользователей(может быть и под каждого бд)

**2. REST Semantics**

Все является ресурсами с уникальным идентификатором (URL)

Все операции клиента с сервером stateless, т.е. сервер не должен хранить вообще никакой информации о клиенте – никакой сессии

Все запросы можно поделить на 4 типа в соответствии с CRUD, причем каждому типу сопоставляется HTTP метод – Post, Get, Put и Delete

Вся логика крутится вокруг ресурсов, а не операций

**3. Stateless Web**

Stateless - There's no memory (state) that's maintained by the program

Stateful - The program has a memory (state)

To illustrate the concept of state I'll define a function which is stateful and one which is stateless

Stateless

//The state is derived by what is passed into the function

function int addOne(int number)

{

return number + 1;

}

Stateful

//The state is maintained by the function

private int \_number = 0; //initially zero

function int addOne()

{

\_number++;

return \_number;

}

In computing, a stateless protocol is a communications protocol that treats each request as an independent transaction that is unrelated to any previous request so that the communication consists of independent pairs of request and response. A stateless protocol does not require the server to retain session information or status about each communications partner for the duration of multiple requests. In contrast, a protocol which requires keeping of the internal state on the server is known as a stateful protocol.

Examples of stateless protocols include the Internet Protocol (IP) which is the foundation for the Internet, and the Hypertext Transfer Protocol (HTTP) which is the foundation of data communication for the World Wide Web.

**4. Sticky Sessions**

One solution to this problem is "sticky session" (aka "session affinity") where each user is assigned to a single server and his/her state data is contained on that server exclusively throughout the duration of the session.

Pros:

it's easy-- no app changes required.

better utilizes local RAM caches (e.g. look up user profile once, cache it, and can re-use it on subsequent visits from same user)

Cons:

if the server goes down, session is lost. (note that this is a con of storing session info locally on the web server-- not of sticky sessions per se). if what's in the session is really important to the user (e.g. a draft email) or to the site (e.g. a shopping cart) then losing one of your servers can be very painful.

depending on "sticky" implementation in your load balancer, may direct unequal load to some servers vs. others

bringing a new server online doesn't immediately give the new server lots of load-- if you have a dynamic load-balancing system to deal with spikes, stickiness may slow your ability to respond quickly to a spike. That said, this is somewhat of a corner case and really only applies to very large and sophisticated sites.

if you have relatively few users but a single user's traffic can swamp one server (e.g. complex pages with SSL, AJAX, dynamically-generated images, dynamic compression, etc.), then stickines may hurt end-user response time since you're not spreading a single user's load evenly across servers. If you have a lot of concurrent users, this is a non-issue since all your servers will be swamped!

But if you must use server-local session state, sticky sessions are definitely the way to go-- and even if you don't use server-local session state, stickiness has benefits when it comes to cache utilization (see above). Your load balancer should be able to look at HTTP cookies (not only IP address) to determine stickiness, since IP addresses can change during a single session (e.g. docking a laptop between a wired and wireless network).

Even better, don't use session state on the web server at all! If session state is very painful to lose (e.g. shopping carts), store it in a central database and clear out old sessions periodically. If session state is not critical (e.g. username/avatar URL), then stick it in a cookie-- just make sure you're not shoving too much data into the cookie.

Modern versions of Rails, by default, store session variables in a cookie for the reasons above. Other web frameworks may have a "store in cookie" and/or "store in DB" option.

**5. MVC, DI, IoC**

Инверсия управления (англ. Inversion of Control, IoC) — это принцип объектно-ориентированного программирования, при котором объекты программы не зависят от конкретных реализаций других объектов, но могут иметь знание об их абстракциях (интерфейсах) для последующего взаимодействия.

Внедрение зависимостей (англ. Dependency Injection) — это композиция структурных шаблонов проектирования, при которой за каждую функцию приложения отвечает один, условно независимый объект (сервис), который может иметь необходимость использовать другие объекты (зависимости), известные ему интерфейсами. Зависимости передаются (внедряются) сервису в момент его создания.

http://habrahabr.ru/post/62830/

https://en.wikipedia.org/wiki/Service\_locator\_pattern

http://habrahabr.ru/post/131993/

IOC реализация:

1.DI

2.Service\_locator

3.Factory

https://gist.github.com/codedokode/e1d31a31b37d5f635057

**6. Distributed vs. Replicated Cache**

https://docs.oracle.com/cd/E15357\_01/coh.360/e15723/cache\_intro.htm#COHDG319

**7. RDBMS, SQL, Transactions, ACID**

Требования ACID

Atomicity — Атомарность

Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены и система вернётся во «внешне исходное» состояние — со стороны будет казаться, что транзакции и не было. (Естественно, счётчики, индексы и другие внутренние структуры могут измениться, но, если СУБД запрограммирована без ошибок, это не повлияет на внешнее её поведение.)

Consistency — Согласованность

Основная статья: Согласованность данных

Транзакция достигающая своего нормального завершения (EOT – end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты. Это условие является необходимым для поддержки четвертого свойства.

Согласованность является более широким понятием. Например, в банковской системе может существовать требование равенства суммы, списываемой с одного счёта, сумме, зачисляемой на другой. Это бизнес-правило и оно не может быть гарантировано только проверками целостности, его должны соблюсти программисты при написании кода транзакций. Если какая-либо транзакция произведёт списание, но не произведёт зачисление, то система останется в некорректном состоянии и свойство согласованности будет нарушено.

Наконец, ещё одно замечание касается того, что в ходе выполнения транзакции согласованность не требуется. В нашем примере, списание и зачисление будут, скорее всего, двумя разными подоперациями и между их выполнением внутри транзакции будет видно несогласованное состояние системы. Однако не нужно забывать, что при выполнении требования изоляции, никаким другим транзакциям эта несогласованность не будет видна. А атомарность гарантирует, что транзакция либо будет полностью завершена, либо ни одна из операций транзакции не будет выполнена. Тем самым эта промежуточная несогласованность является скрытой.

Isolation — Изолированность[править | править вики-текст]

Во время выполнения транзакции параллельные транзакции не должны оказывать влияние на её результат. Изолированность — требование дорогое, поэтому в реальных БД существуют режимы, не полностью изолирующие транзакцию (уровни изолированности Repeatable Read и ниже).

Durability — Надежность[править | править вики-текст]

Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

**8. NoSQL, CAP**

<http://softwaremaniacs.org/blog/2010/01/31/brewers-cap-theorem/>

**9. ORM (Hibernate, JPA)**

ORM (англ. Object-relational mapping, рус. Объектно-реляционное отображение) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

JPA – это технология, обеспечивающая объектно-реляционное отображение простых JAVA объектов и предоставляющая API для сохранения, получения и управления такими объектами.

JPA – это спецификация (документ, утвержденный как стандарт, описывающий все аспекты технологии), часть EJB3 спецификации.

Сам JPA не умеет ни сохранять, ни управлять объектами, JPA только определяет правила игры: как что-то будет действовать. JPA также определяет интерфейсы, которые должны будут быть реализованы провайдерами. Плюс к этому JPA определяет правила о том, как должны описываться метаданные отображения и о том, как должны работать провайдеры. Дальше, каждый провайдер, реализуя JPA определяет получение, сохранение и управление объектами. У каждого провайдера реализация разная.

Реализации JPA:

Hibernate

Oracle TopLink

Apache OpenJPA

JPA состоит из трех основных пунктов:

API – интерфейсы в пакете javax.persistance. Набор интерфейсов, которые позволяют организовать взаимодействие с ORM провайдером.

JPQL – объектный язык запросов. Очень похож на SQL, но запросы выполняются к объектам.

Metadata – аннотации над объектами. Набор аннотаций, которыми мы описываем метаданные отображения. Тогда уже JPA знает какой объект в какую таблицу нужно сохранить. Метаданные можно описывать двумя способами: XML-файлом или через аннотации.

**10. Actors/Akka**

**11. Async I/O (Promises/Futures)**

<http://habrahabr.ru/post/112960/>

<http://12devs.co.uk/articles/promises-an-alternative-way-to-approach-asynchronous-javascript/>

<http://www.slideshare.net/xmlilley/mastering-async-io-in-javascript-promises-async-190913>

**12. Ruby/Python – Duck Typing, Dynamic Typing**

<http://www.voidspace.org.uk/python/articles/duck_typing.shtml>

The idea is that you don't need a type in order to invoke an existing method on an object - if a method is defined on it, you can invoke it.  
<http://rubylearning.com/satishtalim/duck_typing.html>

**13. JavaScript: jQuery – What is Selector?**

Селекторами называют строчные выражения, с помощью которых задаются условия поиска элементов DOM на странице. Вы можете находить интересующие вас элементы, используя выражения в стиле CSS 1-3, в дополнении с некоторыми другими возможностями. Селекторы позволяют находить элементы по различным признакам: значению атрибутов, содержимому элементов, родительским элементам, дочерним элементам, порядковым номерам, ну и конечно по именам классов, идентификаторов и/или тегов.

Например выражение $('div') осуществит поиск всех div-элементов на странице, $('.className') найдет все элементы с классом *className* и т. д. В данном примере селекторами являются строки 'div' и '.className', а $( ) — это функция, которая осуществляет поиск элементов по заданному селектору (а так же имеет массу [других возможностей)](http://jquery.page2page.ru/index.php5/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_jQuery). Ниже представлены все организованные в jQuery селекторы и правила их комбинирования.

При необходимости, можно [создавать собственные селекторы](http://jquery.page2page.ru/index.php5/%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5_%D1%81%D0%BE%D0%B1%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D1%85_%D1%81%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BE%D0%B2_jQuery).

**14. JavaScript: MVC (AngularJS, Backbone) – What is Binding?**

**15. CSS, LESS, NodeJS**

**CSS** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Cascading Style Sheets* — *каскадные таблицы стилей*) — [формальный язык](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) описания внешнего вида документа, написанного с использованием [языка разметки](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%80%D0%B0%D0%B7%D0%BC%D0%B5%D1%82%D0%BA%D0%B8).

Преимущественно используется как средство описания, оформления внешнего вида [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0), написанных с помощью [языков разметки](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%80%D0%B0%D0%B7%D0%BC%D0%B5%D1%82%D0%BA%D0%B8) [HTML](https://ru.wikipedia.org/wiki/HTML) и [XHTML](https://ru.wikipedia.org/wiki/XHTML), но может также применяться к любым [XML-документам](https://ru.wikipedia.org/wiki/XML), например, к [SVG](https://ru.wikipedia.org/wiki/SVG) или [XUL](https://ru.wikipedia.org/wiki/XUL).

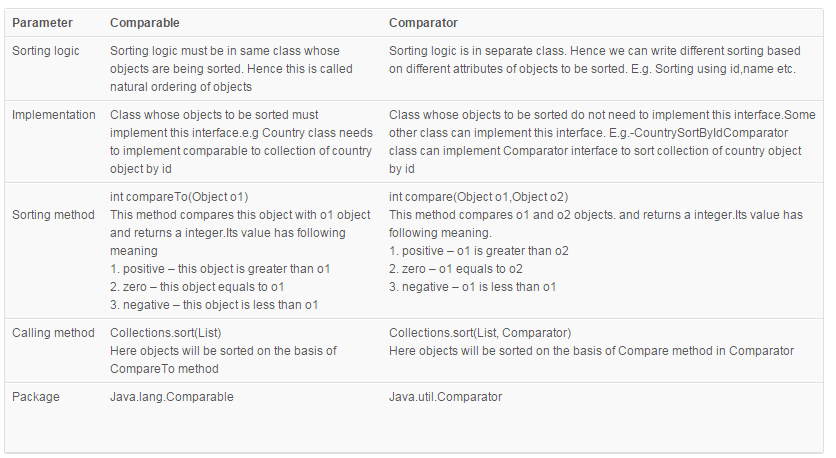
Java:

- Java’s Object: State & Behavior

- Interface, Inheritance

- By-Reference/By-Value

- TreeSet/Comparable/Comparator/NavigableSet/SortedSet



Comparable  
public class Country implements Comparable{

@Override

public int compareTo(Object arg0) {

Country country=(Country) arg0;

return (this.countryId < country.countryId ) ? -1: (this.countryId > country.countryId ) ? 1:0 ;

}}

Comparator

public class CountrySortByIdComparator implements Comparator<Country>{

@Override

public int compare(Country country1, Country country2) {

return (country1.getCountryId() < country2.getCountryId() ) ? -1: (country1.getCountryId() > country2.getCountryId() ) ? 1:0 ;

}

}

**SortedSet.** A [Set](http://docs.oracle.com/javase/6/docs/api/java/util/Set.html) that further provides a *total ordering* on its elements. The elements are ordered using their [natural ordering](http://docs.oracle.com/javase/6/docs/api/java/lang/Comparable.html), or by a [Comparator](http://docs.oracle.com/javase/6/docs/api/java/util/Comparator.html) typically provided at sorted set creation time.

**NavigableSet.** A [SortedSet](http://docs.oracle.com/javase/6/docs/api/java/util/SortedSet.html" \o "interface in java.util) extended with navigation methods reporting closest matches for given search targets. Methods lower, floor, ceiling, and higher return elements respectively less than, less than or equal, greater than or equal, and greater than a given element, returning null if there is no such element. A NavigableSet may be accessed and traversed in either ascending or descending order. The descendingSet method returns a view of the set with the senses of all relational and directional methods inverted. The performance of ascending operations and views is likely to be faster than that of descending ones. This interface additionally defines methods pollFirst and pollLast that return and remove the lowest and highest element, if one exists, else returning null. Methods subSet, headSet, and tailSet differ from the like-named SortedSet methods in accepting additional arguments describing whether lower and upper bounds are inclusive versus exclusive. Subsets of any NavigableSet must implement the NavigableSet interface.

**Treeset**. A [NavigableSet](http://docs.oracle.com/javase/7/docs/api/java/util/NavigableSet.html" \o "interface in java.util) implementation based on a [TreeMap](http://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html" \o "class in java.util). The elements are ordered using their [natural ordering](http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html), or by a [Comparator](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html) provided at set creation time, depending on which constructor is used.

This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains).

- HashMap: hashCode() & equals()

- ArrayList vs. LinkedList

- ConcurrentHashMap

- Atomics

- Concurrency: synchronized/volatile, happens-before (JMM)

- java.util.concurrent: Concurrent Collections (CopyOnWrite e.g.), Queues, Synchronizers (Semaphore, CountDownLatch, CyclicBarrier, Exchanger, Phaser), Executors (Future & Callable), Locks, Atomics

- Fork Join

- NIO (Channels/Selectors)

- SoftReference, WeakReference