**Java Core**

**OOP**

Инкапсуляция – отношения одних эл. Системы к другим

Наследование – развертывание функц

Полиморфизм – один и тот же поведение разными способами

Абстракция – получение только необходимых параметры

Interface - Abstract class - Ordinary class – Enum

Inner – логическая группа

Nested static - функции без инициализации

Nested - локальная группа без создания класса сравнения.

Anonyms — создать призрачный код без имени (если создание локального стоит дорого)

Class – abstract, final, static

Meth – abstract, final, static, native, synchronized.

Field – static, final, transient, volatile

String objects are themselves immutable.

StringBuffer is synchronized.

StringBuilder is faster

Метаанотации, которые расширяют возможности уже созданных аннотаций. @Target @Retention

(c)Throwable -> (c)Exceptions | (un)Errors -> (un)Runtime | (c)IFileNotF

(Checked || Unchecked)

Неправильно ловить исключения и ничего с ними не делать

Нужно избегать критических ситуаций в важных моментах.

Не закрывайте ресурс руками

Бросайте раньше, ловите позже

Не забываем про лог

Сначала поймайте узкое место, затем широкую

ArrayList – классический (быстрый доступ, +- добавление, удаление)

LinkedList – быстрое добавление, удаление +- получить

Queue – FIFO (linked)

PriorityQueue – FIFO

ArrayDeque — голова + хвост

Set – набор уникальных

HashSet — основная реализация

LinkedHashSet – связанные элементы набора

TreeSet – красно-черная сортировка деревьев

HashMap — классическая система бакетов

LinkedHashMap – бакеты + порядок

HashTable – thread save.

TreeMap – как set, ключи в красно-черном дереве.

**Синхронизация**

Synchronized.collections (sList, sSet, sMap) (коллекция в качестве монитора)

util. concurrent пакет — ConcurrentHashMap (backet lock) CopyOnWriteArrayList (копия коллекции для чтения с периодическим мержем)

NEW /RUNNABLE /BLOCKED /TIME\_W /TERMINATED

Вручную, Interrupt() устарел

stop() – остановка после отработки

yield() сообщает, что нужно запустить еще один поток

join() сообщает, что поток запустится, когда основной закончит

Happens before – JVM может менять иструкции. Volatile решит проблемы

PECS – некоторые потоки добавляют значения в blocking queue, некоторые получают значения.

ReentrantLock – получить разрешение потока на войти в монитор, не останавливаясь (lock9, tryLock, unlock)

ReadWriteLock – блокировать чтение или запись

IO – чтение/запись сразу. Thread save NIO – Buffer, Channel, Selector – Быстрее, больше контроля, не блокирует поток (потоконебезопасный)

Serializable – классический способ

Externalizable — read/writeExternal Если родитель — ser. Childs также.

Если на поле есть ссылка, необходимо указать ser.

Serial \ Parallel (default) \ CMS \ G1

-------------------------------------------------------

**Java 8**

**Default methods in interfaces**. Это помогает нам без особых проблем добавлять новые функции в существующие реализации.

**Functional interfaces -** ровно один абстрактный метод (для стрим апи)

**Lambda expressions** - не нужно тратить время на создание анонимных объектов

**References to methods -** для упрощения кода (Comparator, Runnable, Callable)

**Optional** - способ избежать NPE

**Stream** - упрощение кода с построением цепочек вызовов. (peak)

(FlatMap) - преобразование нескольких списков списков в один список.

**DateTimeApi** – новые объекты, помогающие работать со временем и датой.

-------------------------------------------------------

**Testing**

Functional \ Non-functional

Positive \ Negative

By access (black\white\grey)

**Manual** – низкая стоимость и высокая скорость.

**Auto** – высокие затраты на старте, безопаснее в конце

**Unit** – one logical unit (desc. FIRST rule)

**Fast/Independent/Reliable/Self valid /Timeliness** (as soon as possible)

**Integration Testing –** логически связанный модуль

**System** – глобальная проверка приложения BL

**Functional** – эмулируйте поведение пользователя

**Smoke testing** – test before app run

**Regression test** – запуск уже созданных тестовых случаев.

**Acceptance test –** проверка блока BL

**Penetration test** – различные стресс-тесты

**Fuzzing test** – случайные входные данные

**Test Pyramid** это абстракция, которая означает группировку тестов на разные уровни и показывает, сколько тестов должно быть в каждой из этих групп.

**TDD** Coding as Production vs. Coding as Thinking (когда нет четкого поведения, но есть набор входных и выходных значений)

-------------------------------------------------------

**SQL**

DBMS (Database Management System)

**Relational** – различные объекты представлены в виде таблиц c PK and FK (Azure Data Explorer, MySQL Query Analyzer)

**Non-relational** – множество объектов представлено не только в виде таблиц/json/docs/xml.

**Normalization**

**1NF –** отсутствие дубликатов и использование ПК

**2NF** - концепция удаления дублирующихся данных. Столбцы, не относящиеся к PK, не могут зависеть друг от друга.

**3NF** - все столбцы должны зависеть от PK.

**ACID**

**Atomicity –** транзакция должна сработать сразу или быть полностью отменена.

**Consistency –** данные должны соответствовать всем правилам таблицы (проверки, тип данных).

**Isolation –** параллельные транзакции не должны оказывать влияние на результат каждой транзакции.

**Durability –** будьте уверены, что после завершения транзакции результат будет сохранен.

**Read uncommitted** - наихудшая согласованность данных (самая высокая скорость), каждая транзакция видит незафиксированные изменения другой транзакции (ошибка грязного чтения)

**Read committed -** на этом уровне одновременно выполняющиеся транзакции видят только зафиксированные изменения из других транзакций (работа с разными данными).

**Repeatable read** - транзакции могут блокировать другие строки для записи или чтения (фантомное чтение).

**Serializable** - транзакции вызываются строго по цепочке. Высочайшая согласованность данных, но самая низкая скорость производительности.

**CAP**

Consistency - данные в разных репликах одинаковы.

Availability – можно получить ответ на любой запрос, но он может быть неактуален.

partition tolerance - в случае недоступности одного из разделов система продолжает работать

CA – MySQL/Postgres

AP – Dynamo/Simple DB

CP – MondoDB/Redis

**Indexes** это специальные таблицы, которые поисковая система базы данных может использовать для ускорения поиска данных.

**Single column // Unique // Composite** // **Default**

-------------------------------------------------------

**REST**

GET/HEAD/POST/PUT/DELETE/OPTIONS (list of possible server commands)

**Client-Server** - client and server

должны быть отделены друг от друга и иметь возможность развиваться индивидуально.

**Stateless -** Сервер не должен запоминать состояние. информация для работы транзакции передается в каждом запросе.

**Caching –** мы можем кешировать повторяющиеся запросы для экономии производительности.

**Unified Interface** – все запросы должны быть построены по единому шаблону. + HATEOS + selfDescriptive

**Multilevel system** - означает, что клиент не знает, является ли сервер, который отвечает, конечным сервером.

**Ease of presentation** – формат ответа не должен меняться.

**идемпотентный**, если повторный запрос, выполненный один или несколько раз подряд, не меняет состояние сервера. (POST not)

**Безопасные методы** — это те, которые не изменяют внутренние ресурсы. (OPTIONS, GET, HEAD)

**Rest best practices**

Один формат получения и отправки

возвращать коды состояния ошибок с сообщением в теле ошибки

Не использовать глаголы

используйте существительные во множественном числе

не забывайте о документации

используйте вложение ресурсов /users/+id/+orders

SOAP это протокол, используемый для реализации веб-сервисов.

REST это набор архитектурных правил.

REST может использовать SOAP, SOAP не может использовать REST.

- Только SOAP XML

- и не может быть кэширован

- работает медленнее

- не так легко протестировать, как REST

+ SOAP используется, когда невозможно использовать rest (требование состояния)

+ Поддержка устаревших систем

+ отличный уровень безопасности (как на примере PayPal на мыле)

WEB socket – подходит для прямого доступа, например игр, у них есть только корпус

Это может быть не клиент-сервер. Сервер может задать клиенту вопрос. Сохраняет состояние сеанса

- зависит от IP и порта

- только вертикальное масштабирование

**Spring**

+ много компонентов

+ АОП и простой MVC

+ реализация IoC

- высокий порог входа

- в некоторых ситуациях может быть медленнее, чем другие либы

- это рамки

IoC это архитектурное решение — когда программист делегирует работу по управлению объектами фреймворку. Состоит из (BeanFactory, Application Context)

DI (injection)- @Autowired когда спринг сам находит зависимость

DL (lookup) - getBean() когда мы руками запрашиваем зависимость

Singleton – как паттрн (управление Sprig)

Prototype – object (контроль создания)

Request – bean-компонент, ограниченный HTTP-запросом

Session – http session

G-session – global http session

Пользовательский bean:при котором нужно реализовать скоуп, и методы контроля инициализации и дестроя

Bean LC

- Конструктор объекта

- Инъекция зависимостей \*Autowiring

- Aware interfaces чтобы определить метадату (BeanNameAware, BeanFactoryAware)

- initialization степы (@PostCostruct/initMethod/afterPropertiesSet ())

- post степы (BeanPostProssesor.postProcAfterInit())

Destroy LC

- @PreDestroy

- destroyMethod

- DisposableBean.destroy()

@Autowired (fields, method or m init params, constructor) @Quilifier @Lazy @Bean

Configure

XML based configuration

+ самый старый метод, имеющий множество примеров и некоторые хитрые атрибуты

+ сосредоточено в 1 или нескольких XML-файлах

= это xml, высокий порог входа, свойства которого вам нужно знать

= вы увидите ошибки только при запуске приложения

Конфигурация на основе аннотаций:

вместо использования xml. Используйте аннотации (@Component, @Service, @Repo ComponentScan и т. д.)

+ Простота использования \ децентрализованность

Конфигурация на основе Java

Аннотировал класс как @Configuration, затем аннотировал методы создания объекта, используя аннотацию @Bean.

+ централизация \ + Собственная логика

АОП предоставляет возможность динамически добавлять логику к фактической логике с помощью простых функций.

Advice – когда вызывается новая логика

Aspect - класс аналог (хранить эдвайсы)

Joinpoint — (как аннотация) — точка, куда будет вызван совет.

Pointcut - (условие) – выбор необходимого совета

-------------------------------------------------------

**Spring Boot**

это модуль Spring, предоставляющий функцию RAD (Rapid Application Development)

Все что нужно чтобы заставить работать это:

Maven deps/@SpringBootApplication + @ComponentScan to class/@Component or @RestController

И у нас все работает

@Transactional - (2-фазный коммт) любое исключение автоматически откатывает предыдущие изменения.

-------------------------------------------------------

Logging – log4j / Cloud Logging

Tracking – Debug/ JVM review/Splunk

Monitoring – Java VisualVM/ DataDog/ NewReliq

Amazon X-Ray/ CloudWatch

**Soft Proc**

SDLC steps

1. Анализ и планирование 2. Требования 3. Проектирование и прототипирование

4. Разработка программного обеспечения 5. Тестирование

6. Развертывание 7. Обслуживание и обновления (UIT)

**Waterfall**

Полный дизайн и планирование

В случае изменений необходимо вернуться к предыдущим шагам, что увеличивает затраты и время.

**Agile and Scrum**

Вы можете вносить изменения в продукт в любое время

Неопределенность. Количество спринтов может быть неограниченным, нужно четко подобрать команду

Артефакты – бэклоги/обзор спринта/разработка/обратная связь/ретро/morning scrum

**Kanban**

Канбан — это методология, основная цель которой — визуализация процесса, а Скрам строится вокруг команды.

**Incremental and Iterative**

Планирование – реализация – проверка – оценка

раннее создание работающего программного обеспечения

проблемы с реализацией общей архитектуры системы

-------------------------------------------------------

**CI/CD**

CI (integration) - быстрая обратная связь для разработчиков.

- выполнение тестов; - развертывание dev envs – отправка отчетов

CD - Continuous Deployment это практика автоматизации всего процесса выпуска программного обеспечения.

**Patterns**

**Builder** - solves the problem of creating objects with many parameters, without creating large constructors for all cases, and without creating many subclasses.

**Decorator** - in the case when we have several implementations, but we need to have create combo implementations. Create a layer between the interface and implementations and make kombo calls.

**Facade** - simple interface to a complex subsystem. As an example, call center. This is one simple call method, and it already calls complex methods for processing.

**Adapter** - allows objects with mismatched interfaces to work together. For example, the response of one service in xml and the input of another in json. Create an additional data mapper layer and create a json based on xml

**Single-Responsibility** – один класс одна ответственность. упрощение работы с кодом, меньше конфликтов слияния, избежание создания божьих классов, которые трудно обновлять

**Open-Closed** - открыт для расширения и закрыт для модификации. мы должны добавить новое поведение, не затрагивая существующий код. Это связано с тем, что если мы изменим существующий код, у нас возникнет риск возникновения ошибки.

**Liskov Substitution** - подклассы должны заменять свои базовые классы. (т.е. если мы заменим класс на саб класс от этого не сломается приложение)

**Interface Segregation** - многие интерфейсы, специфичные для сервиса, лучше, чем один интерфейс общего назначения.

**Dependency Inversion** - классы должны зависеть от абстракции, а не от конкретных классов.

**DRY** - Don't Repeat Yourself - do not use the same block of code in the program but use a call to this block of code.

Java 9 -16

Arrays.createList -> List.of/Set.of

Streams -> takeWhile/dropWhile

Private methods in interfaces

Var – method local variable

New Switch Case view like lambdas

case SUBSCRIBER -> true;

Null Pointer теперь сообщают какой объект выбил NPE

s

**Primitives | Expand | Reflections api | cheap | expensive |**