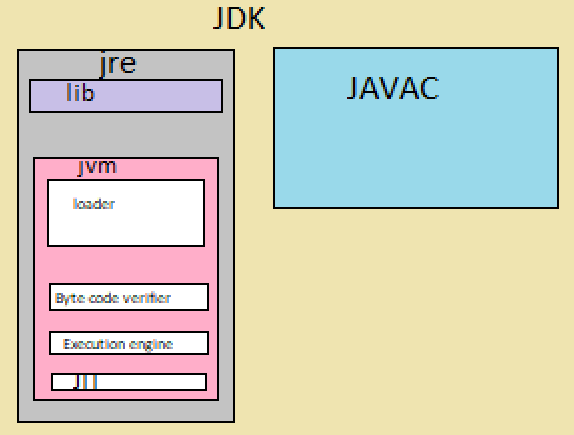
**Java Fundamentals**

1. JVM-JDK-JRE. Что это такое? Кто кого включает и как взаимодействуют.

JDK – комплект разработчика приложений на java



1. Как скомпилировать и запустить класс, используя консоль?

**javac –d classes –sourcepath src src\com\epam\varapay\Main.java**

**java com.epam.varapay.Main**

1. Что такое classpath. Если в classpath есть две одинаковые библиотеки (или разные версии одной библиотеки),объект класса из какой библиотеки создастся?

Переменная среды CLASSPATH указывает класслоадеру, какие классы необходимо загрузить и где они расположены. Пути разделяются в консоли : , а в переменной среды ; . Так же есть ключ командной строки –clsspath с помощью которого можно переопределить значение переменной среды.

Выберется та, которая попадется первой, если ее не найдет будет загружена вторая.

1. Какие области памяти использует java для размещения простых типов, объектов, методов и т.
2. Пакеты в java. Зачем применяются? Принцип именования пакетов.

Обратный интернет адресс производителя или заказчика по, затем название проекта. Только маленькие буквы. Пакеты позваляют структирирвать проект и реализовать инкапсуляцтю.

1. Модификаторы доступа.
2. Почему метод main() объявлен как public static void?

* Public. JVM вызывает main, но JVM не принадлежит к пакету в котором лежит класс с main().
* Static. Если в классе два конструктора, либо конструктор с параметрами, то было бы не понятно какой вызвать и какие параметры передать.

1. Что такое package level access. Пример использования.

ProxyConnection имеет package level access коннструктор и метод realyClose. Это запрещает создавать и закрывать прокси соединение вне пакета. Тоесть внешний код не сможет добавить лишний прокси конекшен в пул или закрыть прокси конекшен. Внешний код обязан обраиться к пулу для этих целей.

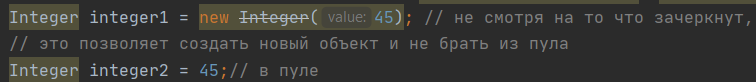
1. Может ли объект получить доступ к private-переменной класса? Если, да, то каким образом?

 public class Main {  
 public static void main(String []sss) throws IllegalAccessException, NoSuchFieldException {  
 Field f = A.class.getDeclaredField("field\_name");  
 f.setAccessible(true);  
 A a = new A();  
 f.get(a);  
 }  
}  
class A{  
 private String field\_name = "Hello";  
}

1. Классы-оболочки.

Неизменяемые.

1. Autoboxing и unboxing. Принцип действия на примерах.



* **== сравнивает ссылки**
* **> распаковывает**
* **Int == Integer распаковывает**
* **При распаковке может быть NPE**

1. int a = 1; Integer b = 2;

int a+b - результат?

А каков будет результат если Integer b = null ?

Интеджер распакуется и будет 3. Во втором случае NullPointerException.

1. Что такое var? Достоинства и недостатки.

* Снижает читабельность.
* В dimand операторе нужно указывать тип.
* Использует конкретную реализацию а не интерфейс
* Присвоение массива не требует скобок
* Нельзя объявить несколько переменных в одной строке
* Упрощает тернарный оператор, не нужно делать приведение
* Могут быть final
* Могут быть effectively final
* Нельзя инициализировать null
* Нельзя не инициализировать

**Java Strings**

1. В чем разница между созданием строки как new String() и литералом (при помощи двойных кавычек)?
2. Как реализуется класс String, какие поля там есть?

Хешкод вычесляется однажды.

Класс стринг совмещает функции контейнера данных и сервиса, для обработки этих данных.

 implements Comparable<String>, CharSequence,

private final byte[] value;

* Length
* isEmpty
* charAt
* equalsIgnoreCase
* substring
* matches
* contains
* split
* trim
* public native String intern();

1. Как работает метод substring() класса String?

substring(int beginIndex, int endIndex)

substring(int startIndex)

StringIndexOutOfBoundsException

1. Понятие Юникод. UTF-8, описание кодировки. Отличие от UTF-16

Unicode это стандарт который определяет таблицу символов и способы машинного представления этих символов в байтах. Диапазон от 0 до 10FFFF. Всего 17 групп.

Unicode Transformation Format

* UTF-8 использует от 1 до 4 байт и покрывает все символы.
* UTF-16 либо 2 байта либо 4 байта. (LE, BE) байты в обратном порядке.

1. StringBuilder

New StringBuilder(int capacity);

New StringBulder(); - по умолчанию 16

New StringBuilder(String s); - capacity = 16 + s.length

New StringBuilder(CharSequence cs) – capacity = 16 + cs.kength

Если передаем налл, то запишется строка “null”.

В отличие от HashMap, StringBuilder не имеет лоадфактора. Может быть такое что capacity == length.

Если |||добавлемоя строка + старая > capacity \* 2 + 2|||, то новая емкость будет длинна старой строки + длинна новой, причем свободных мест не останется, длинна будет равна емкости.

Если добавлемая |||добавлемоя строка + старая > capacity \* 2 + 2|||, то newCapacity = (oldCapacity << 1) + 2;

**Java Classes**

1. Базовый класс в java. Методы класса.

* public final native Class<?> getClass();
* protected native Object clone() throws CloneNotSupportedException
* protected void finalize() throws Throwable

1. OOP abstraction. Принципы ООП.

Полиморфизм – возможность объеков с одной спецификацией иметь разные реализации.

Абстракция – выделение значимых характеристик объекта. Поззволяет работать с объектами не вдаваясь в особенности их реализации.

Ковариантность возвращаемого значения.

1. Правила переопределения метода boolean equals(Object o).
   1. Рефлексивность – для любого заданного x выражение x.equals(x) будет true.
   2. Симметричность – для заданных x & y выражение x.equals(y) должно возвращать true только если y.equals(x) возвращает true.
   3. Транзитивность.
   4. Согласованность – повторные вызовы возвращают всегда одно значение.
   5. Ложь если null.
2. Зачем переопределять методы hashCode и equals одновременно?

Что бы использывать HashMap HashSet

1. Написать метод equals для класса, содержащего одно поле типа String

Если есть иерархия, еще нужно проверить super.equals();

class A{  
 String s ="dsf";  
 @Override  
 public boolean equals(Object o){  
 if(o== this){  
 return true;  
 }  
 if(o == null || !o.getClass().equals(this.getClass())){  
 return false;  
 }  
 A a = (A) o;  
 if(this.s == null && a.s == null){  
 return true;  
 }  
 return s.equals(a.s);  
 }  
}

1. Правила переопределения метода int hashCode(). Можно ли в качестве результата возвращать константу?

Использывать те же поля что и в equals

@Override **public** **int** hashCode() {

**final** **int** PRIME = 31;

**int** result = 1;

result = PRIME \* result + getId();

**return** result;}

Первый вызов медленный, поледующие быстрые System.identityHashCode(). Дефолтный хеш код дает случайное значени и созраняется в заголовке объекта.

-XX:+UnlockExperimentalVMOptions -XX:hashCode={от 0 до 5}

 0 – [Park-Miller RNG](http://en.wikipedia.org/wiki/Park-Miller_random_number_generator) (по умолчанию)  
  1 – f(адрес, глобальное\_состояние)  
  2 – константа 1  
  3 – последовательный счетчик  
  4 – адрес объекта  
  5 – Thread-local [Xorshift](http://en.wikipedia.org/wiki/Xorshift)

1. Правила переопределения метода clone().

Создает новый объект и сетает в поля ссылки на объекты из полей копируемого объекта.(Shallow) Причем конструктор не вызывается.

Shallow :

@Override

public Driver clone() throws CloneNotSupportedException {

return (Driver) super.clone();

}

Deep:

@Override

public Car clone() throws CloneNotSupportedException {

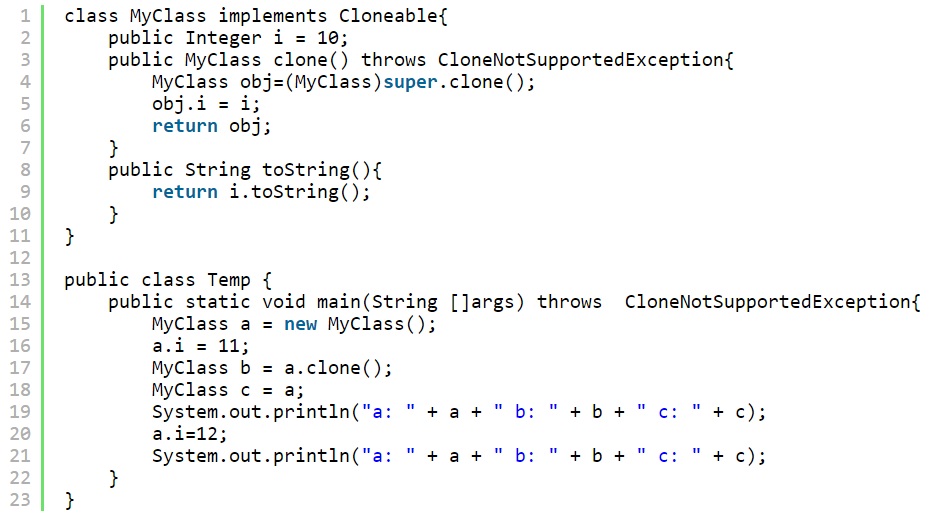
Car newCar = (Car) super.clone();

Driver driver = this.getDriver().clone();

newCar.setDriver(driver);

return newCar;

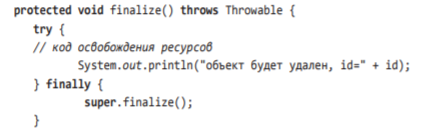
}



1. Чем отличаются finally и finalize? Для чего используется ключевое слово final?

Когда обект утратил ссылку на себя, он становится доступным для сборщика мусора. Сборщик мусора запускается нерегулярно. Мы не можем его запустить сами, мы можем лишь рекомендовать JVM запустить сборку мусора. Иногда эта рекомендация реально работает, с ней сборщик запускается сразу, а без нее не запускается.

* System.gc()
* Runtime.getRuntime().gc()



1. JavaBeans: основные требования к классам Bean-компонентов, соглашения об именах.
   1. Serializabel
   2. Private fields
   3. Public constructor
   4. Get set
   5. Equals hashcode tostring
2. Как работает Garbage Collector. Какие самые распространенные алгоритмы? Можно ли самому указать сборщику мусора, какой объект удалить из памяти.

* Удаляет ненеужные объекты
* Реструктурирует память

Сборщик может работать параллельно либо остановить приложение и запуститься.(stop the world phase)

Названия GC

* Serial Garbage Collection
* Parallel Garbage Collection
* CMS Garbage Collection
* G1 Garbage Collection (default)

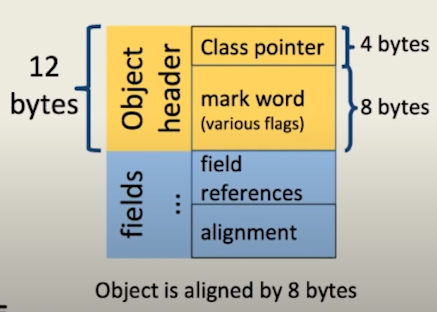
Есть два типа обнаружения ненужных объектов:

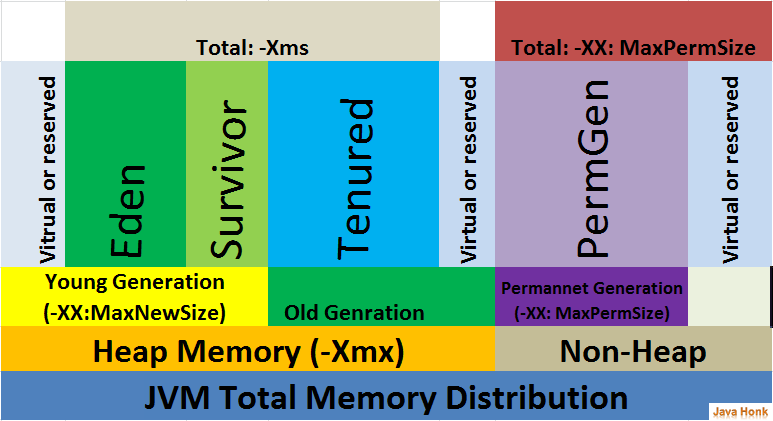
* 1. Reference counting
  2. Tracing

GC root – это то что точно не можут быть мусором.

* Параметры мейн и еог локальные переменные
* Поток который выполняет мейн
* Статические поля класса, в котором выполняется мейн

Есть несколько алгоритмов очистки памяти, mark-sweep-compact.





Объект может состоять из:

* Ссылка на класс 4
* Identity hashCode в Mark word 4
* Synchronization state
* Количество пережитых GC

1. В каких областях памяти хранятся значения и объекты, массивы?

Массив int занимает 12 + 4 + 4\*размер

Для каждого потока свой стек. Куча для всех потоков общая. Объект размещается на хипе, а ссылка на объект на стеке. Стековая память работает быстрее кучи.

1. Какие идентификаторы по умолчанию имеют поля интрефейса

PUBLIC STATIC

1. Чем отличается абстрактный класс от интерфейса.

Абстрактный класс может иметь конструкторы, их можо вызвать из подклассов.

* Интерфейс – только поведение, абстрактный класс – поведение + состояние.
* Абстрактный класс объединяет классы имеющие очень близкую связь(Bird). А один интерфейс могут реализовывать классы вобще из разных областей(Летающий- комар, самолет).
* Унаследовать можно только один абстрактный класс.
* Интерфейсы не могут иметь конструкторов.
* Поля абстрактоно класса могут иметь разные модификаторы доступа.(С java9 можно декларировать приватные методы в интерфейсе.

1. Когда применять интерфейс логичнее, а когда абстрактный класс?

Абстрактный класс когда нам необходимо иметь состояние объекта. В остальных случаях можно обойтись интерфейсом.

1. Бывают ли интерфейсы без методов. Для чего?

 Clonable, Serializable – маркерные. А так же бывает когда в интерфейсе написан тоСтринг – это значит , что в реализациях стандартная логика тоСтринг будет изменена.

1. Перегрузка и переопределение. Можно ли менять модификатор доступа метода, если да, то каким образом?

Можно менять с расширением видимости

1. Private
2. Default
3. Protected
4. public
5. Перегрузка и переопределение. Можно ли менять возвращаемый тип метода, если да, то как? Можно ли менять тип передаваемых параметров?

Можно менять тип возвращаемого значения на дочерний. Number -> Integer.

Если менять тип передавемого значения, то будет перегрузка.

1. Каким образом передаются переменные в методы, по значению или по ссылке?
2. Что такое конструктор по умолчанию?
3. Свойства конструктора. Способы его вызова.

* Из конструктора дочернего класса неявно вызывается конструктор родителя. super();
* Если мы хотим явно вызвать super(); , то это должно быть первой строкой.
* Также можно вызывать this(); Тоже должен быть первым.

1. Mutable и Immutable классы. Привести примеры. Как создать класс, который будет immutable.

После создания экземпляра, его модификация невозможна. Все поля класса являются final и в то же время Immutable, то есть одного модификатора final недостаточно. Необходимо, что бы все подобекты тоже были неизменяемы.

Неизменяемые объекты не требуют синхронизации при многопоточном программировании.

Record(14)

1. static - что такое? Что будет, если его значение изменить через объект класса? Всегда ли static поле содержит одинаковые значения для всех его объектов?
2. Внутренние классы, какие бывают и для каких целей используются. Области видимости данных при определенных ситуациях.

* Static nested class
  + Не имеют доступа к нестатическим полям и методам обрамляющего класса.
  + Помогают логически групировать сущности и улучшают инкапсуляцию.
  + Внутри статичского класса можно создавать нестатические поля.
* Member inner class
  + Не может содержать статических методов или полей.
  + Неявно связан с объектом внешненго класса
  + A.I i = a.new I();
* Local inner class
  + Объявляются в блоках инициализации, конструкторах, иетодах.
  + Не может иметь модификаторов доступа или быть статическим(иполя тоже), но может быить абстрактным или final
  + Может обращаться к статик, нестатик полям и методам внешнего класса.
  + Не может обратится к локальной переменной метода в котором объявлен класс, если эта переменная не effectively final(int h;h = 2; бедет e.f.)
* Anonymous inner class
  + Может реализовать лишь один интерфейс либо удаследовать лишь один класс.
  + Если объявляется полем класса, то может быть статик.

1. Анонимные классы. Практическое применение
2. Generics. Что это такое и для чего применяются. Во что превращается во время выполнения? Использованиеwildcards

Дженерики появились в 5 джаве и сделали использывание коллекция на много эффективнее. Ошибки связанные с некорректным использываением типов теперь обнаруживаются на этапе компиляции.

Например до джененриков, если мы работаем с листом, извлекаем объект и хотим вызвать метод на этом объекте, нам необходимо сделать приведение, а это может вызвать ClassCastException.

Type erasure

Стирание происходит в левую границу. Но не стирается информация о параметризации полей класса, и возвращаемых значений методов.

Компилятор подставляет кастование вместо дженериков.

* Ковариантность – это сохранение иеерархии наследования исходных типов в производных типах втом же порядке. (массивы, List<? extends Number> n = new ArrayList<Integer>();)
* Контрвариантность(List<? extends Number> n = new ArrayList<Integer>();)
* Инвариантность(дженерики)

PECS

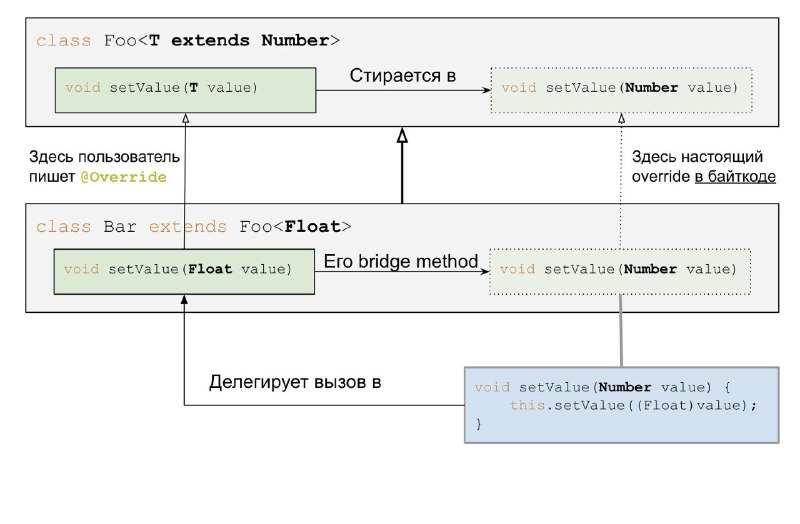
Переменная типа может быть только <T extends Comparable<T>>

Множественные ограницения разделяются &.

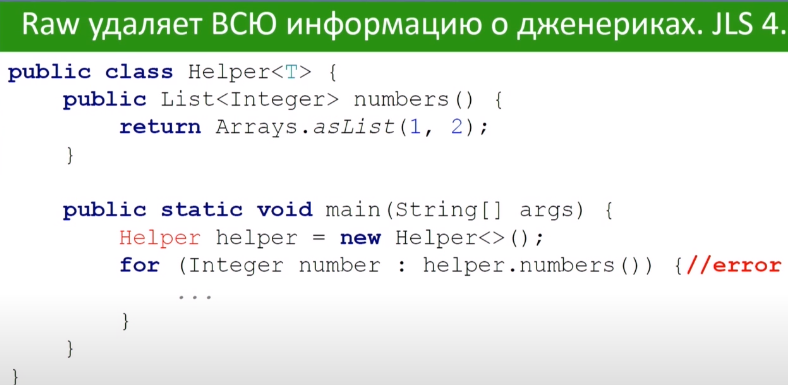
Нельзя параметризировать Throwable, enum.

HeapPlution

В java отсутсвует ковариантность переопределенных методов по параметрам. Когда дженерик параметр конкретизируется в методе наследнике, тогда получается что в наследнике тип конкретный, а в родителе стертый до верхней границы(то есть это не переопредиление) и тогда компилятор подставляет bridge method. Увидить эти методы можно через рефлексию. Такие методы делают каст и делегируют управление в метод написанный программистом.



Если мы не пишем параметризацию когда создаем объект класса ,который может быть параметризирован, тогда компилятор стирает всю информацию о дженериках.



1. Что такое enum? Область применения.

Используется для представления множества взаимоисключающих состояний. Может использываться в switch case. Implements Comparable. Мы можем определять несколько конструкторов и они все приватные. Можем определять абстрактные методы и реализовывать их.

* valueOf()
* ordinal() от нуля
* values

Неявно реализует абстрактный класс java.lang.Enum у которого есть final String name; final int ordinal;

Объекты енама являются синглтонами.

1. Отличия в применении интерфейсов Comparator и Comparable?

Comparable is located in the java.lang

compareTo(T o);

Если метод возвращает отрицательное число, то текущий объект будет расположен перед переданным. Причем метод вызывается на объекте который уже записан в коллекцию.

Comparator is located in the java.util

* Comparator.comparing();
* Comparator.comparingInt();
* Comparator.reversed()
* Comparator.naturalOrder() сошлется на Comparable.
* Comparator.thanComparing()

Compare(T o1, T o2);

Если метод возвращает отрацательное значение, тогда o1 предшествует o2.

TreeSet выбросит ClassCastException если добавить в него элемент который не Comparable.

1. Класс Optional. Как помогает бороться с null?

Устраняет избыточный код. Несериализуем. Функциональный стиль.

Если метод возвращает просто ссылку, то программист просто будет ее использывать. Если возвращает Optional, то у прогаммиста появляются инструменты по безопасному доступу к этой ссыле, причем это можно сделать в одну строку. Даже если програмист напишет get() то сонарлинт будет ему напоминать, а с обычной ссылкой такого не происходит.

* Of()
* ofNullable()
* empty()

Optional.ifPrasent() принимает лямда выражение Consumer.

* orElse
* orElseGet
* orElseThrow

Map – изменяет объект и снова обарачивает. FlatMap снимает один слой Optional и возвращает Optional и может вызываться только когда есть несколько завернутых друг в друга Optional.

1. Принципы SOLID, Yagni, Kiss, Dry.

* Подклассы не могут замещать поведение базовых классов, а могут лишь дополнять. Если s является подтипом t, тогда объекты типа t в программе могут быть заменены на объекты типа s. Если в функцию приходит родитель, то если передать наследника, все сработает. Подкласс не должен требовать от вызывающего кода больше чем базовый класс и не должен предоставлять вызывающему коду меньше чем базовый класс. Пример несоблюдения: parant.m() – ok; child.init(); cild.m(); - ok; But simply child.m() – error. Клиентский код не должен знать с каким типом он заботает(проверка instanceof – значит принцип не соблюдается).
  + - Иногда незвозможно соблюсти LSP при наследовании, квадрат не может быть наследником прямоугольника, в противном случае LSP несоблюден.
    - Шаблонный метод увеличивает шансы того что LSP будет соблюден. Программист который наследует свой класс от класса с шаблонным методом скорее всего будет вынужден реализовать предполагаемую логику. Запустить двигатель(подключить стартер, подготовить топливо..)
* Dependency inversion. Используй интерфейсы вместо конретных реализаций. Модули зависят от абстракций.

1. Если Interf – это интерфейс, а Klass – это класс, который реализует интерфейс и в нем объявлен метод(которого нет в Interf), например, method1(). Корректно ли выражение:

Interf a = new Klass(); a.method1();

То же самое, если Interf – не интерфейс, а абстрактный класс.

1. Diamand problem

Множественное наследование интерфейсов не создает проблем, так как это происходит с класами. При множественном наследовании классов если ребенок патается вызвать метод родителей, то не понятно какой именно. А с интерфейсами этой проблемы нет, мы просто переопределяем метод для всех предков сразу. Проблему создает только реализация методов, а не их описание.

Если методы(abstract or default) объявлены более чем в одном интерфейсе, то реализация по умолчанию не наследуется – ошибка компиляции.

Обход компилятора:

1. I1 содержит дефолтный метод m()

2. I2 пустой

3. С реализует оба и запускает m() если нет параметров

компилим, запускаем

1. добавляем в I2 абстрактный m()

2. компилим I2 отдельно

3. заменяем скомпилированный I2 на новый

запускаем, вызвался дефолтный I1 - C унаследовал дефолтную реализацию,

не смотря на то что методы есть в двух интерфейсах

1. заменяем абстрактный m() в I2 на дефолтный

2. компилим отдельно

3. заменяем

запускаем - исключение

запускаем с параметром(m()не вызывается) - класс отработал не смотря на то что

реализует два интерфейса с одинаковыми дефолтными методами.

49. Типы ссылок.

Объект слабой ссылки помещается в очередь перед вызовом finalize, а объект фантомно ссылки помещается в очередь по факту удаления.

* SoftReference. Объекты на которые указывают такие ссылки будут очищены первыми, если JVM будет не хватать памяти. Если памяти всегда достаточно, то объекты будут жить. Используются для реализации кеша.
* WeakReference. Объект будет очищен после первого запуска сборщика мусора, но тем не мение мы имеем доступ к этому объекту. Если мы обратимся к объекту паосле очистки, получим NPE. WeakHashMap хранит слабые ссылки в качестве ключей. Если на объект ключа(именно ссылка на объект, не equals) перестают ссылаться GC roots, то обект удаляется, а так же удаляется вся Entry вместе со значением.
* PhantomReference. Отличается от других, поскольку он не предназнаяен для доступа к объекту. Метод get всегда возвращает null. Обязательно передавать ReferenceQueue в конструктор. Когда объект, который хранит ссылка удаляется, то сам объект ссылки становится доступним в очереди.

**Java I\O**

1. Что такое сериализация, для чего нужна, когда применяется? Ключевое слово transient, для чего нужно.

Сериализация – это процесс котрый переводит объект в последовательность байтов, по которой затем можно полностью восстановить объект. Срок жизни объекта ограничен окончанием работы программы, сериализация позволяет вынести жизненный цикл объекта за рамки времени работы программы. Можно передавать файл с объектами по сети.

По умоляанию не сохраняет статик, но это можно сделать самостоятельно. Нужно перед записью объекта записать статические поля в стрим, а потом записать объект, когда десериализуем сначало прочитать статические поля и засетать их в класс, а потом прочитать сам объект.(ПОЛНОСТЬЮ ВРУЧНУЮ).

Serializable сохраняет final и читает. Externalizable может только сохранить final, но не может присвоить его во премя чтения.

Сохраняется граф объектов, только если эти объеты тоже Serializable

Serializable does not clall constructor

Что бы сериализовать объект нужно

* Создать FileOutputStream fos
* Обернуть fos в ObjectOutputStream
* Вызвать oos.writeObject

Serializable позволяет добавить методы

* Private writeObject(ObjectOutputStream stream){stream.writeObject(login);}
* Private void readObject(ObjectInputStream stream)

Externalizable заставляет риализвовать

* Void writeExternal(ObjectOutput out){out.writeObject(login);}
* Void readExternal(ObjectInput in){login = (String) in.readObject();}
* Требует вызов публичного конструктора.

private static final long serialVersionUID

Идентификатор версси сериализованого класса вычисляется по полям класса, если добавить поле, а то идентификатор изменится и при попытке десериализовать объект с другим идентификатором выпадет исключение. Что бы это предотвратить задают константу, тогда при десириализации новые поля будет с дефлотными значениями.

1. (сериализация)Возможно ли сохранить объект не в байт-код, а в xml-файл?

Класс может не реализовывать Serializable.

 XmlMapper xmlMapper = new XmlMapper();  
xmlMapper.writeValue(new File("simple\_\_bean.xml"), new SimpleBean());

1. Что такое ClassLoader? Если изменить static переменную в классе, загруженном одним ClassLoader, что будет видно в том же классе, загруженном другим. Возможно

ли синглтоны создавать несколько раз?

1. Bootstrap (standart JDK classes from rt.jar)

2. Extension

3. System(classpath)

Если один класслоадер(или его родитель) уже загрузил класс с таким именем, то при попытке загрузить другой класс(или такой же) с тем же именем, класслодер вернет первый класс. Поэтому я устанавливаю родителя null.

Каждый загрузчик ведет учет классов, которые были загружены именно им. Перед загрузкой класса загрузчик сначала опрашивает родителей не загружали ли они уже такой класс.

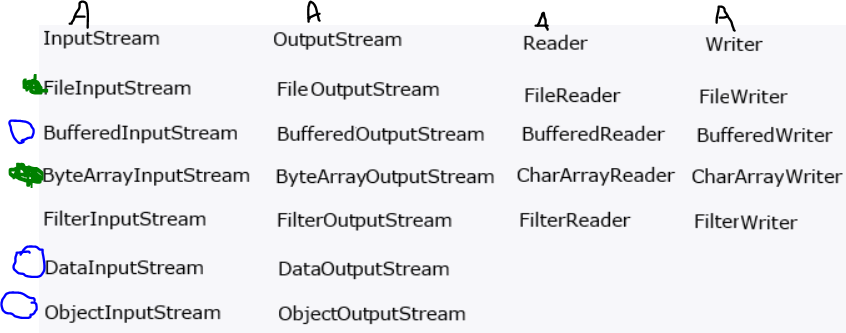
* 1. Скомпилированный класс поместить в архив, сохранив структуру пакутов.
  2. Поместить архив в папку с ресурсами.
  3. loader.loadClass(“fullClassName”);
  4. Мы не сможем сделать приведение обекта класса загруженного вторым загрузчиком к классу загруженным первым.

void loadSingleton() throws Exception {  
 URL jarUrl = Main.class.getClassLoader().getResource("by.jar");  
 ClassLoader loader = URLClassLoader.*newInstance*(new URL[]{jarUrl}, null);  
 Class<?> singletonClass = loader.loadClass("by.org.singleton.Singleton");  
 Object secondSingleton = singletonClass.getMethod("getInstance").invoke(null);  
}  
  
void reflectSingleton() throws Exception {  
 Constructor<Singleton> constructor = Singleton.class.getDeclaredConstructor();  
 constructor.setAccessible(true);  
 Singleton secondSingleton = constructor.newInstance();  
}  
  
void deserialize() throws Exception {  
 try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("data.bat"))) {  
 Singleton singleton = (Singleton) ois.readObject();  
 }  
}

1. Написать класс, который принимает имя файла и выводит строки в консоль.

 public String readLinesFromFile(String file) {  
 try (Stream<String> stream =

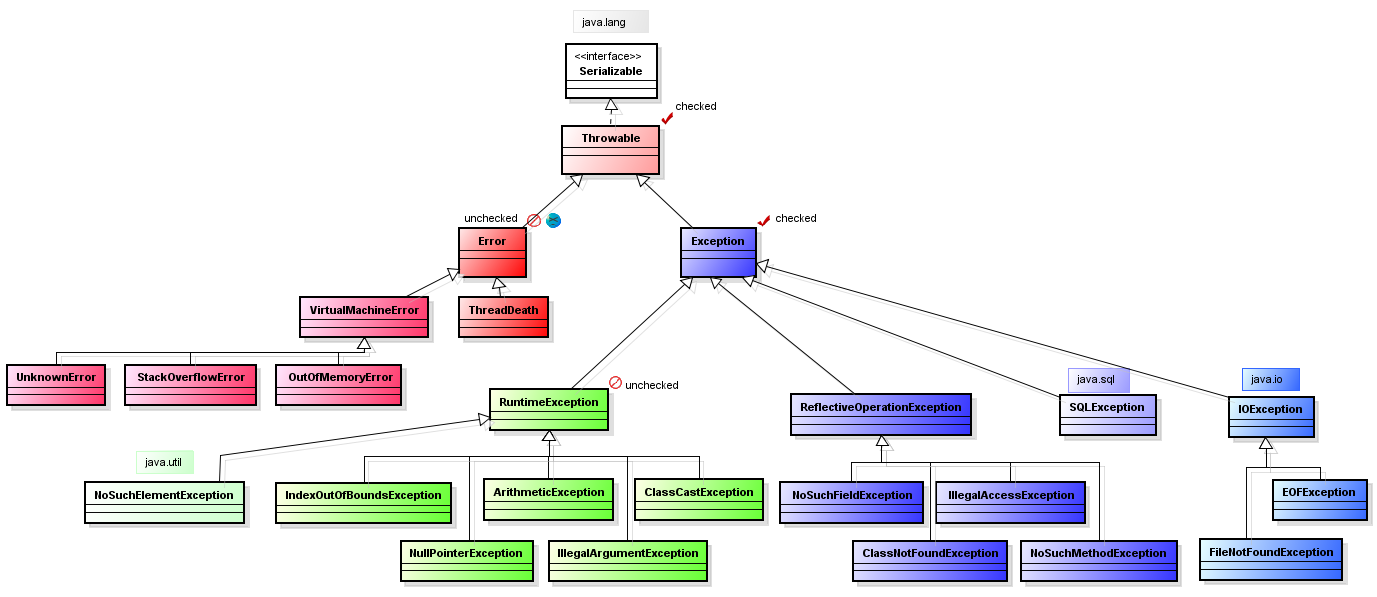
Files.*lines*(Paths.*get*(file), StandardCharsets.*UTF\_8*)) {  
 return stream.collect(Collectors.*joining*("\n"));  
 } catch (IOException e) {  
 throw new RuntimeException();  
 }  
}

52. 

flush() есть только у потоков вывода. Этот метод выводит из буфера в поток оставшиеся в буфере данные данные. По дефолту этот метод вызывается во время close(). Но если мы хотим сделать это раньше, мы вызовем этот метод.

**Java Exceptions**

1. Опишите иерархию исключений.



1. Что такое checked и unchecked Exception? Их отличия.

* Если метод родителя не кидает checked exception, то и переопределенный метод не может
* Переопределенный метод может кидать только наследника того checked исключения которое определено у метода родителя либо любое непроверяемое.

1. Опишите работу блока try-catch-finally. Может ли работать данный блок без catch
2. Что такое Error. Перехват Error. Можно ли, есть ли смысл, в каких случаях возникает и что с ним делать.
3. Для чего нужны собственные исключения? Как их создать.
4. Чем отличается OutOfMemoryError от StackOverflowError. Как их избежать?

StackOverflowError возникает когда не достаточно места на стеке что бы создать новый фрейм для вызова метода. Эта ошибка разматывает рекурсию пока не наткется на catch, память стака освобождается.

* -Xms256m начальный размер памяти(но может уменьшится потом)
* -Xmx6048m максимальный размер памяти
* -Xss129k размер стека для каждого потока.

**Java TestNG**

1. Механизм assert. Когда возникает AssertionError? В чем его смысл?

-ea

 assert (false) : "hello";

1. JUnit. TestNG. Что это такое?  Принципы написания.

**Java & XML**

1. XML – парсеры.

Сериализация –

* что то локальное
* Нами написанная программа сериализует, и нами написанная программа десериализует

маршалинг –

* передача данных.

Маршалинг – создание структуры для передачи чего то.

DOM – сохраняет дерево документа в памяти и позволяет работать с ним

SAX – событияно ориентированный API, не загружает в память

STAX -

1. Что лучше использовать в каких случаях, well-formed, valid.
2. В чем отличие dtd и xsd, какая из этих схем написана в формате xml?

XSDs are written in XML

DTD – нельзя указать типы полей.

**Многопоточность**

1. Что такое поток? Состояния потока.

**New.** После создания экземпляра потока, он находится в состоянии Новый до тех пор, пока не вызван метод start(). В этом состоянии поток не считается живым.

**Runnable.** Поток переходит в состояние Работоспособный, когда вызывается метод start(). Поток может перейти в это состояние также из состояния Работающий или из состояния Блокирован. Когда поток находится в этом состоянии, он считается живым.

**Running.** Поток переходит из состояния Работоспособный в состояние Работающий, когда Планировщик потоков выбирает его как работающий в данный момент.

**BLOCKED.** Поток может перейти в это состояние, в ожидании ресурса, такого как ввод/вывод или из-за блокировки другого объекта. В этом случае поток переходит в состояние Работоспособный, когда ресурс становится доступен.

**WAITING.** wait(); join();LockSupport.park()

**TIMED\_WAITING**. Sleep(long); wait(long); join(long); LockSupport.parkNanos(long); LockSupport.parkUntil(long);

**TERMINATED.** Поток считается мёртвым, когда его метод run() полностью выполнен. Мёртвый поток не может перейти ни в какое другое состояние, даже если для него вызван метод start()

1. Как создать поток? Какими способами можно создать поток, запустить его, приостановить, прервать (завершить,убить)?

* Унаследоваться от Tread, override run(). Create object and call start().
* Реализовать Runnable, override run(). Create Thread and convey runnable instance to the constructor.
* ExecutorService executor = Executors.newFixedThreadPool(2). Executor.submit(callable or runnable). Callble<V>{ V call() throws Exception;}

1. Как выполнить набор команд в отдельном потоке?
2. Как работают методы wait и notify/notifyAll?
3. Чем отличается работа метода wait с параметром и без параметра?
4. Как работает метод yield()? Чем отличаются методы Thread.sleep() и Thread.yield()?

Yield переводит пото в состояние готовый к запуску, тем самым освобождая процессор для других потоков ожидающих запуска. Тем не мение может произойти такое, что планировщик потоков сразу выберет тот же пото и звапустит его.

1. Чем отличаются методы Thread.sleep и wait?

Wait() освобождает монитор, а sleep() нет. Метод wait можно вызвать у объекта только когда его монитор занят, значит только в synchronized.

1. Как работает метод join()?

Метод join() вызывается на объекте Thread. Метод блокирует поток из которго он вызван до тех пор пока не выполнится поток на котором он вызван либо не истечет время если оно передано. Вызывается после start().

1. Как правильно завершить работу потока? (Иногда говорят, убить поток).

Подождать когда закончится метод run. Если есть бесконечный цикл, то изменить значение переменной.

Методы приостонавливающие работу потока выбрасывают InteraptedException если до завершения переиода сна на потоке вызван метод interapt()

1. Что такое синхронизация? Зачем она нужна? Для чего нужно ключевое слово synсhronized? Какие методы синхронизации вы знаете? Какими средствами достигается?

Синхронизация – обеспечение безопасной работы нескольких потоков с одним ресурсом. С помощью java.util.concurrent можно настроить работу программы гораздо тоньше, а значит эффективнее.

Lock lock = new ReentrantLock();  
int count = 0;  
void increment() {  
 lock.lock();  
 try { count++; }  
 finally { lock.unlock(); }  
}

А так же есть метод lock.tryLock() он не останавливает текущий поток даже если ресурс занят. Обязательно использывать с If, в противном случае если ресурс заблокирован первым потоком, а строй делает tryLock , то второй поток повредит первый поток и он упадет с исключением.

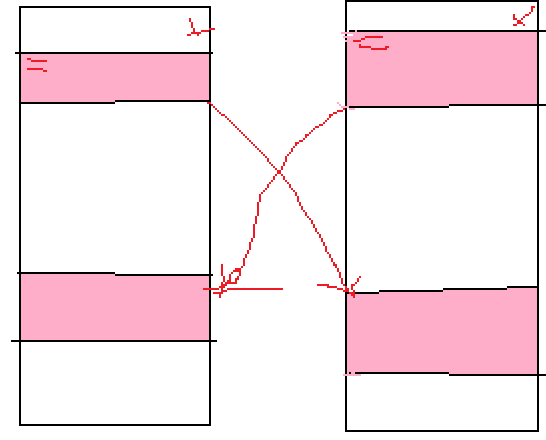
ReadWriteLock. Считывать данные безопасно(любому количеству потоков) до тех пор пока ни один из них не записывает данные.

Блокировка чтения – если ни один поток не заблокировал ReadWriteLock для записи, то несколько потоков могут получить доступ к блокировке чтения.

Блокировка записи – если ни один поток ни читает и не пишет , то один поток может получить доступ к блокировке записи.

1. Отличия работы synсhronized от Lock?
2. Что такое deadlock? Нарисовать схему, как это происходит.

Deadlock – состояние когда один поток заблокировал свой монитор, обратился к другому монитору, который уже заблокирован вторым потоком. Теперь второй поток заблокировавший свой монитор обращается к монитору первого потока.



1. Semaphore, CyclicBarrier, CountDownLatch. Чем похожи на Lock и чем от него отличаются?

java.util.concurrent

atomic

lock

CyclicBarrier

ConuntDownLatch

Semaphore

Executor

Executors

Callable

ConcurrentHashMap

CyclicBarrier имеет два конструктора

* (количество вызовов await)
* (n вызовов await, Runnable который выполнится после вызова await n раз)

В момент вызова await поток останавливается. После достижения нужного количества все потоки оживают и завершают выполнение. Если был переданнан Runnable то он выполнется тоже.

CountDownLathc в конструктор принимает количесвто вызовов метода countdown() после достижения которого будут запущениы потоки в которых был вызван await(). То есть сначала какие-то потоки вызывают await() и замерают, а потом другие потоки вызывают countdown() (сами не замирают).

executorService.shoutDown() запрещает добавлять новые потоки в executorService.

Boolean executorService.awaitTermination(long, TimeUnit) ставит текущий поток ожидание пока executorService не закончит выполнять все потоки, либо пока не истечет время.

executorService.shoutDown() закрывает пул потоков и завершает программу.

Semaphore в конструктор принимает количество потоков, которые могут одновременно заходит в критическую секцию. Methods : acquire(); release();

1. Написать deadlock, придумать примеры с использованием synchronized, AtomicInteger

 public class Main {  
 public static void main(String []sss) {  
 A a = new A();  
 B b = new B();  
 Thread t1 = new Thread(() -> a.a1(b));  
 Thread t2 = new Thread(() -> b.b1(a));  
 t1.start();  
 t2.start();  
 }  
}  
class A{  
 synchronized void a1(B b){  
 try { Thread.*sleep*(100); } catch (InterruptedException e) { }  
 b.b2();  
 }  
 synchronized void a2(){ System.*out*.println("a2"); }  
}  
class B{  
 synchronized void b1(A a){  
 try { Thread.*sleep*(100); } catch (InterruptedException e) { }  
 a.a2();  
 }  
 synchronized void b2(){ System.*out*.println("b2 never"); }  
}

1. По каким объектам синхронизируются статические и нестатические методы?
2. Для чего применяется volatile? Пакет java.util.concurrent.atomic.

По умолчаню все примитивы атомарны для чтения и записи кроме Long & Double.

* Делает переменную атомарной для записи и чтения. Но инкремент не будет атомарным.
* Запрещает процессору кешировать переменню. Переменная всегда в оперативной памяти. В результате в любой момент времени любой поток может прочитать актуальное значение переменной.
* Участвует в happens-before
* Запрещает публиковать частично инициализированные объекты

Reordering – это когда с точки зрения потока, наблюдающего за выполнением операций в другогом потоке, операции могут быть выполненны не в том порядке, в которм они в исходном коде. Это происходит когда первый поток помещает результат первой операции в кеш, а второй в память, тогда кажется, что вторая операция уже выполнилась, а первая – нет. Либо процессор просто может менять порядок выполнения операций.

В одном потоке reordering не проявляется.

Happens before

Если операция X связана отношением happens before с операцией Y, то весть код после операции Y в одном потоке видит все изменения сделанные до операции X в другом потоке. Причем второй поток будет видить в том числе изменения не volatile полей потому что в момент выполнения операции X все регистры и кеши процессора выгружаются в основную память, а в момент выполнения операции Y, процессор на котором выполняется Y инвалидирует свой кеш и зачитывает свежие данные из памяти. Связь happens before транзитивна.

Операциям ниже границы happens before позволено выполняться до границы, но операции выши не могут выполнится после границы.

Операции связанные отношением happens before:

* В одном потоке любая операция happens before любой операции следующей за ней в коде.
* Unlock happens before lock
* Выход из synchronized блока happens before вход в synchronized блок
* Запись volatile happens before чтение volatile

Static безопасен в многопоточной среде.

1. Есть массив из N-ти элементов. Создать N потоков, которые принимают по числу из массива, обрабатывают и возвращают обратно. Собрать все обработанные числа обратно в массив.

**Java. Collection**

1. Основные интерфейсы коллекций (List, Set, Map). Какие бывают коллекции? В чём особенности разных видов коллекций? Когда какие стоит применять?

Все кроме мап реализуют iterable. Каждая конкретная реализация, так же наследует абстрактный класс (AbstractCollection, AbstractList, AbstractSet, AbstractQueue, AbstractMap). Кроме ArrayDeque( наследует AbstractCollection). Эти абстрактные классы содержат общие реализации некоторых методов.

Реализации листов имеют внутренние классы, которые релизуют ListIterator.

Stack extends Vector(и не реализует никакх интерфейсов). Если мы хотим использывать Stack ,нам придется ипользыват его без интерфейса, что бы получить доступ его специфичным методам.



1. Сравнить ArrayList и LinkedList,

LinkedList занимает в два раза больше памяти. LinkedList extends AbstractSequentialList.

* Вставка в начало и конец O(1), вставка в середину O(n)
* Поиск середины O(n), сама вставка в середину O(1).

ArrayLIst default size : 10

* Вставка в конец если есть свободные места O(1), в остальных случаях O(n).
* Поиск середины O(1), сама вставка в середину O(n).

Обчно вставка в середину быстрее у ArrayList.

1. Сравнить HashMap и Hashtable.

* HashTable все методы синхронизированы, это уменьшает производительность в однопоточных приложениях.
* Не может иметь null ни как ключ, ни как значение.

1. Как устроены HashSet, TreeMap, TreeSet

TreeMap не может хранить null ключ.

1. Принцип работы и реализации HashMap. Изменения HashMap в java8.

Default capcity : 16

При достижении максимальной емкости, емкость увеличивается в два раза и массив перестраивается. Так же если в одном бакете 8 элементов, а бакетов меньше 64, то происходит resize() не смотря на то что лоадфактор не достигнут.

Key should be immutable

 capacity = number of buckets \* load factor

При добавлении, первый 16 бит хеш кода водмешиваются во вторые 16 бит с помощью XOR.

(h = key.hashCode()) ^ (h >>> 16)

На то в какой бакет попадет элеметн, влияют только нижние биты хеша, поэтому и происходит такой пересчет. Если бы этого не было, то хеш коды Integer(11112222) и Integer(41112222) попадали бы в один бакет, и распределение элеметов по бакетам было бы не равномерным, некоторые бакеты перегружены.

index = hashCode(key) & (n-1).

TREEIFY\_THRESHOLD = 8 – когда в бакете 8 элементов перестраиваетсяв дерево.

UNTREEIFY\_THRESHOLD = 6 – если количество элементов в бакете становится меньше 6, то перераспределяется в список.

MIN\_TREEIFY\_CAPACITY = 64 – при достижении этой емкости (количество элементов массива), может произойти перестроение в дерево.

В мапе может быть несколько ключей разного типа, по этому, когда формируется дерево, сравниваются классы ключей, и дерево разделяетсян агруппы по своим классам. Not Comparable ключи выстраиваются в дерево через сравнение хеш кодов, если хеш коды совпали, тогда сравнивают IdentityHashCode, если и они совпали , то элемент идет влево.

Так как при построении дерева использовыны IdentityHashCode, то поиск по not Comparable ключам осуществляется обходом усего дерева как листа.

IdentityHashMap использует вместо quals == и System.identityHashCode вместо hashCode().

1. Чем отличается ArrayList от Vector?

* Vector is synchronized
* Vecor is deprecated
* ArrayList увеличивается newCapacity = (old \*1.5) + 1, а вектор увеличивается в два раза.

1. Особенности интерфейса Set.

Новых методов не добавлено. Но написанные методы из Collection. Уникальность по equals.

1. Как добавляются объекты в HashSet ?
2. Какими способами можно отсортировать коллекцию? (привести три способа)

public static void bubble(int[] ms) {  
 for (int i = 0; i < ms.length; i++) {  
 for (int j = ms.length - 1; j > i; j--) {  
 if (ms[j - 1] > ms[j]) {  
 int c = ms[j];  
 ms[j] = ms[j - 1];  
 ms[j - 1] = c;  
 }  
 }  
 }  
}

1. Как правильно удалить элемент из коллекции при итерации в цикле?

 Iterator.remove() удаляет последний элемент, возвращенный методом next()

1. Как правильно удалить элемент из ArrayList (или другой коллекции) при поиске этого элемента в цикле?
2. Коллекции из пакета concurrent. Их особенности.

Расскажу про Блокирующую очередь.

1. Что происходит при добавлении в ArrayList нового элемента и как это реализовано.
2. Thread-safe and non-thread safe collections.
3. Метод для преобразования потоко-небезопасной коллекции в потоко-безопасную.

 Collections.*synchronizedCollection*(new ArrayList<>());

1. Написать метод в котором проверяется HashMap на наличие в нем некоторого значения, и его извлечения, если такого значения нет, надо добавать значение с пустой строкой и её вернуть. Написать код, чтобы он был как можно более эффективным (меньше затратных действий).
2. Какие потокобезопасные коллекции более «быстрые» – legacy(Vector, HashTable) или из пакета concurrent?
3. Если в коллекцию часто добавлять элементы, и удалять, какую лучше использовать? Почему? Как они устроены?
4. Как быстро получить копию коллекции. Записать код преобразования.

* Через конструктор
* addAll()
* Collections.copy(dest, src)(dest.size>= src)
* List<String> copy = list.stream().collect(Collectors.toList());
* List<T> copy = List.copyOf(list);

этого достаточно, если коллекция содежит неизменяемые объекты(String).

**Functional Programming**

1. Функциональные интерфейсы. Определение. Default & static методы. Область применения.

@FunctionalInterface вызывает ошибку компиляции если интерфейс имеет несколько методов.

Методы функциональных интерфейсов(определенных в jdk) не пробразывают исключения, поэтому приходится писать try catch в лямбде. Но мы можем создать свой фукциональный интерфейс и позволить методу пробразывать исключение.

Могут содержать дефолтные методы.

1. Лямбда-выражение. Синтаксис. Характеристики. Замыкания.

Замыкание – это функция, которая создается в теле другой функции и ссылается на effectively final локальные переменные, а так же на поля класса. (Анонимные классы, локальные классы, лямбды). В случае замыкания ссылки на переменные внешней функции действительны внутри вложенной функции до тех пор пока работает вложенная функция, даже если внешняя уже закончила работу.

* Имеют доступ к effectively final локальным переменным.
* Могут быть заменены анонимной реализацией интерфейса.
* Могут быть заменены ссылкой на метод с такими же параметрами
* Создается синглтон на котором вызывается метод.

1. Function, Supplier, Predicate, Consumer. Их применение.
2. Ссылка на метод? Что это такое? Или это все же ссылка на объект?

Ссылки на метод работают только если совпадают параметрыв лямбде и самом методе.

1. Function<String, Boolean> valueOf = Boolean::valueOf;
2. Supplier<String> toString = new Integer(5)::toString;
3. Function<String, String> function = String::toLowerCase;
4. Supplier s = Car::new;
5. Собственные функциональные интерфейсы.

**Java. Streams**

1. Чем Stream отличается от коллекции?

Коллекции позволяют работать с элементами по отдельности, а стримы позволяют только выполнять операции над всеми элементами как над одним целым.

1. Прмежуточные и терминальные операции.

Даже при работе с паралельными потоками, сохраняется порядок элементов, но исключением является forEach().

forEachOrdered() –сохраняет поряд в паралельном стриме.

Unordered() – отключает сохранение порядка, тем самым уменьшает издержки на сохранение порядка, производительность растет.

1. Методы: map() vs flatMap()

flatMap() используется когда каждый элемент стрима мы превращаем во внутренний стрм. Получается что то типа стрима, содержащего другие стримы. И flatMap() объединяет все стримы в один.

Если вызвать обычный map instead of flatMap(), то мы получим стрим, содержащий стримы.

По сути flatMap убирает внешний стрим в котором лежат потенциальные внутранние стримы и скливает внутренние стримы в один.

1. Что такое потоковая обработка данных.

Stream – это элемент, который может выполнять внутреннюю итерацию своих элементов. К потоку можно прикрепить слушателей. Слушатель вызывается когда происходит одна итерация. Слушатель вызывается один раз для каждого элемента. Слушатели потоко формируют цепочку слушателей. Слушатель может обработать элемент, а затем вернуть новый элемент для обработки следующим слушателем.

Разпараллеливание потоков позволяет задействоват несколько ядер процессора(если их несколько) и тем самым увеличить производительность. Если тоько одно ядро, то нет смысла применять parallel().

parallelStream() – превращает коллекцию в параллельный стрим. (Интерфейс Collection)

parallel() – превращает обычный стрим в паралельный

sequential() – превращает паральный в паследовательный

**JDBC**

1. Как создать Connection?
2. import java.sql.\*;
3. Регистрация драйвера. Есть два способа:
   * Class.forName(“org.postgresql.Driver”);
   * DriverManager.registerDriver(new org.postgresql.Driver());
4. DriverManager.getConnection(String url, Properties props); есть три перегрузки
5. Как правильно закрыть Connection?

St.close(); // closes also and resultSet

Cn.close();

Если используем трай с ресурсами, то закрытие происходит по очереди начиная с того который указан последним в try(). Эта очередность соблюдается даже если ресурсы создаются вне блока try(), а в него лишь передаются ссылки (java 9).

Try() разворачивается в:

try { res.doSomething(); }   
catch (Throwable var5) {  
 try { res.close(); }   
 catch (Throwable var4) { var5.addSuppressed(var4); }  
 throw var5;  
}  
res.close();

1. Какие есть типы драйверов для соединения с СУБД?

Драйвер – это сущность которая реализует интерфейсы JDBC API для взаимодействия с сервером базы данных.

* + 1. JDBC-ODBC транслятор.
    2. JDBC – native Api. Jdbc api преобразовывается в уникальный для каждой бд нативный C/C++ API.
    3. JDBC driver на основе библиоьеки java. Есть промежуточный сервер, содержащий несколько драйверов. Промежуточный сервер сам выбирает какой драйвер использывать
    4. Чистая Java. Разработан на джава и работает с бд через сокетное соединение. Наибольшая производительнность. Обычно предоставляется разработчиком БД.

1. Чем отличается Statement от PreparedStatement? Где сохраняется запрос после первого вызова PreparedStatement?Будет ли тот же самый эффект как и от PreparedStatement, если формировать запрос просто в строке и отправлять его в Statement?

Statement – полезен для статических запросов.

PreparedStatement отличается от обычного, тем что после первого вызова он сохраняется в СУБД. Драйвер сохраняет ID запроса. При выполнении такого запроса драйвер передает в СУБД только ID запроса и параметры, а СУБД находит соответсвующий запрос, подставляет параметры выполнет. Эффект может быть разный, так как в обычный Statement может попасть инъекция.

Interface PreparedStatement extends Statement

Interface CallableStatement extends PreparedStaement

1. Зачем нужен CallableStatement?

connection.prepareCall(“{call proc\_name ()}”)

1. Отличие executeUpdate от executeQuery

Все методы из интерфейса Statement.

* Boolean execute() – вернет true если resultSet может быть получен. Может заменить оба следующих gepUpdateCount(), getResultSet() .
* Int executeUpdate() – верент количество строк которое изменил запрос.
* ResultSet executeQuery() – если записи не найдены, то пустой resultSet(not null)

1. Как в объекте ResultSet вернуться в предыдущую строку? Всегда ли можно вернуться в предыдущую строку?

Result set работает по принципу итератора. Имеет указатель который указывает на текущую строку.

Во время создания стейтмента определяются характеристики резалт сета. Методы createStatement, prepareStatement, prepareCall

Имеют перегрузки принимающие  (int RSType, int RSConcurrency)

RSType:

* + ResultSet.TYPE\_FORWARD\_ONLY
  + ResultSet.TYPE\_SCROLL\_INTENSIVE
  + ResultSet.TYPE\_SCROLL\_SENSETIVE

RsConcurrency:

* + ResultSet.CONCUR\_READ\_ONLY
  + ResultSet.CONCUR\_UPDATABLE
* Методы получения данных
  + - Int getInt(int columnIndex)
    - Int getting(String columnName)
* Методы изменения данных
  + - Void updateString(int index, String newValue)
    - Void updateString(String columnName, String newValue)
    - Void updateRow() нужно вызывать после updateString(..)
    - Void deleteRow()
    - Void insertRow()

rs.moveToInsertRow();

rs.updateInt("ID", id);

rs.updateString("First\_Name", "Ishant");

rs.insertRow();

* Методы навигации
  + - Void beforeFIrst() перемещает указатель на мето перед первым рядом.
    - Void afterLast()
    - Boolean first() перемещает указатель на первый ряд
    - Boolean last() перемещает указатель н крайний ряд
    - Boolean previous() перемещает указатель на предыдущий
    - Void relative(int)
    - Void absolute(int)
    - Int getRow()

1. Последовательность действий необходимых для выполнения запроса к БД.
2. Как получить сгенерированный СУБД первичный ключ без выполнения дополнительного запроса к БД?

preparedStatement = connection.prepareStatement(query, Statement.*RETURN\_GENERATED\_KEYS*);  
setParameters(preparedStatement, parameters);  
preparedStatement.executeUpdate();  
ResultSet rs = preparedStatement.getGeneratedKeys();  
rs.next();  
int i = rs.getInt(1);

1. Когда мы используем интерфейс savepoint , мы определяем точку до которой кроизойдет откат если нам понадобится откатить изменения. Подтвердить Savepoint нельзя, все они автоматически подтверждаются в конце транзакции.

* connection.setSavepoint(String name) возвращает объект Savepoint с заданным именем.
* connection.releaseSavepoint(Savepoint savepoint) удаляет точку сохранения.
* Connection.rollback(Savepoint savepoint). Если попробывать rollback на удаленную точку сохранения, то выпадет исключение.

**Функция** должна возвращать значение, но в **Хранимой процедуре** это необязательно

**Design Patterns**

1. Зачем нужны паттерны? Привести примеры из проекта.
2. Какие паттерны вы знаете и как их применяет Java SE.

Gof23 = 5 + 7s + 11b

1. Factory Method, Builder

Пораждающие паттерны отделяют код создания продуктов от кода где они используются.

Builder

* Используем, когда нам нужно создать несколько объектов одного класса, причем нужные нам объекты после создания должны иметь разные значения полей. Под каждый специфичный объект этого класса мы создадим по Builder. Один класс – много билдеров. House : WoodenHouse, ConcreateHouse.
* Устраняет антипаттерн Telescoping constructor

Factory method

* Позволяет создавать объекты различных классов, не указывая конкретные классы.
* Используем когда, зарание не известны типы объектов, с которыми работает наш код.
* Используем если у нас есть метод, который внутри себя должен создавать иногда стул, иногда стол и тд. Мы не хотим делать два метода для стола и для стула. Тогда мы делаем что бы наш метод принимал объект фабричного метода(который создает конкретный объект либо стула, либо стола).
* Позволяет легко рассширять архитектуру с помощью наследования.
* Недостаток: может привести к созданию больших параллельных иерархий классов.

1. Singleton (class Runtime)

* Static field initialization. Greedy. Инициализация произойдет во время обращения к любому другому static (not final) полю.
* Double check, volatile. В java 1.4 и ранних версиях могла возникнуть ситуация, когда первый поток выходит из синхронайзд блока(тем самым публикует объект), а объект создался только частично, и тогда втрой поток получает ссылку на частично создвнный объект. Volatile решает эту проблему.
* Private static Holder class
* Перечисление

1. Как сделать чтобы в Singleton не работала двойная блокировка?
2. Как у Singleton  создать второй объект? И как воспрепятствовать этому.

* getDeclaredConstructor(все конструкторы), getConstructor(только публичные).
* Serializable. Но не Externalizable, так как требует публичный конструктор.
* Нет синхронизации с double check реализацией
* Разные класслоадеры

1. Prototype

Позволяет копировать объеты не вдаваясь в подробности реализации. Паттерн реализуется с помощь. Конструктора копирования либо реализации интерфейса Clonable. Имеем некоторый дефолтный объект или несколько, клонируем его, и меняем некоторые свойства клона для наших нужд.

1. Command
2. Composite

Позволяет сгрупировать множество объектов в древовидную структуру, а затем работать с ней , как будто еденичный объект, причем каждый составной объект также может являтся комбинацией составных объектов.

Получается каждый элемент дерева содержит список на ветки отходящие от этого элемента.

* Папки на компютере
* Части текста
* Xml
* Коробки вложенные друг в друга

1. Chain of responsibility (Filter, closing io stream, closing connection)

Позволяет последовательно передавть запрос по цепочке обработчиков, каждый последующий обработчик обрабатывет запрос и передает дальше. Класс обработчика имеет :

* поле со ссылкой на следующий обработчик
* сетер для следующего обработчика
* метод execute, который выполняет зпрос и вызывает следующий обработчик или не вызывает если он null

Клиентский код с помощью сетера определяет порядок следования обработчиков.

1. State (Thread.State)

Позволяет объектам менять свое поведение в зависмости от состояния.

1. Iterator (Enumeration, Iterator, ListIterator)

Позволяет последовательно обходить элементы составных объектов. hasNext(), next(), remove().

1. Proxy

Предоставляет объект – заменитель. Этот объект перехватывает вызов оригинального объекта, позволяя сделать что то до и после вызова оригинала.

* ProxyConnection перед вызовом рельного метода close() сначала вернет connection в пул.
* Логирующий прокси. Логирует, а потом только вызывает рельный объект.
* Ленивая инициализация. Предоставляет прокси объект, в которм не проинициализирован реальный. Реальный инициализируется только после вызова метода.
* Защищающий прокси
* Кеширующий ответы. Не будет происходит реального запроса, будет выдан кеш.

1. Observer (Listener-s)

Создает механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие с другим объектом.

Проблема:

Покупатель ходит в магазин каждый день и проверяет появился ли товар. Магазин не может разослать оповещения всем покупателям, по тому что не всем это интересно.

Издатель содержит список подписчиков, подписчики могут добавлять себя сами. Все подписчики имеют единый интерфейс. Когда происходит вызов определенного метода издателя(События на которое подписываются), в конце тела метода происходит вызов всех подписчиков в цикле. Метод интерфейса подписчиков может принимать определенные параметры.

1. Wrapper = Decorator(Input-Output streams)

Позволяет динамически добавлять объектам новую фунциональность. Расширяет фунционал объекта с помощью композиции, а не наследования.

Декорируемый объект реализует интерфейс A. Каждый

Каждый класс декоратора реализует тот же A и содержит поле со ссылкой типа А на другой декоратор. Декораторы сетаются в конструкторе.

1. Immutable
2. MVC

FF

1. DAO vs. Repository

Repository представляет более высокий уровень абстракции. Выполняет роль коллекии объектов в памяти.

**JEE**

1. Что входит в JEE?

* Servlet
* Jsp
* Jsf
* Ejb
* JSON-P
* JSON-B
* WebSocket
* Mail
* **Security API**

1. Что такое сервер приложений? Что такое веб-сервер, в чём его отличие от сервера приложений? Привести примеры веб-сервера и сервера приложений.

Веб-сервер раздает только статический контент по HTTP.

Сервер приложений выполняет неторую логику и генерирует динамические страницы.

1. Что такое контейнер сервлетов? Что такое сервлет?

ServletConfig для каждого свой, ServletContext для всех одщий (информация сервера)

Контейнер сервлетов - server который занимается системной поддержкой сервлетов и обеспечивает их жизненный цикл. Обеспечивает обмен данными между сервлетами и клиентами. Идентифицирует и авторизирует клиентов, организует сессию для каждого клиента. Создает потоки для новых клиентов.

Сервлет - интерфейс предназначенный для обработки запросов и отправки ответов. Может обслуживать любые запросы, но обычно используется для расширения веб серверов.

1. Методы сервлета. Жизненный цикл сервлета.
2. Что такое jsp. Жизненный цикл.

* <%= %>
* <%! %>
* <% %>
* <%@ %>
* <%-- --%>

Extends HttpJspBase

названиеФайла\_jsp

В папке work/Catalina/localhost/varapay можно посмотреть скомпилированные jsp.

out.write(“<html></html>”);

* Проверка компилировалась ли страница раньше или была изменена с момента последней компиляции.
* Парсинг
* Превращение в сервлет
* Компиляция сервлета
* **JspInit**
* **\_jspService**
* **jspDestroy**

1. Что такое сессия? Жизненный цикл.

Сессия позволяет сохранять информацию о клиенте во время сеанса. Когда клиент впервые обращается к серверу, происходит проверка есть ли в куках идентификатор сессии. Если нет, то томкат создет идентификатор сессии и прикрепляет его к объекту ответа и создает сессию. Когда ответ приходит в браузер он сохраняется в куках. При следующих обращениях к серверу в куках уже будет идентификаор сессии.

<session-config>  
 <session-timeout>180</session-timeout>  
</session-config>

1. Сookies. Как можно достать Сookies, а если Сookies удалить, можно ли достать сессию?

Протокол HTTP не имеет сессии и при каждом новом запросе создается новое соединение.

Для поддержания сессии можно использывать

* Куки req.getCookies(); resp.addCookie();
* Скрытые поля форм
* Перезаписать url

1. Что нужно написать в браузерной строке, чтобы обратиться к сервлету? Можно ли из браузерной строки напрямую вызвать метод сервлета?

<http://localhost:8080/varaPay_war_exploded/mainServlet>

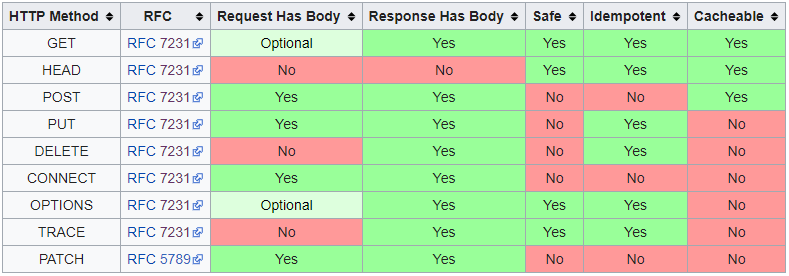
Напрямую вызвать метод сервлета нельзя, только после того как управление будет передано сервлету.

1. Чем отличается методы POST и GET. Если не указать напрямую, какой из этих методов выполнится по умолчанию?

Условный GET запрашивает передачу объекта, только если он удовлетворяет условиям указанным в заголовках. Предназначен для уменьшения ненеужной нагрузки сети.

Headers:

* If-modified-since (304 Not Modified)
* If-unmodified-since
* If-mathc
* If-non-match
* If-range



1. Как сделать redirect незаметно для пользователя?

Врядли

1. Какие scopes (области видимости) переменных существуют в JSP?

* requestScope
* sessionScope
* applicationScope
* pageScope

1. В чём различие forward и redirect?
2. Отличия getAttribute() от getParameter() в сервлете
3. Из чего состоит url?

<http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2>

1. протокол
2. доменное имя, которое будет преобразовано в IP через запрос к DNS. Запрос к DNS происходит не всегда, если браузер рание обращался к данному серверу, то адрес может быть сохранен в кеше. Можно назвать это адресом сервера веб приложений.
3. Порт(не всегда виден)
4. Мапинг сервлета
5. После ? значения из форм представленые как ключ-значени через &(Если это GET)
6. Отличие jsp:include от директивы include

<jsp:include page=”/mainServlet”/> Выполняет запрос и вставляет ответ. Работает в рантайме. Работает медленне так как нужно отправить запрос.

<%@include file=”” %> Просто вставляет текст. Используется для подключения статических файдлов. Работает на этапе трансляции. При генерации сервлета просто вставится out.write(“many times”);

<c:import url=””/> также как и jsp:include, толко блее совершенный.

При вставке статического файла, его содержимое(<c:set/>) будет влиять на текущий файл. При динамическом обращении вставляется результат выполнения, то есть сами HTML теги, но не <c:set/>

1. Применение классов HttpServletRequestWrapper и HttpServletResponseWrapper.

Позволяют рассширить классы req, resp и создать собственную реализацию этих классов.

1. Что делает RequestDispatcher.include()?

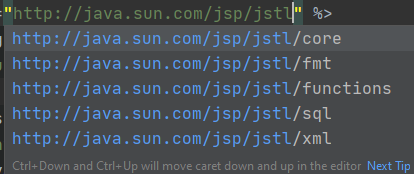
Включает ответ одного сервлета в ответ другого. Клиент получает овет сразу двух сервлетов.

1. В какой последовательности выполняются сервлет фильтры?
2. Как обрабатывается тег с телом?
3. JSTL.

Позволяет не использывать java код на jsp страницах. В JSTL ислульзуется EL для доступа к данным хранящимся в контейнерах атрибутов.

<c:set var value scope/> помещает value по ключу var в указанную scope(default page).

<jsp:useBean id scope type/> асоциирует объект, определенный в scope с заданным id на этой jsp странице.



1. Что нужно написать в строке браузера, чтобы обратиться к хосту, на котором установлен tomcat, развёрнуто приложение, в котором есть несколько сервлетов? Как обратиться к конкретному сервлету? Что такое www?Где нужно указывать порт?

<http://localhost:8080/varaPay_war_exploded/mainServlet>

1. Можно ли в web.xml определить сервлет без указания url паттерна и как к нему обратиться?
2. Что такое HTTP? Отличия HTTP 1.0 и HTTP 2.

TCP & Udp работают на транспортном уровне.

TCP использует трехстороннее рукопожатие (SYN, SYN ACK, ACK). Гарантирует, что данные будут доставлены без потерь.

UDP передает данные без предварительной проверки готовности принимающей стороны, не устанавливает сессию.(данные могут потеряться, упорядоченность данных не соблюдается.) Используется для аудио и видио звонков.

* 1xx – информационные 102(сервер получил запрос и обрабатывает его, но обработка не завершена)
* 2xx – успех.
* 3xx – требуются дополнительные действия – редирект.
* 4xx – ошибка клиента. 403(нет прав) 405(неподдерживается метод)
* 5xx – ошибка сервера.

GET /mapping HTTP/1.1

HTTP/1.1 200 OK

HTTP 2.0 появился в 2015

В HTTP1.0 если клиент хочет создать несколько параллельных соединений для повышения производительности, ему необходимо создать несколько TCP соединений.

HTTP2.0

* Использывание нескольких потоков в одном TCP соединении.
* Каждый поток иммет связанный вес и зависимость.
* Сжатие хедеров
* Отправка кешируемой информации по инициативе сервера. Отпраляются данные ,которые сейчас не нужны клиенту, но могут понадобится в будущем.

1. Cохраняет ли http протокол своё состояние.
2. Как сервер понимает, что для пользователя создана сессия и не нужно её создавать?
3. Отличие авторизации от аутентификации.

* Аутентификация – процедура проверки подлинности, доказательство что пользователь имнно тот за кого себя выдает.
* Авторизация – предоставление прав.

1. Пошагово рассказать, что происходит, когда пользователь нажимает на кнопку. (с формы поля сетаются вrequest, потом вызывается контейнер сервлетов, потом он как-то по request понимает, куда нужно идти дальше(в дескриптор развертывания, а их может быть несколько, нужно как-то понимать в какой).
2. Какие бывают WebService?

Веб сервис – набор методов предоставляемый приложением и доступный для использывания по сети. Позволяет приложению предоставлять свои услуги для других приложений. Позволят подружить приложения на разных технологиях. Если посмотреть на веб сервисы в разрезе стека тетевых протоколов, то это всего лишь еще одно надстройка над протоколом HTTP.

SOAP более применим в сложных архитектурах

1. RESTful сервис. Принципы работы.

Representational state transfer.

1. **Client-Server**. Клиенты не связанны с хранением данных, которое остается внутри каждого сервера. Аким образом мобильность кода клиента улучшается.
2. **Stateless.** Каждый запрос клинта должен содержать всю необходимую информацию и не может полагться на какое либо состояние храящееся на сервере. Данные для айтентификации передаются с каждым запросом.
3. **Cacheable**. Для данных приходящих с сервера должно быть явно указано можно ли их кешировать. При следуюзих обращениях донные берутся из кеша.
4. **Uniform interface.** Единый интерфейс между клиентами и серверами позволяет каждой части развиваться самостоятельно.
5. **Layered system.** Допускается разделить сиситему на иеерархию слоев, но с условием, что каждый слой может видеть только компоненты следющего слоя. Не может знать о других слоях кроме соседних.
6. **Code on demand.** Позволяется загрузка и выполнение кода на стороне клиента.
7. XSS атака.

Атака внедряющая JavaScrip код. Примеры:

* Злоумышленник вводит в форму или в строку раузера JavaScript. Если на форумк опубликовать комментарий с вредоносным кодом, то когда другие пользователи будут загружать эстраницу с этим коментарием, будет выполнятся вредоносный код, который может отсылать куки авторизации пользователей на сервер злоумышленника. Потом злоумышленник посылает запрос с авторизационными куками на сайт и получает доступ к аккаунту.
* Переход по ссылке на реальный сайт. Ссылка содержит URL реального сайта и один из параметров ссылки – это JS. Когда пользыватель переходит, посылается запрос, выполняется JS и кики пользователя отсылаются злоумышленнику.

**SQL**

1. Что такое нормализация.

Ситуацию в которой утверждений о состоянии какой-то части системы более одного, называют дублированием, избыточностью. Если есть дублирование информации, то возможна ситуация, когда данные будут противоречить друг другу. В этом случае БД не консистентна, и возможны ошибки.

Нормализация - устранение дублирования данных с целю достижения консистентности.

НФ(7) - шаги, которые нужно пройти на пути к консистентности. Нф 3 - достаточно

НФ1(атомарность)

НФ2(все не ключевые атрибуты зависят от всего ключа отношения)

НФ3(Все атрибуты отношения НАППРЯМУЮ зависят от ключа отношения. Зависимости через цепочку атрибутов быть не должно)

В результате нормализации уменьшается размер бд.

Денормализация - внесение избыточности с целью избежать join и ускорить запрос.

Типы моделей данных:

* OLTP. Обработка транзакций в реальном времени. Требуется минимальное время отклика. Используются INSERT, UPDATE, DELETE. Объем такой базы не большой. Потеря данных приводит к денежной и юридитеческой ответсвенности. Избыточность недопустима, так как это усложняет операции изменения данных(изменение одного объекта затрагивает большое количество строк). Имеют более сложные связи.
* OLAP (DWH). Оперирует большими массивами данных накопленных на предприятии. Добавление данных происходит редко(раз в день) но крупными блоками. Данные никогда не удаляются. Скорость выполнения запросов не критична. Предназначены для нахождения зависимостей между данными и динамического анализа. Пример : dashboard с графиками курсов валют. Пользователь работает не с отдельными транзакциями с показателями рассчитанными тысячами-миллионами других транзакций. Данные в OLAP системы могут попадать из OLTP. Потеря данных не так страшна. Применяется лишь 2 НФ.

На основе отчетов построенных на базе OLAP рождаются решения которые влияют на бизнес процессы.

1. Какие есть типы связей в базе данных. Привести пример.

create table employees(

    employee\_id serial not null primary key,

    name varchar(20)

);

create table positions(

        position\_id serial not null primary key,

        position\_name varchar(20) not null

);

create table employee\_position(

    position\_id int,

    employee\_id int,

    constraint fk\_employee

        foreign key(employee\_id)

        references employees(employee\_id),

    constraint fk\_position

        foreign key(position\_id)

        references positions(position\_id),

    primary key(position\_id, employee\_id)

)

create table DisabledPerson(

    disabled\_person\_id serial primary key,

    employee\_id int unique,

    constraint fk\_employee

        foreign key (employee\_id)

        references employees(employee\_id)

)

1. Что такое primary key (первичный ключ)?

Ограничение позволяющее однозначно идентифицировать каждую запись в таблице.

1. Что такое foreign key (внешний ключ)?

Внешний ключ устанавливается для столбцов из подчиненной таблицы и указывает на один из столбцов главной таблицы. Внешний ключ может ссылаться только на primary key главной таблицы.

1. Что такое индексы в базе данных? Для чего их используют? Чем они хороши и чем плохи?

Индексы - специальные таблицы, которые могут быть использованы базой для ускорения получения данных. Автоматически создаются для unique и PK

Ускоряют выборку с where , но не относятся к having. Замедляют insert, update

create index some\_ind on employees(name)

* HashMap
* Bit-map
* Binary tree
* B-tree

1. Какие есть типы JOIN'ов. Кратко опишите каждый из типов.

UNION, INTERSECT or EXCEPT operator must have an equal number of expressions and the same type of the expressions (varchar can have different capacity) in their target lists.

1. Inner join = cross join + where
2. Left join
3. Right join
4. Full join
5. Cross join = перечисление через запятую . Работает с where.
6. Natural join – джоинит по полям имеющим одинаковые имена. Причем если указано \* то общие поля выводятся только единожды. Если общих полей нет, то будет cross join. Работает с where
7. Union
8. Union all
9. Intersect
10. except

1. Для чего используется слово HAVING? Отличия от WHERE.

Having используется для выборки груп после группировки. Where is used to tuples. В having sum(\_) можно использывать поля, не участвующие в group by. Так же в select sum(\_) можно использывать поля, не участвющие в group by.

1. Зачем нужно View и какие поля там будут.

Нельзя редактировать, вставлять и удалять из view, который имеет group by.

Можно всталять строки в таблицу через представление даже если новая строка не удовлетворяет where в представлении, при этом такая строка сохранится в таблице, но не будет отображаться в представлении. Можно запретить добавление строк не соответсвующих where, для этого в создании представления нужно добавить WITH CHECK OPTION

1. Есть таблицы Customer и Order. Вывести всех сustomer, у которых суммарный заказ будет > 10000.

create table customers(

    customer\_id serial primary key,

    customer\_name varchar(20)

);

create table orders(

    order\_id serial primary key,

    customer\_id int references customers(customer\_id),

    order\_sum int not null);

1. Что такое агрегирующая функция, примеры.

Вычисляют одно значение над некоторым набором строк.

* Используются сами по себе
* Используются с группировкой

1. Результат запроса *Select \* from Table1, Table2*;
2. Есть 2 таблицы

create table department(

department\_id serial primary key,

department\_name varchar(20) not null);

create table employee(

employee\_id serial primary key,

salary int not null,

department\_id int references department(department\_id));

insert into department (department\_name) values

('first'),

('second'),

('third');

insert into employee (salary, department\_id) values

(901, 2),

(200, 2),

(123, 1);

select department\_name from department

where department\_id in

(

select department\_id from

(

select department\_id, sum(salary) as total

from employee group by department\_id

) T where total > 1000

)

select department\_name from department

join (select department\_id from employee

group by department\_id

having sum(salary) > 1000) as temp\_table

on temp\_table.department\_id = department.department\_id

select department\_name from department

join employee on department.department\_id = employee.department\_id

group by department\_name

having sum(salary) > 1000

**0 — Неподтверждённое чтение** (Read Uncommited, Dirty Read, грязное чтение) — чтение незафиксированных изменений своей транзакции и конкурирующих транзакций, возможны нечистые, неповторяемые чтения и фантомы

**1 — Подтверждённое чтение** (Read Commited) — чтение всех изменений своей транзакции и зафиксированных изменений конкурирующих транзакций, нечистые чтения невозможны, возможны неповторяемые чтения и фантомы

**2 — Повторяемое чтение** (Repeatable Read, Snapshot) — чтение всех изменений своей транзакции, любые изменения, внесённые конкурирующими транзакциями после начала своей, недоступны, нечистые и неповторяемые чтения невозможны, возможны фантомы

**3 — Упорядоченный** — (Serializable, сериализуемый) — [упорядоченные (сериализуемые)](http://ru.wikipedia.org/w/index.php?title=Сериализуемость&action=edit&redlink=1) транзакции. Идентичен ситуации, при которой транзакции выполняются строго последовательно, одна после другой. То есть транзакции, результат действия которых не зависит от порядка выполнения шагов транзакции (запрещено чтение всех данных, изменённых с начала транзакции, в том числе и своей транзакцией). Фантомы невозможны.

В postgresql

* READ\_UNCOMMITED = READ\_COMMITED
* REPEATABLE\_READ = SERIALIZABLE

Отличие SERIALIZABLE от READ\_COMMITED в том что при READ\_COMMITED если

1. Одновременно начались две транзакции
2. Первая изменила данные
3. Вторая изменила те же данные(вторая автоматически блокируется и ждет)
4. Первая закомитилась.(Вторая разблокировалась и перопределила изменения первой

То при SERILIZABLE когда первая комитится во второй будет ошибка(это вынудит сделать ролбек второй транзакции). Это происходит по тому что вторая транзакция началсь вместе с первой(пока не закомитилась первая). А если бы вторая началсь после того как закомитилась первая, то втроая бы выполнилась успешно.

Если две транзакции обновляют разные строки, то вторая транзакция не будет блокироваться, а сразу выполнится, никаких огранизений нет.

1. Author\_book

create table author(

author\_id serial primary key,

author\_name varchar(20) not null);

create table book(

book\_id serial primary key,

book\_name varchar(20) not null);

create table author\_book(

author\_id int references author(author\_id),

book\_id int references book(book\_id),

constraint pk primary key(author\_id, book\_id));

insert into author (author\_name) values

('a1'),('a2'),('a3');

insert into book (book\_name) values

('b1'),('b2');

insert into author\_book (author\_id, book\_id) values

(1, 1),(2,1),

(2,2);

select book.\* from book join

author\_book on book.book\_id = author\_book.book\_id

join author on author\_book.author\_id = author.author\_id

and author.author\_name like 'a2'



**Technology**

*Spring*

1. Рассказать про Spring;
2. Spring inversion of control.  Что такое. Как используется и для чего.
3. Spring dependency injection.  Что такое. Как используется и для чего.

*Hibernate*

1. Уровни кеша;
2. Способы запросов;
3. Как из select \* from a,b достать только объекты b;
4. Entities состояния;
5. Сессия в хибернейте;
6. sessionFactory.getCurrentSession() vs entityManagerFactory.createEntityManager()

1. У какого объекта (если использовать Hibernate) вызываются методы для работы с БД;
2. Если в хибернейте сделать:

Object o = transaction.get(id, table);

o.setName(name);

o.commit();

Сохранится ли name?

*JMS*

1. Что такое JMS

1. Различия Ant & Maven;
2. Команды в консоли для JVM, Maven, Ant;
3. Какой командой билдится проект?

*Web services*