lichMax

40 уровень

Санкт-Петербург

15.07.2017 10508 11

36 уровень. Ответы на вопросы к собеседованию по теме уровня

Опять-таки всё перерыл, ответов не нашёл. Что ж. Выложу свои, хотя я писал их чисто для себя, и по возможности — кратко. Но всё лучше, чем ничего. Итак, были такие вопросы:

Вопросы к собеседованию:

1. Что такое **MVC**?
2. Что такое **DAO** и **DTO**?
3. Что такое **POJO**?
4. Что такое **Entity**?
5. Какие **коллекции-списки** вы знаете?
6. Какие **коллекции-множества** вы знаете?
7. Что такое **map**, чем он отличается от «**словаря**»?
8. Что такое **Queue** и **Dequeue**?
9. Какие классы, реализующие **интерфейс Queeue** вы знаете?
10. Что такое **дерево**?

А теперь мои ответы:

1. **MVC** — это такой паттерн проектирования приложение, при котором приложение разделяется на три отдельные части:

на изменения модели. А контроллер интерпретирует действия пользователя, оповещая модель о необходимости изменений. Таким образом каждый из компонентов этой схемы слабо связан с другими компонентами, за счёт чего достигается гибкость программы. Чаще всего вся бизнес-логика размещается в модели, хотя иногда она содержится и в контроллере. В первом случае модель называют тонкой, в последнем — толстой.

2. **DAO** (Data Access Object) — это объект, основная задача которого сохранять данные в базу данные, а также извлекать их из неё. DTO (Data Transfer Object) - это объект, предназначенный для транспортировки данных. Поэтому его основная задача — хранить эти данные. Никакой логики он не содержится. Кроме того, он должен быть сериализуемым, так как транспортировка объектов обычно происходит с помощью сериализации-десериализации.

3. **POJO** переводится как "объект Java в старом стиле". Их противопоставляют EJB-объектами. Последние следуют специальной конвенции и обычно жёстко привязаны к какому-то конкретному enterprise-фреймворку (например, у них должен быть публичный конструктор без параметров, должен быть геттеры и сеттеры для полей, они должны быть сериализуемыми и т.д.). POJO — это, соответственно, обычный класс, не наследующий ни от каких специальных классов и не реализующий никаких специальных библиотек. Обычно POJO ничего особенного не делает, и содержит только состояние.

4. **Entity Bean** — это бин, цель которого хранить некоторые данные. В логику такого бина встроен механизм сохранения себя и своих полей в базу данных. Такой объект может быть уничтожен, а потом воссоздан из базы заново. Но кроме хранения данных у него нет никакой логики. А бин в свою очередь — это особый класс, которые должен выполнять следующие правила:

   a. Класс должен иметь конструктор без параметров, с модификатором доступа public. Такой конструктор позволяет инструментам создать объект без дополнительных сложностей с параметрами.
   b. Свойства класса должны быть доступны через get, set и другие методы (так называемые методы доступа), которые должны подчиняться стандартному соглашению об именах. Это легко позволяет инструментам автоматически определять и обновлять содержание bean’ов. Многие инструменты даже имеют специализированные редакторы для различных типов свойств.
   c. Класс должен быть сериализуем. Это даёт возможность надёжно сохранять, хранить и восстанавливать состояние bean независимым от платформы и виртуальной машины способом.
   d. Класс должен иметь переопределенные методы equals(), hashCode() и toString().

5. Все **коллекции-списки** реализуют интерфейс List<E> и наследуются от абстрактного класса AbstractList<E>. Среди них можно выделить ArrayList<E> и LinkedList<E7gt;. ArrayList7lt;E> — это список, основаный на массиве, а LinkedList<E> — это классический двусвязный список.

6. **Коллекции-множества** в Java реализуют интерфейс Set<E> и наследуются от AbstractSet<E>. Множества — это такие наборы данных, в которых все элементы уникальны. Среди них в Java есть HashSet, LinkedHashSet и TreeSet. Первая коллекция хранит свои объекты на основе хеш-кодов. Вторая — это модифицированная первая, в ней элементы ещё к тому же располагаются в связном списке, поэтому они все расположены в порядке добавления. Третья коллекция обеспечивает сортировку своих элементов.

7. **Map** — это вид коллекций, хранящих свои элементы в виде пар "ключ-значения". Причём все ключи должны быть уникальными. Среди реализаций есть HashMap и TreeMap. Первая реализация хранит элементы с использованием хэш-кодов. Вторая - хранит элементы в отсортированном по ключу порядке.

8. **Очередь (Queue)** — это структура данных, работающая по принципу "Первый вошёл — первый вышел". То есть элементы в очередь добавляются с одного конца, а извлекаются — с другого. **Deque** — это двусторонняя очередь. В этой очереди элементы можно добавлять как в начало, так и в конец, а также брать элементы тоже можно и из начала, и из конца очереди. Соответственно есть методы, которые позволяю положить элемент (это методы add(e) и offer(e)), и есть методы, позволяющие извлечь элемент из очереди (это такие методы, как remove() и poll()). Кроме того, есть методы, которые позволяют просто получить элемент из очереди без его удаления оттуда (это методы element() и peek()). В интерфейсе Deque дополнительно есть методы для добавления элементов в начало и конец очереди, извлечения элементов из начала или конца, а также получения элементов из начала или конца очереди (без их удаления из очереди).

9. Среди простых реализаций можно отметить **ArrayDeque**, **LinkedList** и **PriorityQueue**. Также существуют много классов в Concurrent Collections, которые реализуют эти два интерфейса (оба сразу или только один из них).

10. **Дерево** — это связный граф без петель и кратных рёбер. Обычно, если в дереве N вершин, то количество рёбер как

В программировании деревья используются довольно широко, и придумано уже много видов этого дерева. Одно из самых широкоприменяемых деревьев - это бинарное дерево. В этом дереве у каждого элемента имеется не больше двух потомков (то есть может быть от 0 до 2). Одной из разновидностью двоичного дерева является BST — бинарное дерево поиска. В этом дереве на элементы накладывается правило: левый потомок элемента должны быть по значению меньше его, а правый — по значению больше его или равен ему.
Также ещё существуют красно-чёрные деревья. Это разновидность двоичных деревьев поиска. В красно-чёрных деревьх вводится ещё одно свойство элемента — цвет. Цвет может быть чёрный или красный. Также каждое красно-чёрное дерево должно удовлетворять следующим требованиям:

1. корень дерева — чёрный;
2. узел либо красный, либо чёрный;
3. все листья дерева — чёрные;
4. оба потомка красно узла - чёрные;
5. всякий путь от данного узла до любого листового узла, являющегося его потомка, содержит одинаковое количество чёрных узлов.

Эти правила позволяют добиться сбалансированности дерева. Дерево сбалансированно, когда длина пути от корня до любого листового узла отличается не более, чем на 1. (То есть по-простому, в дереве нет перекосов и длинных ветвей.)

реализовать паттерн "*Наблюдатель*" который так же обеспечивает "слабое связывание" ипозволяет вести разработку и поддержку бизнес логики абсолютно ничего не зная и не завися от пользовательского интерфейса (Представления и Контроллера).
Так же ***Контроллер*** не должен на себя брать часть (или всю) бизнес логику, а просто должен обрабатывать пользовательские запросы и передавать их ***Модели*** (Фасаду).

Ответить +7

Алексей Уровень 41, Россия 19 августа 2019, 10:27

map - это не коллекция (не наследуется от Collection)

Ответить +1

IceBerg Уровень 41, Кривой Рог, Украина MASTER 12 сентября 2019, 11:49

Не согласен, Map является частью фреймворка Collections.

Ответить +2

Алексей Уровень 41, Россия 19 сентября 2019, 09:04

Я понимаю, ваше не согласие, но свою позицию я обозначил. Map не реализует интерфейс Collection, а значит не содержит методов обозначенных в этом интерфейсе. А значит ее нельзя назвать коллекцией, особенно если вы хотите избежать путаницы с этими методами.

Ответить 0

Даниил Salesforce Developer в **Viseven** MASTER 21 сентября 2019, 23:34

Ну Map действительно не являеться коллекцией в понимании объекта, но во фреймворк точно входит, а значит его (наверное) можно рассматривать как коллекцию в понимании структуры данных) В поддержку обоих сторон стоит упомянуть про Set, он как бы по обе стороны баррикад находиться.

Ответить 0

NodeOne Уровень 41 EXPERT 26 января 2019, 10:41

про очереди как то не правильно. Очередь может быть Первый вошел - первый вышел(FIFO) - обычная очередь как в магазине и Последний вошел - первый вышел(LIFO) - типичный пример stack.
А вы понятия смешали. И, имхо не критикующее ваш подход и не придирка, может стоит сделать в формате вопрос-ответ, как ранее все были сделаны. Как то не хватает этого.

Ответить +2

Leonid Java Developer в **ProgForce** EXPERT 10 июня 2020, 20:12

А вот это совсем неправильный наезд на автора - stack не является очередью! Иерархия:
java.lang.Object
java.util.AbstractCollection<E>
java.util.AbstractList<E>
java.util.Vector<E>
java.util.Stack<E>

Ответить +1

Владимир Панов Уровень 40, Харьков, Украина 23 августа 2018, 21:57

Модель или контроллер - тонкий и толстый)))

Ответить 0

проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский