Часть первая ответника, Java

GoncharovA Follow
Jan 1 · 7 min read

Не так давно моя хорошая подруга Веселина Зацепина опубликовала список вопросов, которые часто задают на собеседованиях на должность разработчика приложений для платформы Android. Это благородное, очень правильное начинание. Наличие списка вопросов даёт уже даёт толику уверенности перед собеседованием.
Сегодня мне хотелось бы опубликовать первую часть списка ответов на эти часто задаваемые вопросы. Разумеется, ответы эти краткие: предназначены только чтобы зазубрить и озвучить текст. Да, просто текст. Не для обучения или понимания; Просто чтобы сопоставить множество текстовых вопросов и текстовых ответов. Получится карта<Строка, Строка>, из которой нужно будет лишь выдавать по ключу соответствующие значения.

. . .

Первым блоком идут вопросы о языке программирования Java.

1. Модификаторы доступа Java (в порядке от private до public):

private — ограничивает видимость данных и методов пределами одного класса.

protected — Поля и методы, обозначенные модификатором доступа protected, будут видны в пределах всех классов, находящихся в том же пакете, что и наш и в пределах всех классов-наследников нашего класса.

default (package visible) — `default` или, как его еще называют, `package visible`. Он не обозначается ключевым словом, поскольку установлен в Java по умолчанию для всех полей и методов. Действует также, как и protected, за исключением наследования.

public — Не накладывает никаких ограничений на доступ; предназначаются для конечного пользователя.

2. Реализация «кучи» (где хранятся объекты):

Эта область памяти используется для объектов и классов. Новые объекты всегда создаются в куче, а ссылки на них хранятся в стеке. Эти объекты имеют глобальный доступ и могут быть получены из любого места программы.

Эта область памяти разбита на несколько более мелких частей, называемых поколениями:

Young Generation — область где размещаются недавно созданные объекты. Когда она заполняется, происходит быстрая сборка мусора.

Old (Tenured) Generation — здесь хранятся долгоживущие объекты. Когда объекты из Young Generation достигают определенного порога “возраста”, они перемещаются в Old Generation.

Permanent Generation — эта область содержит метаинформацию о классах и методах приложения, но начиная с Java 8 данная область памяти была упразднена.

Помимо рассмотренных ранее, куча имеет следующие ключевые особенности:Когда эта область памяти полностью заполняется, Java бросает java.lang.OutOfMemoryError Доступ к ней медленнее, чем к стеку. Эта память, в отличие от стека, автоматически не освобождается. Для сбора неиспользуемых объектов используется сборщик мусора. В отличие от стека, куча не является потокобезопасной и ее необходимо контролировать, правильно синхронизируя код.

3. Где и как используются методы equals() и hashcode():

Метод equals() являются ли два объекта одного происхождения логически равными. Создатель класса сам определяет характеристики, по которым проверяется равенство объектов этого класса.

Объекты должны быть экземплярами одного класса и не должны быть null. Переопределяя метод equals(), обязательно соблюдение этих требований:

Рефлексивность: Любой объект должен быть equals() самому себе.

Симметричность: Если a.equals(b) == true, то и b.equals(a) должно возвращать true.

Транзитивность: Если два объекта равны какому-то третьему объекту, значит, они должны быть равны друг и другу. Если a.equals(b) == true и a.equals(c) == true, значит проверка b.equals(c) тоже должна возвращать true.

Постоянность: Результаты работы equals() должны меняться только при изменении входящих в него полей. Если данные двух объектов не менялись, результаты проверки на equals() должны быть всегда одинаковыми.

Сравнение с null для любого объекта a.equals(null) должно возвращать false.

Метод hashCode() возвращает для любого объекта 32-битное число типа int. Если два объекта равны (т.е. метод equals() возвращает true), у них должен быть одинаковый хэш-код. Проверка по hashCode() должна идти первой для повышения быстродействия. Если метод hashCode() вызывается несколько раз на одном и том же объекте, каждый раз он должен возвращать одно и то же число. Одинаковый хэш-код может быть у двух разных объектов. Методы equals и hashCode необходимо переопределять вместе.

4. Что будет, если не переопределить метод hashcode():

Тогда с точки зрения метода equals два объекта будут логически равны, в то время как с точки зрения метода hashCode они не будут иметь ничего общего.

5. Реализация HashMap:

HashMap — основан на хэш-таблицах, реализует интерфейс Map (что подразумевает хранение данных в виде пар ключ/значение). Ключи и значения

могут быть любых типов, в том числе и null. Сначала ключ проверяется на равенство с null. Если это проверка вернула true, будет вызван метод putForNullKey(value). Далее генерируется хэш на основе ключа. Для генерации используется метод hash(hashCode), в который передается key.hashCode(). С помощью метода indexFor(hash, tableLength), определяется позиция в массиве, куда будет помещен элемент. Теперь, зная индекс в массиве, мы получаем список (цепочку) элементов, привязанных к этой ячейке. Хэш и ключ нового элемента поочередно сравниваются с хэшами и ключами элементов из списка и, при совпадении этих параметров, значение элемента перезаписывается. Если же предыдущий шаг не выявил совпадений, будет вызван метод addEntry(hash, key, value, index) для добавления нового элемента.

6. Как разрешается коллизия в HashMap (метод цепочек или открытая адресация):

Разрешение коллизий при помощи цепочек. Каждая ячейка массива H является указателем на связный список (цепочку) пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются цепочки длиной более одного элемента.

7. Виды исключений в Java:

Unchecked(Error, RunTimeException); Checked(Exception)

8. Виды коллекций Java (перечисляете всё, что знаете):

List, Set, Queue, Deque, Map.

9. Типы ссылок Java и где используются, в порядке убывания жёсткости:

Сильные, они же обычные, нужны для указания на объекты, которые должны обязательно оставаться в памяти всё то время, что эти ссылки на него существуют. Если не складывается, получите OutOfMemoryError.

Мягкие ссылки полезны для кэшей, чувствительных к доступному объёму оперативной памяти. Объекты по ним могут зачиститься, но только в случае необходимости. Например, если нужно насоздавать ещё объектов с сильными

ссылками, а уже негде, лучше освободить кэш и замедлить работу, чем уронить процесс напрочь.

Слабые ссылки полезны для сопоставления объектов чему-нибудь без удерживания их от зачистки когда они больше не нужны (а-ля Map<Ключ, WeakRef<Значение>>). На возможность зачистки они не влияют вообще никак, слабые ссылки будут очищены при очередном запуске сборщика.

Фантомные ссылки возникают, когда объект уже признан мусором, финализирован и находится в процессе зачистки, о чём можно узнать с помощью класса Cleaner и выполнить в это время какие-то собственные действия.

10. Способы синхронизации Java:

Синхронизация относится к многопоточности. Синхронизированый блок кода может быть выполнен только одним потоком одновременно. Синхронизация достигается в Java использованием зарезервированного слова synchronized. Вы можете использовать его в своих классах, определяя синхронизированные методы или блоки. Вы не сможете использовать synchronized в переменных или атрибутах в определении класса.

Блокировка на уровне объекта-это механизм, когда требуется синхронизировать нестатический метод или нестатический блок кода таким образом, чтобы только один поток мог выполнить блок кода на данном экземпляре класса. Это всегда должно быть сделано, чтобы сделать потокобезопасными данные уровня экземпляра.

Блокировка уровня класса предотвращает ввод нескольких потоков в синхронизированный блок в любом из всех доступных экземпляров во время выполнения. Это означает, что если во время выполнения есть 100 экземпляров DemoClass, то только один поток сможет выполнить demoMethod() в любом из экземпляров одновременно, а все остальные экземпляры будут заблокированы для других потоков.

11. Volatile — что это:

Запрещает помещать значение переменной в кэш. Ключевое слово volatile указывается для поля для того, чтобы указать компилятору, что все операции присвоения этой переменной и все операции чтения из неё должны быть атомарными. Более того, присвоение значения этой переменной имеет связь happens-before (произошло-до) для последующих чтений из этой переменной для любых потоков, то есть после присвоения нового значения переменной все потоки увидят это новое значение.

Дело в том, что Java позволяет потокам в целях производительности сохранять локальные копии переменной для каждого потока, который её использует (например в кешах или регистрах процессора). В таком случае после записи другим потоком нового значения в исходную переменную, первый поток будет видеть свою локальную копию со старым значением.

Использование ключевого слова volatile гарантирует, что все потоки всегда будут использовать общее, исходное значение, и они будут видеть изменения этого исходного значения другими потоками сразу же. Аналогично все изменения переменных, произошедшие внутри sychronized-методов и synchronized-блоков, а также блоков с другими блокировками вроде реализаций интерфейса java.util.concurrent.locks.Lock после выхода из блокировки будут гарантировано видны любым другим потокам после взятия блокировки над тем же самым объектом, но если более сложные блокировки не нужны, то можно использовать volatile.

12. StringBuilder vs String:

Благодаря неизменности, хэшкод экземпляра класса String кэшируется. Его не нужно вычислять каждый раз, потому что значения полей объекта никогда не изменятся после его создания. Это дает высокую производительность при использовании данного класса в качестве ключа для HashMap.

StringBuilder — изменяемый класс, поэтому при работе с ним не возникает такого же количества мусора в памяти, как со String. Поэтому если над строками проводится много модификаций, лучше использовать StringBuilder.

13. Назовите синхронизированную версию HashMap (Hashtable, но устарела):

SynchronizedMap и ConcurrentHashMap.

Методы SynchronizedMap удерживают блокировку на объекте, в то время как ConcurrentHashMap есть понятие «чередование блокировок», когда вместо блокировок удерживаются блоки содержимого. Таким образом улучшается масштабируемость и производительность.

14. Реализация ArrayList в Java (на базе массива):

Класс ArrayList реализует интерфейс List и может менять свой размер во время исполнения программы, при этом не обязательно указывать размерность при создании объекта. Элементы ArrayList могут быть абсолютно любых типов в том числе и null. Внутри у него находится массив, в котором и хранятся элементы. УArrayList'а есть специальный механизм по работе с ним: Когда этот внутренний массив заполняется, ArrayList создает внутри себя новый массив. Его размер = (размер старого массива * 1,5) +1. Все данные копируются из старого массива в новый Старый массив удаляется сборщиком мусора.

15. Модификатор final в Java — где используется и что даёт:

Применяться к классам, методам, переменным (в том числе аргументам методов) Для класса это означает, что класс не сможет иметь подклассов, т.е. запрещено наследование. Для метода final означает, что он не может быть переопределен в подклассах. Для переменных примитивного типа это означает, что однажды присвоенное значение не может быть изменено. Для ссылочных переменных это означает, что после присвоения объекта, нельзя изменить ссылку на данный объект. Ссылку изменить нельзя, но состояние объекта изменять можно.

. . .

Кроме того, хотелось бы добавить вопрос из своего опыта:

Какие методы имеются у класса Object?

public String toString() — Возвращает строковое представление объекта.

public native int hashCode() и public boolean equals(Object obj) — Пара методов, которые используются для сравнения объектов.

public final native Class getClass() — Возвращает специальный объект, который описывает текущий класс.

public final native void notify(),
public final native void notifyAll(),
public final native void wait(long timeout),
public final void wait(long timeout, intnanos),
public final void wait() — Методы для контроля доступа к объекту из различных нитей. Управление синхронизацией нитей.

protected void finalize() — Метод позволяет «освободить» родные не-Java ресурсы: закрыть файлы, потоки и т.д.

protected native Object clone() — Метод позволяет клонировать объект: создает дубликат объекта.

. . .

Все ответы собраны из следующих источников:

https://javarush.ru/

https://docs.oracle.com/javase/

https://coderoad.ru/

http://javastudy.ru/interview/collections/

Android Development Java