Ответы нужно давать лаконичные, достаточно продемонстрировать понимание предмета.

|  |  |
| --- | --- |
| Ф.И.О. | Бадьин Роман Сергеевич |
| **Java** | |
| Что такое переопределение метода? | Переопределение метода (method overriding) в Java — это механизм, который позволяет подклассу предоставить свою собственную реализацию метода, который уже определён в его суперклассе.   * Метод в подклассе должен иметь ту же сигнатуру, что и метод в суперклассе (то есть то же имя, тип возвращаемого значения и параметры). * Метод в подклассе должен быть доступнее или равным по уровню доступа методу в суперклассе. * Переопределение позволяет подклассу специфически реагировать на вызовы метода, предоставляя специализированную реализацию.   Вот как я переопределил метод интерфейса в классе который в свою очередь имплементирует интерфейс  @Override public List<Flight> filter(List<Flight> flights) {  LocalDateTime now = LocalDateTime.*now*();  return flights.stream()  .filter(flight -> flight.getSegments().get(0).getDepartureDate().isAfter(now))  .collect(Collectors.*toList*()); } |
| Какие бывают виды классов? | * **Обычные классы (Regular classes):** Это стандартные классы, создаваемые для хранения данных и методов. * **Абстрактные классы (Abstract classes):** Классы, которые не могут быть инстанцированы и могут содержать абстрактные методы. * **Конкретные классы (Concrete classes):** Классы, которые предоставляют реализацию для всех методов. * **Внутренние классы (Inner classes):** Классы, определённые внутри другого класса. Включают в себя:   + Статические внутренние классы (Static nested classes)   + Вложенные классы-члены (Member inner classes)   + Локальные внутренние классы (Local inner classes)   + Анонимные внутренние классы (Anonymous inner classes) * **Классы-обёртки (Wrapper classes):** Классы, которые оборачивают примитивные типы данных, такие как Integer, Double и т. д. * **Запечатанные классы (Sealed classes) (Java 17):** Классы, которые ограничивают другие классы от расширения.  (public **sealed** class Shape **permits** Circle, Square {) |
| Как и зачем можно использовать модификатор final? | Модификатор **final** может использоваться для:   * **Классов:** Запрещает наследование класса.   public final class Constants {  // code  }   * **Методов:** Запрещает переопределение метода в подклассах.   public class Parent {  public final void display() {  // code  }  }   * **Переменных:** объявляет константу. Значение переменной не может быть изменено после инициализации.   public class Example {  public static final int MAX\_SIZE = 100;  } |
| Какие есть варианты использования ключевого слова try? | Ключевое слово **try** используется для обработки исключений и имеет несколько вариантов:   * **Простое использование catch:**   try {  // код, который может вызвать исключение  } catch (ExceptionType e) {  // обработка исключения  }   * Использование с finally:   try {  // код, который может вызвать исключение  } catch (ExceptionType e) {  // обработка исключения  } finally {  // код, который выполнится в любом случае  }   * **Try-with-resources (начиная с Java 7):** Используется для автоматического закрытия ресурсов, таких как файлы, соединения и т. д.   try (Resource resource = new Resource()) {  // использование ресурса  } catch (ExceptionType e) {  // обработка исключения  } |
| Какие есть стандартные реализации интерфейса List и в каких ситуациях их нужно использовать? | Основные реализации интерфейса List в Java:   * **ArrayList:**   + **Когда использовать:** Если требуется быстрая произвольная выборка элементов и частое чтение.   + **Особенности:** Основан на массиве, предоставляет амортизированное время выполнения O(1) для доступа по индексу, но может быть медленным при частых вставках/удалениях не в конце списка. * **LinkedList:**   + **Когда использовать:** Если требуется частое добавление и удаление элементов в начале или середине списка.   + **Особенности:** Основан на двусвязном списке, операции вставки и удаления занимают O(1), но доступ по индексу выполняется за O(n). * **CopyOnWriteArrayList:**   + **Когда использовать:** Если нужен потокобезопасный список, который редко изменяется, но часто читается.   + **Особенности:** Создаёт копию массива при каждой модификации, что делает его неподходящим для частых изменений. |
| В чём основная идея стримов из пакета java.util.stream? | Стримы (streams) в Java предоставляют функциональный подход к обработке коллекций и других источников данных.  **Основные моменты:**   * **Ленивость:** Операции со стримами не выполняются сразу, а только при вызове терминальной операции (например, collect, forEach). * **Функциональные операции:** Включают filter, map, reduce, collect, sorted, и т. д. * **Параллелизм:** Легко поддерживают параллельную обработку данных с помощью метода parallelStream. * **Чистый код:** Улучшают читаемость и поддержку кода.   List<String> names = Arrays.asList("Алиса", "Bob", "Charlie");  List<String> filteredNames = names.stream()  .filter(name -> name.startsWith("A"))  .collect(Collectors.toList()); |
| **Разное** | |
| Каким критериям должна удовлетворять «хорошая» хэш-функция? | Хорошая хэш-функция должна:   * **Производить равномерное распределение:** минимизировать количество коллизий. * **Быть быстрой:** иметь низкую вычислительную сложность. * **Быть детерминированной:** всегда возвращать одинаковый хэш для одинакового ввода.(идемпотичность) * **Минимизировать коллизии:** Разные входные данные должны приводить к разным хэшам. |
| В чём причина популярности и широкого распространения кодировки UTF-8? | UTF-8 популярен из-за:   * **Совместимость с ASCII:** Первые 128 символов совпадают с ASCII, что упрощает работу с существующими системами. * **Эффективность:** Кодирование символов занимает от 1 до 4 байт, что эффективно для большинства текстов. * **Глобальная поддержка:** Поддерживает все символы Unicode, подходя для международных приложений. * **Простота:** Прост в использовании и широко поддерживается. |
| Сравните форматы XML и JSON. Когда какой использовать? | **XML (eXtensible Markup Language):**   * **Когда использовать:** Для сложных документов с иерархическими данными, когда нужно поддерживать схемы, валидацию и метаданные. * **Особенности:** Более вербоузен, поддерживает комментарии, атрибуты.   **JSON (JavaScript Object Notation):**   * **Когда использовать:** Для простого обмена данными между клиентом и сервером, особенно в веб-приложениях. * **Особенности:** Более лёгкий и компактный, проще для чтения и написания, нативно поддерживается JavaScript. Активно используется в новых проектах. |
| Опишите что будет происходить «под капотом» после ввода адреса сайта в браузере и нажатия Enter? |  **DNS-запрос:** Браузер проверяет кэш DNS и отправляет запрос на DNS-сервер для получения IP-адреса домена. Днс помогает сопоставить символьное название и ip адрес сайта.   **Установка соединения:** Браузер инициирует TCP/IP соединение с сервером по полученному IP-адресу.   **Отправка HTTP-запроса:** Браузер отправляет HTTP-запрос на сервер.   **Обработка запроса:** Веб-сервер обрабатывает запрос, возможно, обращается к базе данных, выполняет логику и генерирует ответ.   **Получение ответа:** Браузер получает HTTP-ответ от сервера, содержащий HTML, CSS, JavaScript и другие ресурсы.   **Рендеринг страницы:** Браузер обрабатывает HTML, загружает и применяет CSS, выполняет JavaScript, отображает контент на экране.   **Дополнительные запросы:** Браузер выполняет дополнительные запросы для ресурсов (изображений, стилей, скриптов), необходимых для полной загрузки страницы. |