Ответы нужно давать лаконичные, достаточно продемонстрировать понимание предмета.

Ф.И.О.	Четвёркин Александр Сергеевич
Java	
Что такое переопределение метода?	Изменение / реализация метода родительского класса в потомке. Необходимые условия: переопределяющий метод должен иметь то же имя, параметры (количество, тип, порядок), и тип возвращаемого значения (либо подкласс этого типа). Желательно помечать переопределяющие методы аннотацией @Override.

Какие бывают виды классов?	● Concrete: класс, содержащий полную реализацию всех объявленных методов. Мы можем создавать объекты этого класса и наследоваться от него.
	● Final: класс помеченный как final является неизменяемым, все его методы приобретают свойство final а также от такого класса нельзя наследоваться.
	● Abstract: класс помеченный как abstract является чем-то вроде шаблона для конкретных классов, зачастую такой класс содержит абстрактные методы и/или реализует (implements) какие-то интерфейсы. Мы не можем инстанцировать этот тип класса.
	● Inner: класс объявленный внутри другого (внешнего) класса или его метода.
	● Static: вложенный класс, помеченный как static может содержать только статические члены а также имеет доступ только к статическим членам внешнего класса. Мы не можем инстанцировать этот тип класса.
	● Anonymous: если нам нужно воспользоваться классом только один раз, мы можем реализовать/переопределить все его методы прямо во время инстанцирования:
	`AbstractClass example = new AbstractClass() {`
	`  //@Override all abstract methods`
	`};`
	● Singleton: один из шаблонов проектирования. Этот класс имеет приватный конструктор, а метод getInstance() возвращает ссылку на единственный экземпляр данного класса.

Как и зачем можно использовать модификатор final?	Модификатор final применим к полям, методам и классам. Поле, помеченное как final становится неизменяемым и должно быть инициализировано в конструкторе класса (однако если это ссылка на объект, сам объект может изменяться, путем вызова соотв. методов). Применимо к методам этот модификатор запрещает переопределение данного метода в дочерних классах. Классы, помеченные как final не подлежат наследованию, все их методы также неявно становятся final.

Какие есть варианты использования ключевого слова try?	try {
	  // код, который может кинуть исключение
	} catch(Exception ignored) {} finally{}
	
	try-with-resources:
	// конструктор может кинуть IOException
	try (FileWriter fw = new FileWriter("output.txt")) {
	  fw.write("My brain hurts!");
	} catch (IOException ignored) {}
	
	Система исключений позволяет отделить код с бизнес-логикой от кода обработки ошибок. (В отличие от C, где порой приходилось протаскивать -1 через весь стек вызовов FUUUUUUUU!!)

Какие есть стандартные реализации интерфейса List и в каких ситуациях их нужно использовать?	ArrayList<E>: динамический массив. Плюсы: доступ к любому элементу за O(1), cache-friendly т.к. все элементы хранятся подряд, одним блоком. Динамически меняет свой размер по мере необходимости. Минусы: при превышении capacity создается новый массив (50% больше) с копированием всех элементов. Также операция вставки/удаления произвольного элемента занимает O(n). Vector<E>: то же что и ArrayList с несколькими нюансами: vector поддерживает синхронизацию доступа к элементам, в многопоточном приложении это позволяет избежать порчи данных и состояний гонки (race conditions). Однако синхронизация делает его более медленным по сравнению с

	ArrayList. При превышении capacity вектор увеличивает свой размер в два раза. LinkedList<E>: двусвязный список. Плюсы: вставка/удаление любого элемента за O(1), не требует копирования элементов при модификации: достаточно изменить ссылки prev и next соответствующей ноды. Минусы: доступ к элементам за O(n), т.к. чтобы добраться до нужного элемента, приходится проходить по всему списку (с начала или с конца). Плохо дружит с кэшем из-за нарушения локальности данных: ноды списка могут находиться в куче где угодно. Stack<E>: тоже может использоваться в качестве реализации List-а, но логически является отдельной структурой данных типа LIFO. С помощью двух стеков можно реализовать очередь. Очень неэффективную очередь. :D
В чём основная идея стримов из пакета java.util.stream?	Стримы в джаве предлагают функциональный подход к обработке коллекций объектов: мы указываем, что должен делать код, вместо того как он должен это делать. Они позволяют очень лаконично описывать разнообразные действия: сортировку, фильтрацию, применение функции к каждому объекту и т.д. Также они предоставляют возможность параллельной обработки .parallelStream()/.parallel()
Разное	
Каким критериям должна удовлетворять «хорошая» хэш-функция?	● Эффективность: хеш-функция должна вычисляться за константное время O(1). ● Детерминированность: если входные данные логически идентичны, они должны производить одинаковый хеш. ● Униформность: итоговая длина хеша не должна зависеть от размера / типа входных данных, сами хеши должны распределяться равномерно. ● Минимальное количество коллизий (одинаковый хеш на разных входных данных).
В чём причина популярности и широкого распространения кодировки UTF-8?	Полная обратная совместимость с ASCII а также поддержка всех возможных языков за счет использования переменной длины кодирования.

Сравните форматы XML и JSON. Когда какой использовать?	JSON — это формат данных, XML — язык разметки. Оба формата используются для получения данных от веб-сервера, однако XML гораздо старше. ● XML поддерживает пространства имён, комментарии, метаданные и разнообразные сложные типы данных (диаграммы, картинки и т.п.) однако из-за системы тегов его сложнее парсить и читать. XML файлы объемнее и медленнее при обработке запросов. Нет нативной поддержки массивов. ● JSON хранит данные в виде карты (ключ: значение), поддерживает массивы, более прост для восприятия и парсинга. Обеспечивает более быструю обработку запросов. Поддерживает только примитивные типы данных: строки, числа, массивы и объекты (из примитивов).
Опишите что будет происходить «под капотом» после ввода адреса сайта в браузере и нажатия Enter?	После того как браузер получил URL он разбивает его на несколько частей: протокол (http/https), доменное имя (host), адрес ресурса (если он есть). Затем необходимо узнать, какой IP адрес закреплен за данным хостом, для этого происходит обращение к DNS-серверу (если адрес ранее уже использовался, то к локальному кэшу), иногда к целой цепочке, пока не будет найден авторитативный сервер для этого домена. Далее происходит установка tcp-соединения через дефолтный порт для данного протокола (http: 80, https: 443) — так называемое трехстороннее рукопожатие. Оно представляет собой обмен пакетами с флагами: SYN ->, SYN/ACK <-, ACK ->. Если используется https, то далее идёт tls handshake: обмен сертификатами, проверка подлинности сертификатов, обмен публичными ключами — это позволяет установить безопасное соединение. Далее клиент формирует http запрос: в заголовке указывается метод (GET/POST/…), адрес ресурса /…, версия http протокола используемая клиентом (1.0/2.0) и другая сервисная информация. После того, как запрос сформирован и отправлен, клиент получает ответ от сервера: 2xx — успех, 3xx/4xx/5xx — произошла ошибка. Если всё хорошо, браузер получает запрошенную html страницу парсит её в несколько проходов, составляет дерево графических объектов, рендерит и отображает страницу.