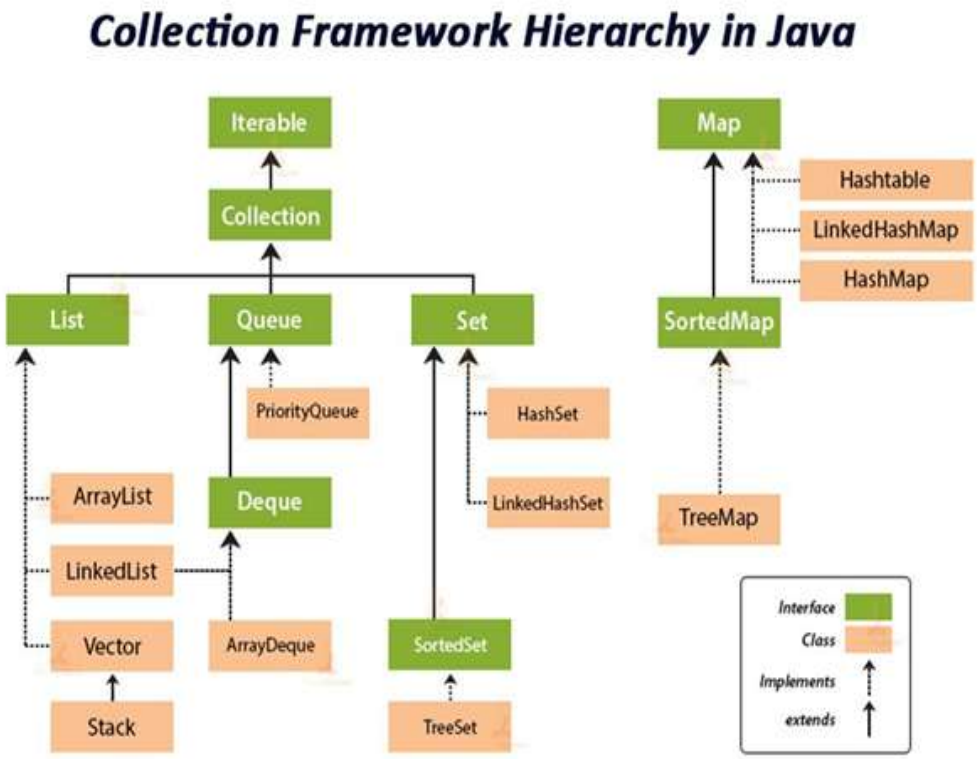
Оглавление

	Дженерики	
2	**1. Что такое дженерики?**	"Дженерики – это параметризованные типы. С их помощью можно объявлять классы, интерфейсы и методы, в которых тип данных указан в виде параметра. Используя дженерики, можно создать единственный класс, который будет автоматически работать с разными типами данных. Эта информация доступна только на этапе компиляции и стирается в runtime, и в байт код попадет только информация о том, что в программе есть некий список List<Object> list вместо List<String> list, например. Появились в версии 1.5 " на "9"
3	**2. Для чего нужны дженерики?**	Для строгой типизации и проверки на этапе компиляции. Дженерики позволяют передавать тип объекта компилятору в форме <тип>. Таким образом, компилятор может выполнить все необходимые действия по проверке типов во время компиляции, обеспечивая безопасность по приведению типов во время выполнения.
4	**3. Стирание типов**	- суть заключается в том, что внутри класса не хранится никакой информации о типе-параметре. Эта информация доступна только на этапе компиляции и стирается (становится недоступной) в runtime.
5	**4. Что такое сырые типы (raw type)?**	Сырые типы — это типы без указания типа в фигурных скобках (List list = new ArrayList<>()), они использовались до появления дженериков. Не указывая их, под капотом используется Object.
6	**5. Что такое вайлдкарды?**	Маске (wildcard) можно задать ограничения: -**“? extends T”** (для получения в методе) - объект, который наследуется от T, либо сам T – ковариантность. Если контейнер объявлен ? extends T, то можно только читать значения. В список нельзя ничего добавить, кроме null. -**“? super T”** (для отдачи в методе) - любой объект подтипа T, включая T – контравариантность. Нельзя прочитать элемент из контейнера с wildcard ? super, кроме объекта класса Object При использовании ? мы сообщаем компилятору, чтобы он игнорировал информацию о типе, т.е. <?> - неограниченный символ подстановки. <?> означает то же что и <? extends Object>, т.е. принимает всё. Это можно обойти, создав обобщенный метод, объявленный с переменной типа T.
7	**6. Расскажите**	Producer Extends Consumer Super wildcard подстановочный знак: Если мы объявили wildcard с extends, то это producer. Он только «производит»,

	про принцип PECS	предоставляет элемент из контейнера, а сам ничего не принимает. Если же мы объявили wildcard с super — то это consumer. Он только принимает, а предоставить ничего не может. Иначе говоря: Если вы только получаете объекты из дженерик-коллекции - это producer и надо использовать extends. Если вы только кладете объекты в коллекцию - это consumer и надо использовать super. Если вы делаете оба эти действия, то не надо использовать ни super, ни extends.
8	**Коллекции**	
9	**1.Что такое «коллекция»?**	Коллекция – это объект, который содержит набор объектов одного типа. Каждый из этих объектов в коллекции называется элементом.
10	**2.Расскажите про иерархию коллекций**	
11	**3.Почему Map — это не Collection, в то время как List и Set являются Collection?**	Коллекция (List и Set) представляет собой совокупность некоторых элементов (обычно экземпляров одного класса).Map -это совокупность пар "ключ"-"значение". У map нет итерабл, не понятно по чему проводить итерацию
12	**4.В чем разница между java.util.Collection и java.util.Collect**	Класс **java.util.Collections содержит исключительно статические методы для работы с коллекциями.** В них входят методы, реализующие полиморфные алгоритмы (такие алгоритмы, использование которых возможно с разными видами структур данных), "оболочки", возвращающие новую коллекцию с инкапсулированной указанной структурой данных и некоторые другие методы. **java.util.Collection** - **это корневой интерфейс Java Collections Framework**. Этот интерфейс в основном применяется там, где требуется высокий уровень абстракции, например, в классе java.util.Collections.

	ions?	
13	**5.Какая разница между итераторами с fail-fast и fail-safe поведением? (С примерами)**	Итератор fail-safe не вызывает исключений при изменении структуры коллекции, потому что работает с её клоном. Пример fail-safe - CopyOnWriteArrayList и итератор keySet коллекции ConcurrentHashMap. Итератор fail-fast генерирует исключение ConcurrentModificationException, если коллекция меняется во время итерации, но работает быстрее. Пример fail-fast - Vector и Hashtable.
14	**6.Чем различаются Enumeration и Iterator?**	Iterator имеет больше методов работы с коллекциями и был специально введен в java2, вместо Enumeration(interface). Рекоммендуется юзать Iterator. Оба интерфейса предназначены для обхода коллекции, но есть различия: -с помощью Enumeration нельзя добавлять/удалять элементы; -в Iterator исправлены имена методов для повышения читаемости кода (Enumeration.hasMoreElements() соответствует Iterator.hasNext(), Enumeration.nextElement() соответствует Iterator.next() и т.д); -Enumeration присутствуют в устаревших классах, таких как Vector/Stack, тогда как Iterator есть во всех современных коллекциях.
15	**7.Как между собой связаны Iterable, Iterator и «for-each»?**	Интерфейс Iterable имеет метод - iterator(), с типом возвращаемого значения - интерфейс Iterator. Экземпляры классов, реализующих интерфейс Iterable, могут использоваться в цикле foreach.
16	**8.Можно ли итерируясь по ArrayList удалить элемент? Какое вылетит исключение?**	Можно, но нужно использовать iterator.remove(). Иначе при прохождении по ArrayList в цикле for сразу после удаления элемента будет ConcurrentModificationException.
17	**9.Как поведёт себя коллекция, если вызвать iterator.remove()?**	Этот метод удаляет текущий элемент. Важный момент заключается в том, что сначала этот элемент необходимо получить с помощью метода next(), если мы вызовем метод remove() до метода next(), то мы получим IllegalStateException.
18	**10.Чем Set отличается от List?**	Set не добавляет новых методов, только вносит изменения унаследованные. В частности, метод add() добавляет элемент в коллекцию и возвращает true, если не было такого элемента. Разрешено наличие только одной ссылки типа null.
19	**11.Расскажите про**	Интерфейс Set расширяет интерфейс Collection. Set не добавляет новых методов, только вносит изменения унаследованные. Set - неупорядоченный набор неповторяющихся элементов В частности, метод add() добавляет элемент в коллекцию и возвращает true, если

<table>
<tr><td></td><td>интерфейс Set.</td><td>не было такого элемента.
Разрешено наличие только одной ссылки типа null.</td></tr>
<tr><td>20</td><td>12.Расскажите про реализации интерфейса Set</td><td>В HashSet порядок добавления элементов будет непредсказуемым - используется хэширование для ускорения выборки.
В TreeSet объекты хранятся отсортированными по возрастанию из-за применения к/ч дерева.
LinkedHashSet хранит элементы в порядке добавления.</td></tr>
<tr><td>21</td><td>13.В чем отличия TreeSet и HashSet?</td><td>HashSet быстрее, чем TreeSet .
В HashSet элементы в случайном порядке, в TreeSet в отсортированном.
HashSet обеспечивает постоянную производительность - O(1) - для большинства операций, таких как add () , remove () и contains () , по сравнению с временем log(n), предлагаемым TreeSet.</td></tr>
<tr><td>22</td><td>14.Чем LinkedHashSet отличается от HashSet?</td><td>Основное различие в том, что LinkedHashSet сохраняет порядок вставки элементов, а HashSet - нет. В основе LinkedHashSet лежит LinkedHashMap вместо HashMap. Благодаря этому порядок элементов при обходе коллекции является идентичным порядку добавления элементов</td></tr>
<tr><td>23</td><td>15.Что будет, если добавлять элементы в TreeSet по возрастанию?</td><td>TreeSet все равно в каком порядке вы добавляете в него элементы, так как в основе TreeSet лежит красно-черное дерево, которое умеет само себя балансировать и хранить элементы по возрастанию.</td></tr>
<tr><td>24</td><td>16.Как устроен HashSet, сложность основных операций.</td><td>HashSet — временная сложность основных операций
<table><tr><th></th><th>Поиск</th><th>Вставка</th><th>Удаление</th></tr><tr><td>Метод</td><td>contains(object)</td><td>add(object)</td><td>remove(index)</td></tr><tr><td>Среднее</td><td>O(1)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Худшее (до Java 8)</td><td>O(n)</td><td>O(n)</td><td>O(n)</td></tr><tr><td>Худшее (Java 8+)</td><td>O(log₂(n))</td><td>O(log₂(n))</td><td>O(log₂(n))</td></tr></table>Все классы, реализующие интерфейс Set, внутренне поддерживаются реализациями Map. HashSet хранит элементы с помощью HashMap. Значение, которые мы передаем в HashSet, является ключом к объекту HashMap, а в качестве значения используется Object.</td></tr>
<tr><td>26</td><td>17.Как устроен LinkedHashSet, сложность основных операций.</td><td>LinkedHashSet — временная сложность основных операций
<table><tr><th></th><th>Поиск</th><th>Вставка</th><th>Удаление</th></tr><tr><td>Метод</td><td>contains(object)</td><td>add(object)</td><td>remove(index)</td></tr><tr><td>Среднее</td><td>O(1)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Худшее (до Java 8)</td><td>O(n)</td><td>O(n)</td><td>O(n)</td></tr><tr><td>Худшее (Java 8+)</td><td>O(log₂(n))</td><td>O(log₂(n))</td><td>O(log₂(n))</td></tr></table>В его основе лежит LinkedHashMap. Благодаря этому порядок элементов при обходе коллекции является идентичным порядку добавления элементов</td></tr>
</table>

28	**18.Как устроен TreeSet, сложность основных операций.**	**TreeSet** – временная сложность основных операций | | Поиск | Вставка | Удаление | |---|---|---|---| | Метод | contains(object) | add(object) | remove(index) | | Среднее | $O(\log_2(n))$ | $O(\log_2(n))$ | $O(\log_2(n))$ | | Худшее | $O(\log_2(n))$ | $O(\log_2(n))$ | $O(\log_2(n))$ | **Время для базовых операций - Логарифмическое время.** **Гарантирует порядок элементов - в основе лежит красно-черное дерево, которое умеет само себя балансировать.** **Не предоставляет каких-либо параметров для настройки производительности** **Предоставляет дополнительные методы для упорядоченного списка: first(), last(), headSet(), tailSet()**
30	**19.Расскажите про интерфейс List**	Контейнеры List хранит элементы в порядке добавления. Интерфейс List дополняет Collection несколькими методами, обеспечивающими вставку и удаление элементов в середине списка.
31	**20.Как устроен ArrayList, сложность основных операций.**	**ArrayList** – временная сложность основных операций | | Индекс | Поиск | Вставка | Удаление | |---|---|---|---|---| | Метод | get(i) | contains(object) | add(object) | remove(index) | | Среднее | O(1) | O(n) | O(n) | O(n) | | Худшее | O(1) | O(n) | O(n) | O(n) | ArrayList реализован внутри в виде обычного массива. Поэтому при вставке элемента в середину, приходится сначала сдвигать на один все элементы после него, а уже затем в освободившееся место вставлять новый элемент. Механизм автоматического «расширения» массива существует, а вот автоматического «сжатия» нет, можно только явно выполнить «сжатие» командой trimToSize()
33	**21.Что такое Queue?**	Queue - коллекция, предназначенная для хранения элементов в порядке, нужном для их обработки. Очереди обычно, но не обязательно, упорядочивают элементы в FIFO (first-in-first-out) порядке.

34	**22.Что такое Deque? Чем отличается от Queue?**	Deque - двухстороняя очередь, расширяет queue. Он отличается от Queue тем, что можно добавлять и удалять элементы как в хвосте так и в голове. Количество методов удваивается. Пример:
		addFirst(E e);
		addLast(E e);
		Помимо этого реализации интерфейса Deque могут строится по принципу FIFO, либо LIFO.
		Реализации и Deque, и Queue обычно не переопределяют методы equals() и hashCode(), а используются методы класса Object, основанные на сравнении ссылок.
		Рекомендуется использовать вместо устравшего Stack.

35	**23.Приведите пример реализации Deque.**	Linked list, ArrayDeque
36	**24.Какая коллекция**	Queue

	реализует FIFO?	
37	**25.Какая коллекция реализует LIFO?**	Vector, ArrayDeque
38	**26.Оцените количество памяти на хранение одного примитива типа byte в LinkedList?**	Для 32-битных систем каждая ссылка занимает 32 бита (4 байта). Сам объект (заголовок) вложенного класса Node занимает 8 байт. 4 + 4 + 4 + 8 = 20 байт, а т.к. размер каждого объекта в Java кратен 8, соответственно получаем 24 байта. Примитив типа byte занимает 1 байт памяти, но в JCF примитивы упаковываются: объект типа Byte занимает в памяти 16 байт (8 байт на заголовок объекта, 1 байт на поле типа byte и 7 байт для кратности 8). Также напомню, что значения от -128 до 127 кэшируются и для них новые объекты каждый раз не создаются. Таким образом, в x32 JVM 24 байта тратятся на хранение одного элемента в списке и 16 байт - на хранение упакованного объекта типа Byte. Итого 40 байт. Для 64-битной JVM каждая ссылка занимает 64 бита (8 байт), размер заголовка каждого объекта составляет 16 байт (два машинных слова). Вычисления аналогичны: 8 + 8 + 8 + 16 = 40байт и 24 байта. Итого 64 байта.
39	**27.Оцените количество памяти на хранение одного примитива типа byte в ArrayList?**	ArrayList основан на массиве. Каждый элемент массива хранит примитивный тип данных - byte, размер которого 1 байт.
40	**28.Какие существуют реализации Map?**	TreeMap, HashMap, HashTable, LinkedHashMap
41	**29.Как устроена HashMap, сложность основных операций? (Расскажите про принцип корзин)**	**HashMap – временная сложность основных операций** Поиск | Вставка | Удаление Метод: get/contains(key) | put(key, object) | remove(key) Среднее: O(1) | O(1) | O(1) Худшее (до Java 8): O(n) | O(n) | O(n) Худшее (Java 8+): $O(\log_2(n))$ | $O(\log_2(n))$ | $O(\log_2(n))$ HashMap – внутри состоит из корзин и списка элементов, на которые ссылаются корзины. Корзины – массив Элементы(Node) – связанный список, то есть каждый элемент списка имеет указатель на следующий элемент. При добавлении нового элемента, хэш-код ключа определяет корзину для элемента с помощью hashFunction(), который принимает hashCode ключа и возвращает номер корзины. В корзине есть ссылка на связанный список, в который будет положен наш объект. Идет проверка, есть ли элементы в этом списке. Если нету, то корзина получает ссылку нового элемента, если есть, то происходит прохождение по списку элементов и сравнивание

		элементов в списке. Проверяется равенство hashcode. Зная о коллизии, проводится еще сравнивание ключей методом equals. Если оба равны: идет перезапись Если не равен equals: добавляется элемент в конец списка HashMap имеет поле loadFactor. Оно может быть задано через конструктор. По умолчанию - 0.75. Его произведение на количество корзин дает нам необходимое число объектов которое нужно добавить чтобы состоялось удвоение количества корзин. Например если у нас мапка с 16-ю(default) корзинами, а loadFactor равняется 0.75, то расширение произойдет когда мы добавим 16 * 0.75 = 12 объектов. После удвоения все объекты будут перераспределены с учетом нового количества корзин
43	**30.Как устроена TreeMap, сложность основных операций?**	**TreeMap** – временная сложность основных операций | | Поиск | Вставка | Удаление | | Метод | get/contains(key) | put(key, object) | remove(key) | | Среднее | $O(\log_2(n))$ | $O(\log_2(n))$ | $O(\log_2(n))$ | | Худшее | $O(\log_2(n))$ | $O(\log_2(n))$ | $O(\log_2(n))$ | Класс TreeMap<K, V> представляет отображение в виде дерева. Он наследуется от класса AbstractMap и реализует интерфейс NavigableMap, а следовательно, также и интерфейс SortedMap. Поэтому в отличие от коллекции HashMap в TreeMap все объекты автоматически сортируются по возрастанию их ключей.
45	**31.Как работает HashMap при попытке сохранить в него два элемента по ключам с одинаковым hashCode(), но для которых equals() == false?**	По значению hashCode() вычисляется индекс ячейки массива, в список которой этот элемент будет добавлен. Перед добавлением осуществляется проверка на наличие элементов в этой ячейке. Если элементы с таким hashCode() уже присутствует, но их equals() методы не равны, то элемент будет добавлен в конец списка.
46	**32.Что будет, если мы кладем в HashMap ключ, у которого equals и hashCode определены**	Объект скорее всего добавится, но обратно мы не сможем получить его.

	некорректно?	
47	**33.Возможна ли ситуация, когда HashMap выродится в список даже с ключами имеющими разные hashCode()?**	Это возможно в случае, если метод, определяющий номер корзины будет возвращать одинаковые значения.
48	**34.Почему нельзя использовать byte[] в качестве ключа в HashMap?**	Хэш-код массива не зависит от хранимых в нем элементов, а присваивается при создании массива (метод вычисления хэш-кода массива не переопределен и вычисляется по стандартному Object.hashCode() на основании адреса массива). Также у массивов не переопределен equals и выполняется сравнение указателей. Это приводит к тому, что обратиться к сохраненному с ключом-массивом элементу не получится при использовании другого массива такого же размера и с такими же элементами, доступ можно осуществить лишь в одном случае — при использовании той же самой ссылки на массив, что использовалась для сохранения элемента.
49	**35.Будет ли работать HashMap, если все добавляемые ключи будут иметь одинаковый hashCode()?**	Да, будет, но в этом случае HashMap вырождается в связный список и теряет свои преимущества.
50	**36.Какое худшее время работы метода get(key) для ключа, которого нет в HashMap?**	O(N). Худший случай - это поиск ключа в таблице, вырожденной в список, перебор ключей которой занимает линейно пропорциональное время количеству хранимых элементов.
51	**37.Какое худшее время работы метода get(key) для ключа, который есть в HashMap?**	O(N) - линейное

52

38.Начальная ёмкость коллекций

Name	Base Class	Base Interface	AD	AN	Inserted Order?	Sorted Order?	Synch -roniz ed	Random Access	Default Capacity	Description
ArrayList	AbstractList	List	Yes	Yes	Yes	No	No	Yes	10	It supports dynamic arrays that can grow as needed.
LinkedList	Abstract SequentialList	List, Deque, Queue	Yes	Yes	Yes	No	No	Yes	0	It provides a Linked List data structure.
HashSet	AbstractSet	Set	No	Yes	No	No	No	No	16	It creates a collection that uses hash table for storage.
LinkedHashSet	HashSet	-	No	Yes	Yes	No	No	No	16	It creates a Linked List with no duplicate elements.
TreeSet	AbstractSet	Navigable Set	No	No	No	Yes	No	No	16	It creates a collection that uses tree for storage.By default, objects are stored in ascending order.
PriorityQueue	Abstract Queue	Queue	Yes	No	No	Yes	No	No	11	It creates a queue that is prioritized based on queue's comparator.
ArrayDeque	Abstract Collection	Deque	Yes	No	Yes	No	No	No	16	It creates a dynamic array.
EnumSet	AbstractSet	Set	No	No	Yes	No	No			It is specifically for use with elements of enum type.

53

Функциональные интерфейсы

54

1.Что такое функциональный интерфейс?

Это интерфейс, который содержит только 1 абстрактный метод. Интерфейс может включать сколько угодно default (и static) методов и при этом оставаться функциональным, потому что default методы - не абстрактные.

55

2.Для чего нужна аннотация @FunctionalInterface?

Нужна чтобы точно определить интерфейс как функциональный. Она обозначит замысел и не даст определить второй абстрактный метод в интерфейсе.

56

3.Какие встроенные функциональные интерфейсы вы знаете?

Predicate<T> - реализуется функция, получающая на вход экземпляр класса T и возвращающая на выходе значение типа boolean

Consumer<T> - реализуется функция, которая получает на вход экземпляр класса T, производит с ним некоторое действие и ничего не возвращает

Function<T,R> - реализуется функция, получающая на вход экземпляр класса T и возвращающая на выходе экземпляр класса R

Supplier<T> - реализуется функция, ничего не принимающая на вход, но возвращающая на выход результат класса T

UnaryOperator<T> - принимает в качестве параметра объект типа T, выполняет над ними операции и возвращает результат операций в виде объекта типа T

		BinaryOperator<T, T> - реализуется функция, получающая на вход два экземпляра класса T и возвращающая на выходе экземпляр класса T
57	**4.Что такое ссылка на метод?**	Ссылка на статический метод - ContainingClass::staticMethodName Ссылка на нестатический метод конкретного объекта - containingObject::instanceMethodName Ссылка на конструктор - ClassName::new Ссылка на метод - это сокращенный синтаксис выражения лямбда, который выполняет только один метод. Это позволяет нам ссылаться на конструкторы или методы, не выполняя их.
58	**5.Что такое лямбда-выражение? Чем его можно заменить?**	Лямбда-выражение - упрощённая запись анонимного класса, реализующего функциональный интерфейс
	Stream API	
60	**1.Что такое Stream API? Для чего нужны стримы?**	Интерфейс java.util.Stream представляет собой последовательность элементов, над которой можно производить различные операции. Операции над стримами бывают или промежуточными или терминальными. Терминальные операции возвращают результат определенного типа, а промежуточные операции возвращают тот же стрим. Таким образом вы можете строить цепочки из несколько операций над одним и тем же стримом. Его задача - упростить работу с наборами данных, в частности, упростить операции фильтрации, сортировки и другие манипуляции с данными.
61	**2.Почему Stream называют ленивым?**	Методы не будут выполняться пока не будет вызван терминальный метод
62	**3.Какие существуют способы создания стрима?**	Пустой стрим: Stream.empty() Стрим из List: list.stream() Стрим из Map: map.entrySet().stream() Стрим из массива: Arrays.stream(array) Стрим из указанных элементов: Stream.of("1", "2", "3") -Можно получить из BufferedReader при помощи метода lines(), который вернет поток строк из потока символов. -Из директории на диске при помощи методов Files.list() и Files.walk() -Можно получить из строки методом chars(), будет IntStream с символами. -Можно порождать динамически, генерировать при помощи supplier. -Итерированием какой-то функции -Можно получить диапазон чисел в виде стрима range и rangeClosed -Конкатенацией других стримов
63	**4.Как из коллекции создать стрим?**	Collection<String> collection = Arrays.asList("a1", "a2", "a3"); Stream<String> streamFromCollection = collection.stream();

64	**5.Какие промежуточные методы в стримах вы знаете?**	filter(boolean - Predicate) map() flatMap() limit(n) skip(n) concat(Stream s1, Stream s2) peek(someFunction) distinct() sorted()
65	**6.Расскажите про метод peek().**	Предполагается, что map() получает на вход один объект, а возвращает другой. Возможно, того же типа, но другой. peek() - это частный случай map(), который возвращает тот же самый объект, который получил на входе, возможно, с изменённым внутренним состоянием. Конечно, можно использовать для этого map(), но есть нюансы. Во-первых, peek() на одну строчку короче - не нужно писать return, Java и так знает, что нужно возвращать. Во-вторых, вы страхуетесь от ошибок - из peek() невозможно вернуть не тот объект, который пришёл на вход.
66	**7.Расскажите про метод map().**	Метод map() заданным образом преобразует каждый элемент стрима, потом преобразует все объекты в итоговый стрим.
67	**8.Расскажите про метод flatMap().**	flatMap возвращает по стриму для каждого объекта в первоначальном стриме, а затем результирующие потоки объединяются в исходный стрим.
68	**9.Чем отличаются методы map() и flatMap().**	map для каждого объекта в стриме возвращает по 1 объекту, потом преобразует все объекты в итоговый стрим. flatMap возвращает по стриму для каждого объекта в первоначальном стриме, а затем результирующие потоки объединяются в исходный стрим.
69	**10.Расскажите про метод filter()**	фильтрует стрим, возвращая только те элементы, что проходят по условию (Predicate) Проверяет значение на “true” и “false”
70	**11.Расскажите про метод limit()**	limit(n) - возвращает новый поток, ограниченный n-результатами
71	**12.Расскажите про метод skip()**	skip(n) - возвращает новый поток, пропуская первые n элементов
72	**13.Расскажите про метод sorted()**	sorted() - возвращает отсортированный поток
73	**14.Расскажите про метод distinct()**	distinct() - возвращает поток равнозначный исходному, но без дубликатов
74	**15.Какие**	-forEach – принимает consumer, которому будут выведены элементы стрима. -forEachOrdered – как и forEach, но гарантирует порядок.

	терминальные методы в стримах вы знаете?	-count() - подсчет всех значений -max() - возвращает максимальный элемент -min() - возвращает минимальный элемент -findAny() - находится вхождение – сразу возвращает результат -anyMatch() проверяет на наличие совпадения -allMatch() – возвращает boolean -noneMatch() – возвращает boolean -findFirst – возвращает первый элемент из стрима, возвращается OptionalInt -collect – собирает элементы в новое хранилище -reduce – результат применения бинарного оператора к каждой паре элементов стрима, пока не останется один элемент. -toArray - возвращет массив Терминальный метод можно вызвать только один раз. Все оконечные методы возвращают Optional - оболочка ответа (этот специальный тип ввели чтобы не возвращать null)
75	**16.Расскажите про метод collect()**	Stream.collect () является одним из терминальных методов. Это позволяет выполнять изменяемые операции свертывания (переупаковка элементов в некоторые структуры данных и применение некоторой дополнительной логики, объединение их и т. Д.) Преобразует стрим в коллекцию
76	**17.Расскажите про метод reduce()**	позволяет выполнять агрегатные функции и возвращать один результат. - Результат применения бинарного оператора к каждой паре элементов стрима, пока не останется один элемент.
77	**18.Расскажите про класс Collectors и его методы.**	Нужен для того, чтобы упаковывать стримы в коллекции: toList() - преобразует поток в список — List<T> toSet() - преобразует поток в список — Set<T> toMap() - преобразует поток в список — Map<K, V> Используются в методе collect().
78	**19.Расскажите о параллельной обработке в Java 8.**	Чтобы сделать обычный последовательный поток параллельным, надо вызвать у объекта Stream метод parallel. А обратный метод - sequential(). Кроме того, можно также использовать блокирующий метод parallelStream() интерфейса Collection для создания параллельного потока из коллекции. В то же время если рабочая машина не является многоядерной, то поток будет выполняться как последовательный. Работает на фреймворке fork/join.
79	**20.Что такое IntStream и DoubleStream?**	В Java 8 создание Stream-ов примитивов напрямую невозможно, из-за дженериков. Но разработчики сделали 3 Stream-а примитивов : IntStream, LongStream, DoubleStream. Работает быстрее, чем стрим с классами-обертками. Поддерживают дополнительные терминальный методы sum(), average(), mapToObj()
	Java 8	
81	**1.Какие нововведения появились в java 8?**	1. Полноценная поддержка лямбда-вражений 2. Ссылки на методы :: 3. Функциональные интерфейсы 4. default методы в интефейсах 5. Потоки для работы с коллекциями 6. Новое api для работы с датами 7. Nashorn движок JavaScript, разрабатываемый полностью на Java компанией Oracle.

		8. Кодировщик/декодировщик. 9. Новые методы для Map - PutIfAbsent(), ComputeIfAbsent()\ComputeIfPresent(), Remove(), GetOrDefault(), Merge() 10. Metaspace пришла на замену PermGen
82	**2.Какие новые классы для работы с датами появились в java 8?**	LocalDate , LocalTime, LocalDateTime, ZonedDateTime, Period, Duration
83	**3.Расскажите про класс Optional**	Optional - новый класс в пакете java.util, является контейнером (оберткой) для значений которая также может безопасно содержать null. Благодаря опциональным типам можно забыть про проверки на null и NullPointerException.
84	**4.Что такое Nashorn?**	В Java 8, Nashorn, представлен значительно улучшенный движок javascript для замены существующего Rhino. Nashorn обеспечивает в 2-10 раз лучшую производительность, так как он напрямую компилирует код в памяти и передает байт-код в JVM. Nashorn использует функцию динамического вызова, представленную в Java 7, для повышения производительности. * Nashorn — немецкое слово (Носорог)
85	**5.Что такое jjs?**	Инструмент командной строки для выполнения JavaScript-кодов на консоли.
86	**6.Какой класс появился в Java 8 для кодирования/декодирования данных?**	public static class Base64.Encoder /public static class Base64.Decoder
87	**7.Как создать Base64 кодировщик и декодировщик?**	Используя метод getDecoder() класса Base64 он возвращает декодировщик Base64.Decoder, который декодирует данные с помощью схемы кодирования base64.

88	**8.Какие доп. методы для работы с ассоциативными массивами (maps) появились в Java 8?**	**putIfAbsent**() добавляет пару «ключ-значение», только если ключ отсутствовал:
		map.putIfAbsent("a", "Aa");
		forEach() принимает функцию, которая производит операцию над каждым элементом:
		map.forEach((k, v) -> System.out.println(v));
		compute() создаёт или обновляет текущее значение на полученное в результате вычисления (возможно использовать ключ и текущее значение):
		map.compute("a", (k, v) -> String.valueOf(k).concat(v)); //["a", "aAa"]
		computeIfPresent() если ключ существует, обновляет текущее значение на полученное в результате вычисления (возможно использовать ключ и текущее значение):
		map.computeIfPresent("a", (k, v) -> k.concat(v));
		computeIfAbsent() если ключ отсутствует, создаёт его со значением, которое

		вычисляется (возможно использовать ключ):
		map.computeIfAbsent("a", k -> "A".concat(k)); //["a","Aa"]
		getOrDefault() в случае отсутствия ключа, возвращает переданное значение по-умолчанию:
		map.getOrDefault("a", "not found");
		merge() принимает ключ, значение и функцию, которая объединяет передаваемое и текущее значения. Если под заданным ключем значение отсутствует, то записывает туда передаваемое значение.
		
		- map.remove(key, value) - Если такое ключ-значение есть в map, то удаляем

89	**9.Что такое LocalDateTime?**	LocalDateTime объединяет вместе LocaleDate и LocalTime, содержит дату и время в календарной системе ISO-8601 без привязки к часовому поясу. Время хранится с точностью до наносекунды. Содержит множество удобных методов, таких как plusMinutes, plusHours, isAfter, toSecondOfDay и т.д.
90	**10.Что такое ZonedDateTime?**	java.time.ZonedDateTime — аналог java.util.Calendar, класс с самым полным объемом информации о временном контексте в календарной системе ISO-8601. Включает объект ZoneId - временную зону(в ZoneId 599 зон), поэтому все операции с временными сдвигами этот класс проводит с её учётом.