# КОЛЛЕКЦИИ

# *Что такое дженерики.*

**это механизм, который позволяет создавать классы, интерфейсы и методы с параметрами типа.**

# *Какую проблему они решают.*

**Использование дженериков позволяет повысить безопасность типов и уменьшить количество ошибок, связанных с привидением типов, а также сделать код более читаемым и повторно используемым**

# *Что было до дженериков.*

**использовали необобщенные типы (non-generic types)**

# *Что можно типизировать.*

**Классы, Интерфейсы, методы, конструкторы, вложенные классы используются в коллекциях**

# *Что такое стирание и сырые типы (raw type - необработанный тип).*

**Стирание типов (Type Erasure) — это процесс, при котором информация о типах дженериков удаляется компилятором на этапе компиляции**

**Необработанные типы (Raw Types) — это использование дженериковых типов без указания параметров типа. Необходимо приведение**

# *К чему приводит использование raw type.*

**При использовании сырых типов приведение типов необходимо и может привести к ClassCastException**

# *Какой механизм обеспечивает обратную совместимость сырых типов и дженериков.*

**Стирание типов (Type Erasure)**

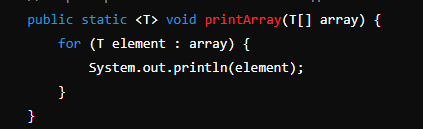
# *Если поле типизировано дженериком как в байт коде будет представлен этот тип.*

**Все типовые параметры заменяются их границами или Object если границы не заданы**

**Дополнительная информация о дженериках сохраняется в метаданных для рефлексии**

# *Как параметризовать статический метод.*

**<T> перед возвращаемым типом метода printArray указывает, что метод будет использовать тип T, определенный при вызове метода.**



# *Что такое даймонд оператор.*

**Это означает, что Java автоматически подставляет тот тип параметра, который должен быть в типе переменной «<>»**

# *Чему эквивалентно <?>.*

представляет собой **неограниченный типовой параметр**

# Можно ли объявить так: class Animal<?> {}

**Нет нельзя. В классах мы должны четко указать параметр**

# *Принцип PECS.*

**Producer Extends**: Если ваш обобщенный тип предназначен только для чтения данных (т.е. вы будете производить данные из этой структуры), используйте wildcard с extends (ограничение сверху). Это позволяет вашему обобщенному типу работать с типом и его подтипами.

**Consumer Super**: Если ваш обобщенный тип предназначен только для записи данных (т.е. вы будете потреблять данные в эту структуру), используйте wildcard с super (ограничение снизу). Это позволяет вашему обобщенному типу работать с типом и его суперклассами

# WILDCARD

***<?> используется только для wildcard-типов в контексте методов, коллекций и других обобщенных структур данных***

# *Почему последняя строчка не скомпилируется?*

**List<ArrayList> arrayLists = new ArrayList<ArrayList>();**

**ArrayList<List> arrayList = new ArrayList<ArrayList>();**

***ошибка компиляции происходит из-за несовместимости обобщённых типов,***

Коллекции.

# Что такое коллекция.

Структура данных, набор каких либо объектов.

# Какие есть типы коллекций? Как они характеризуются?

# Отличие коллекции от массива.

# Иерархия коллекций.

# 

# Внутреннее устройство коллекций.

# Какие структуры данных вы знаете.

# Как задается порядок следования объектов в коллекции, как отсортировать коллекцию?

# Назовите основные реализации List, Set, Queue, Map.

# Может ли коллекция хранить примитивы? Как можно реализовать хранение примитивов?

List

# *Отличие List от Set.*

* **List это упорядоченная последовательность элементов, тогда как Set – это отдельный список элементов, который не упорядочен.**
* **List допускает дублирование, а Set не допускает дублирование элементов.**
* **List разрешает любое количество нулевых значений в своей коллекции, а Set разрешает только одно нулевое значение в своей коллекции.**
* **List может быть вставлен как в прямом, так и в обратном направлении с помощью Listiterator, тогда как Set можно просматривать только в прямом направлении с помощью итератора.**

# *Как устроен LinkedList.*

**Это цепочка элементов. Элементы хранят ссалки на предыдущий элемент и следующий элемент**

# *Отличие ArrayList и LinkedList.*

**ArrayList это список на реализованный на основе массива**

**LinkedList – это список, основанный на обьектах с ссылками между ними**

# *Когда лучше использовать ArrayList, а когда LinkedList.*

**ArrayList - использование метода get**

**LinkedList – при использовании методов add, remove**

# *Скорость вставки элемента в начало середину и конец у ArrayList и LinkedList.*

**LinkidList – в начало О(1)**

**Середина O(n)**

**Конц О(1)**

**ArrayList - в начало О(n)**

**Середина O(n)**

**Конц О(1)**

# *Как работает метод contains в ArrayList и LinkedList.*

ArrayList представляет собой динамический массив. Метод contains в ArrayList работает следующим образом:

1. **Линейный поиск**: Метод contains выполняет линейный поиск (простое последовательное сканирование) через элементы массива.
2. **Использование метода equals**: Для каждого элемента вызывается метод equals для сравнения с искомым элементом.

LinkedList представляет собой двусвязный список. Метод contains в LinkedList работает следующим образом:

1. **Линейный поиск**: Метод contains также выполняет линейный поиск через элементы списка, начиная с головы списка и переходя по ссылкам next.
2. **Использование метода equals**: Для каждого элемента вызывается метод equals для сравнения с искомым элементом.

# *Отличие двусвязного и односвязного списка.*

**Узел не содержит указатель на предыдущий узел / содержит**

**Можно перемещаться только в обном направлении / в обеих**

**Хранит меньше памяти / больше памяти**

**Вставка быстрая только в начале / вставка эфективна в конце и вначале**

***Set***

# *3 реализации Set. Какая упорядоченность в какой и почему.*

**HashSet – по хэш-коду.**

**TreeSet – упорядоченный отсортированный список**

**LinkedHashSet – последовательность добавления объектов**

# *Как работает HashSet. ()*

**Реализован как хэш таблица где элементы хранятся в виде ключей, а значения не используются**

* **При добавлении элемента, HashSet рассчитывает его хэш-код и добавляет в таблицу с соответствующим индексом.**
* **Если в таблице уже есть элемент с таким же хэш-кодом , то выполняется проверка на равенство**
* **Если элементы равны, то новый элемент не добавляется в коллекцию, иначе он добавляется в таблицу.**

# Внутреннее устройство HashSet.

# Что кладется на место значения в HashSet.

Заглушка, некая константа.

# Почему в HashSet вместо value не null а new Object.

# Null в TreeSet.

Не хранит

# Как работает метод contains в HashSet.

# Назовите различия между Set и List?

Map

# *Что такое Map.*

**Элементами map являются пары ключ-значения.**

# *Что должно быть уникальным в Map.*

**Ключи в map уникален**

# *Почему Map не входит в Collection.*

**Потому что map совокупность ключ – значение, Соllection – представляет совокупность неких элементов.**

# *Строение HashMap?*

**В основе HashMap лежит массив. Элементами массива являются структуры linkidList. Данные структура LinkedList и заполняются элементами, который мы добавляем в HashMap.**

**HashMap состоит из «корзин» (buckets). С технической точки зрения «корзины» – это элементы массива, которые хранят ссылки на списки элементов. При добавлении новой пары «ключ-значение» вычисляется хеш-код ключа, на основании которого вычисляется номер корзины (номер ячейки массива), в которую попадет новый элемент. Если корзина пустая, то в нее сохраняется ссылка на вновь добавляемый элемент, если там уже есть элемент, то происходит последовательный переход по ссылкам между элементами в цепочке в поисках последнего элемента, от которого и ставится ссылка на вновь добавленный элемент. Если в списке был найден элемент с таким же ключом, то он заменяется**.

# *Что внутри статического класса Node?*

**Finali int hash - hashCode**

**finali K key, - ключ**

**V value - значение**

**Node<K, V> next – ссылка на следующий элемент**

# *Как работает HashMap.*

**Работает по принципу хэширование при помощи hasCode. Создается массив с 16 ячейками, каждый элемен массива (корзина) содержит LinkidList.**

# *Как расширяется HashMap.*

# *HashMap в Java расширяется автоматически, когда его размер достигает порогового значения. По умолчанию пороговое значение равно 0,75 от начальной емкости.*

# *Когда количество элементов в HashMap достигает порогового значения, HashMap увеличивает свою емкость в два раза.*

# *После увеличения емкости все элементы из старой HashMap копируются в новую HashMap.*

# *При этом каждый элемент пересчитывает свой индекс, чтобы быть размещенным в новом массиве элементов HashMap.*

# *Может ли null быть ключём в HashMap.*

**Ключи элементов должны быть уникальны, они могут быть null**

**Значения могут повторится, они могут быть null**

# *Какой хэш-код у null в HashMap.*

# Хэш код null равен 0.

# *Расскажите подробно, как работает метод put.*

**Сначала проверяется на null -> если null то помещает на 0 индекс. Если не нал вычисляет HashCode для ключа, потом при помощи алгоритма в HashMap находится индекс куда будет помещен данный элемент (если хэшкоды совпадают, то и индекс совпадает. Дальше на индекс добавляется объект Node(хэш код, кей, значение, некс – это ссылка на следующий элемен, если ее нет, то она равна null**

**Если индексы совпадают то идет проверка хэш кода если хэшкоды разные, то объект добавляется в конец linkidList**

**Если хэшкоды совпали то и индекс тоже равен. И в этом проверяются на равенство по equals если equals не равны то объект добавляется в лист. Но если и хэш и equals равны происходит перезапись.**

# *Как работает метод get и remove в HashMap.*

**Работает по ключц, точно также как и put но возвращает значение или удаляет**

# *Красно-черное дерево*

**Это двоичное дерево есть вершина с прова юольшие значения с лева меньшие, и оно является самобалансированной. Т.е система сама перестроит ключи в двоичное дерево.**

**С помощью данного подхода методы по поиску, добавлению обрабатывается быстро O(logn) но не быстрее чем в HashMap O(n)**

**TreeMap – элементами явлюяются пара ключ/значение, элементы хранятся в отсортированном по возрастанию порядке. Уникальность ключей сдесь должна поддерживаться (объекты перезаписываются)**

Чего не гарантирует и что гарантирует красно-черное дерево?

# *Что такое бинарное дерево поиска (BST)*

# структура данных для работы с упорядоченными множествами

# Что такое коллизия. Что происходит при коллизии. Как будет разрешаться коллизия.

Коллизия по хэшкоду.

# *Как перебрать все ключи, значения, пары Map учитывая, что Map - это не Iterable?*

**Для перебора всех ключей в Map используйте метод keySet(), который возвращает Set всех ключей. Затем можно использовать цикл for-each или итератор для обхода ключей.**

**Для перебора всех значений в Map используйте метод values(), который возвращает Collection всех значений. Затем можно использовать цикл for-each или итератор для обхода значений.**

**Для перебора всех пар (ключ-значение) в Map используйте метод entrySet(), который возвращает Set элементов типа Map.Entry. Каждое Map.Entry представляет собой пару ключ-значение. Вы можете итерировать по этим парам с помощью итератора или цикла for-each.**

**Использование метода forEach**: В Java 8 и выше вы можете использовать метод forEach для перебора пар (ключ-значение), что позволяет использовать лямбда-выражения для более краткого и выразительного кода.

Queue

# *Что такое Queue.*

**Это коллекция хранящая последовательность элементов. Добавляются эти элементы в конец очереди. Используется правило FIFO ferst in ferst out**

# *В чём разница между Queue и Deque и Stack.*

**По использованию правил**

**Queue – правило FIFO**

**Deque – правило FIFO и LIFO**

**Stack – правило LIFO**

# *Что такое Dequeue?*

**Двунаправленная очередь. В такой очереди элементы могут использоваться с обоих концов. правило FIFO и LIFO**

Iterator.

# *Что такое Iterator?*

**это интерфейс, предоставляющий методы для последовательного доступа к элементам коллекции без необходимости знать внутреннюю структуру этой коллекции**

# *В каких случаях нужно использовать iterator. почему*.

Обход коллекций

Удаление элементов во время итерации

Когда нужно избежать ConcurrentModificationException

Когда нужно более гибко управлять процессом итерации

 **Универсальность**:

* Итераторы работают с любыми типами коллекций, что делает код более гибким и независимым от конкретной реализации коллекции.

 **Безопасность**:

* Итераторы обеспечивают fail-fast поведение, предупреждая о попытках небезопасных изменений коллекции во время обхода, что помогает предотвратить ошибки.

 **Чистота и читабельность кода**:

* Итераторы предоставляют стандартный способ обхода коллекций, который делает код более чистым и понятным.

 **Удаление элементов**:

* Метод remove() в итераторах позволяет безопасно удалять элементы во время итерации, чего нельзя достичь при использовании цикла for-each.

 **Избежание ConcurrentModificationException**:

* Итераторы автоматически обнаруживают изменения коллекции во время итерации и выбрасывают ConcurrentModificationException, что помогает выявить ошибки на ранней стадии.

# *Зачем в итераторе метод remove.*

**Метод remove() в итераторах позволяет безопасно удалять элементы во время итерации, чего нельзя достичь при использовании цикла for-each.**

# *В чём разница между Iterable и Iterator.*

Iterable представляет объект, который можно итерировать и предоставляет итератор.

Iterator предоставляет методы для фактического перебора элементов.

# *Как между собой связаны Iterable, Iterator и «forEach»?*

 **Iterable** предоставляет механизм для обхода коллекции, предоставляя итератор. Это позволяет использовать как цикл for-each, так и метод forEach.

 **Iterator** используется для последовательного доступа к элементам коллекции, и в некоторых случаях для удаления элементов во время обхода.

 **Метод forEach** предоставляет более современный и функциональный способ итерации по элементам коллекции, который не требует явного использования итератора и позволяет легко интегрировать лямбда-выражения и ссылки на методы.

# *listIterator - что это, в чём отличие от обычного.*

**специальный интерфейс, расширяющий функциональность обычного Iterator который поддерживает двусторонний обход**

**поддерживает итерацию в обоих направлениях**

**меет дополнительные методы add(E e) для добавления элементов и set(E e) для замены текущего элемента.**

**предоставляет методы для получения индексов nextIndex() и previousIndex().**

# *Что произойдет при вызове Iterator.next() без предварительного вызова Iterator.hasNext()?*

**выбросит исключение NoSuchElementException. Это исключение сигнализирует о том, что нет больше элементов для итерации.**

# *Сколько элементов будет пропущено, если Iterator.next() будет вызван после 10-ти вызовов Iterator.hasNext()?*

**Не пропустит не одного вызовется следующий элемент если он есть.**