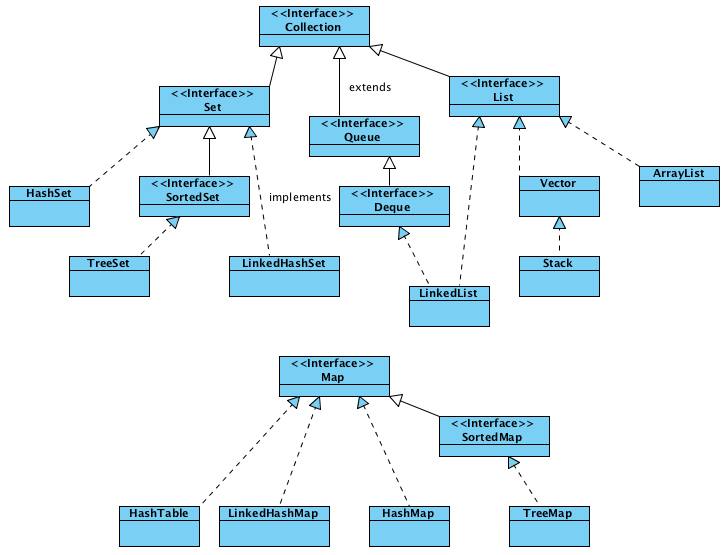
1. Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью оператора enum, после которого идет название перечисления. Затем идет список элементов перечисления через запятую. Перечисление фактически представляет новый тип, поэтому мы можем определить переменную данного типа и использовать ее. Перечисление — это класс. Объявляя enum, мы неявно создаем класс производный от java.lang.Enum. следовательно элементы перечисления — это элементы класса enum.Мы можете определить методы abstract в объявлении enum, если и только если все значения перечисления имеют собственные тела классов с реализациями этих методов (то есть, нет конкретного значения перечисления может отсутствовать реализация). Enum может наследовать интерфейсы. Поскольку Enum тип схож с классом и интерфейсом, он может наследовать интерфейс.Enum может содержать static методы примером может послужить метод статический метод values, который возвращает массив, содержащий значения объекта в порядке их объявления данный метод добавляется компилятором автоматически. Типы enum также могут использоваться в выражениях switch.
2. Поскольку типы перечислений гарантируют, что в JVM существует только один экземпляр констант, мы можем смело использовать оператор «==» для сравнения двух переменных. Сравнивать мы может как сими переменные, так и некоторые поля, например, имя переменной enum всегда хранится в виде String и его можно сравнить с другой строкой, например, tour.getTourType().name().equals(str). (плохо ответил мало инф)
3. исключительная ситуация — Совокупность определенных условий, возникновение которых приводит к нарушению предусмотренной последовательности выполнения в программе. Обработка ошибок в java производится при помощи ключевых слов try – определяет блок кода, в котором может произойти исключение; catch – определяет блок кода, в котором происходит обработка исключения; finally – определяет блок кода, который является необязательным, но при его наличии выполняется в любом случае независимо от результатов выполнения блока try. throw – используется для возбуждения исключения; throws – используется в сигнатуре методов для предупреждения, о том, что метод может выбросить исключение.
4. Что такое исключение описал сверху. Все исключения в Java являются объектами. Поэтому они могут порождаться не только автоматически при возникновении исключительной ситуации, но и создаваться самим разработчиком. Исключение может выбрасываться вручную, например, тогда, когда наша логика программы не позволяет делать данное действие, а для вм все нормально, например, наша программа должна в графу цена записать стоимость покупки, а вместо этого она записала ее в графу телефонный номер покупателя вм ошибки не увидит, а вот для нас это ошибка в следствие чего нужно остановить работу программы и сообщить об ошибке. Исключения делятся на несколько классов, но все они имеют общего предка — класс Throwable. Его потомками являются подклассы Exception и Error. Так же в java есть такие типы исключений как Checked исключения, это те, которые должны обрабатываться блоком catch или описываться в сигнатуре метода. Unchecked могут не обрабатываться и не быть описанными. Unchecked исключения в Java — наследованные от RuntimeException, checked — от Exception (не включая unchecked). Checked исключения отличаются от Unchecked исключения в Java, тем что: Наличие\обработка Checked исключения проверяются на этапе компиляции. Наличие\обработка Unchecked исключения происходит на этапе выполнения. Исключения, выбрасываемые виртуальной машиной это классические исключения отслеживание, которых было написано разработчиками JVM.
5. Исключения делятся на несколько классов, но все они имеют общего предка — класс Throwable. Его потомками являются подклассы Exception и Error. В java есть такие типы исключений как Checked исключения, это те, которые должны обрабатываться блоком catch или описываться в сигнатуре метода. Unchecked могут не обрабатываться и не быть описанными. Unchecked исключения в Java — наследованные от RuntimeException, checked — от Exception (не включая unchecked). Checked исключения отличаются от Unchecked исключения в Java, тем что: Наличие\обработка Checked исключения проверяются на этапе компиляции. Наличие\обработка Unchecked исключения происходит на этапе выполнения.
6. Данный оператор состоит из нескольких частей: try – определяет блок кода, в котором может произойти исключение; catch – определяет блок кода, в котором происходит обработка исключения; finally – определяет блок кода, который является необязательным, но при его наличии выполняется в любом случае независимо от результатов выполнения блока try. Данный оператор используется, когда метод, который мы вызываем может вызвать исключение и в таком случае работа программы не будет остановлена о ошибка будет обработана и работа продолжится. Catch в блоке try может быть сколько угодно но если одно из них сработает то остальные уже нет (возможно). Блоки наиболее специализированных классов исключений должны идти первыми, поскольку ни один подкласс не будет достигнут, если поставить его после суперкласса. Операторы try можно вкладывать друг в друга. Если у оператора try низкого уровня нет раздела catch, соответствующего возбужденному исключению, стек будет развернут на одну ступень выше, и в поисках подходящего обработчика будут проверены разделы catch внешнего оператора try. «try c ресурсами» это оператор try, в котором объявляются один или несколько ресурсов. Ресурс — это объект, который должен быть закрыт после того, как программа закончит с ним работу. «try c ресурсами» берет всю работу по закрытию ресурсов на себя ресурсы будут закрываться снизу-вверх автоматически после завершения работы блока try. При использовании оператора try с ресурсами, совершенно не обязательно использовать блоки catch и finally, они являются опциональными.
7. Блоки наиболее специализированных классов исключений должны идти первыми, поскольку ни один подкласс не будет достигнут, если поставить его после суперкласса. Если возникает необходимость снова сгенерировать исключения из блока, который обрабатывает исключения, можно сделать это путем вызова throw без указания исключения. В результате текущее исключение будет передано во внешнюю последовательность try/catch обработки исключений. Причиной для этого может послужить желание обрабатывать исключения несколькими обработчиками. Например, один обработчик может заниматься одним аспектом исключения, а второй обработчик — другим. Исключение может быть снова сгенерировано или изнутри блока catch, или из функции, вызванной в этом блоке. Когда повторно генерируется исключение, оно не будет перехвачено той же самой инструкцией catch. Оно будет распространяться до следующей внешней инструкции catch.
8. Finally будет вызываться после выполнения блоков кода try или catch. Единственные времена finally будут называться: 1)Если вы вызываете System.exit(); 2)Если JVM падает в первую очередь; 3)Если JVM достигает бесконечного цикла (или некоторого другого непрерываемого, не заканчивающегося оператора) в блоке try catch; 4)Если ОС принудительно завершает процесс JVM; например, "убить -9" в UNIX. 5) Если хост-система умирает; например, сбой питания, аппаратная ошибка, паника ОС и так далее. 6) Если блок finally будет выполняться потоком демона и все остальные потоки, не являющиеся демонами, завершат работу до вызова метода finally. Да finally может выбрасывать исключение (например, если мы в блоке finally вызываем метод который вызовет исключительную ситуацию). Finally срабатывает только 1 раз(не уверен не нашел)
9. Вручную исключения выбрасываются при помощи команды throw – используемой для возбуждения исключения; за раз мы можем выбрасывать только 1 исключение (в throws можно написать несколько, но выкинет только 1). Если во время выполнения конструктора произошло исключение, то новый объект сразу же подходит для сбора мусора (хотя его, возможно, не собирают в течение некоторого времени, конечно). Возможно, что "полуконструированный" объект будет придерживаться, хотя он сделал себя видимым ранее в конструкторе (например, назначив статическое поле или добавив себя в коллекцию). В результате чего мы можем получить не управляемый объект, который будет просто висеть и занимать память. Throw - бросить исключение самому в то время как throws - предупредить что данный метод может бросать такое-то исключение и его надо обработать при вызове метода.
10. Чтобы создать свой класс исключений, надо унаследовать его от класса Exception. В конструкторе нашего класса нужно отправлять в конструктор класса Exception сообщение об ошибке: super(message). Для генерации исключения в методе выбрасывается исключение с помощью оператора throw: throw new Сlass name. В последствии данные классы можно обрабатывать в блоке try catch. Пример: class FactorialException extends Exception{ private int number; public int getNumber(){return number;} public FactorialException(String message, int num){ super(message); number=num; } } Ответ на первую часть вопроса не нашел.
11. Базовым классом для всех исключений является класс Throwable. От него уже наследуются два класса: Error и Exception. Все остальные классы являются производными от этих двух классов. Класс Error описывает внутренние ошибки в исполняющей среде Java. Программист имеет очень ограниченные возможности для обработки подобных ошибок. Исключения наследуются от класса Exception Методы: getMessage() возвращает сообщение об исключении Метод getStackTrace() возвращает массив, содержащий трассировку стека исключения Метод printStackTrace() отображает трассировку стека printStackTrace() printStackTrace(PrintStream s) printStackTrace(PrintWriter s) Выдача в стандартный или указанный поток полной информации о точке возникновения исключения. toString() Краткое сообщение о исключении
12. Java Collection Framework — иерархия интерфейсов и их реализаций, которая является частью JDK и позволяет разработчику пользоваться большим количеством структур данных из «коробки». На вершине иерархии в Java Collection Framework располагаются 2 интерфейса: Collection и Map. Эти интерфейсы разделяют все коллекции, входящие во фреймворк на две части по типу хранения данных: простые последовательные наборы элементов и наборы пар «ключ — значение» (словари). В Collection входят set, list, queue.  List — это упорядоченный список. Объекты хранятся в порядке их добавления в список. Доступ к элементам списка осуществляется по индексу. Set — множество неповторяющихся объектов. В коллекции этого типа разрешено наличие только одной ссылки типа null. Queue — коллекция, предназначенная для хранения элементов в порядке, нужном для их обработки. В дополнение к базовым операциям интерфейса Collection, очередь предоставляет дополнительные операции вставки, получения и контроля. Очереди обычно, но не обязательно, упорядочивают элементы в FIFO (first-in-first-out, «первым вошел — первым вышел») порядке. Map используется для отображения каждого элемента из одного множества объектов (ключей) на другое (значений). При этом, каждому элементу из множества ключей ставится в соответствие множество значений. В то же время одному элементу из множества значений может соответствовать 1, 2 и более элементов из множества ключей.
13. Что входит в set видно на картинке выше. Интерфейс Set описывает множество. Элементы множества не упорядочены, множество не может содержать двух одинаковых элементов. Интерфейс Set унаследован от интерфейса Collection, но никаких новых методов не добавляет. Изменяется только смысл метода add(Object item) — он не станет добавлять объект item, если он уже присутствует во множестве.

|  |  |  |
| --- | --- | --- |
| Set | HashSet | Множество |
| TreeSet | Множество |
| SortedSet | Отсортированное множество |

Как видно все разновидности set это множества содержащие только уникальные элементы. SortedSet наследует Set Реализации этого интерфейса, помимо того, что следят за уникальностью хранимых объектов, поддерживают их в порядке возрастания. Отношение порядка между объектами может быть определено, как с помощью метода compareTo интерфейса Comparable, так и при помощи специального класса-компаратора, наследующего интерфейс Comparator<T>.

Реализации: TreeSet<E> — коллекция, которая хранит свои элементы в виде упорядоченного по значениям дерева. TreeSet инкапсулирует в себе TreeMap, который в свою очередь использует сбалансированное бинарное красно-черное дерево для хранения элементов. TreeSet хорош тем, что для операций add, remove и contains потребуется гарантированное время log(n).

Класс HashSet реализует интерфейс Set, основан на хэш-таблице, а также поддерживается с помощью экземпляра HashMap. В HashSet элементы не упорядочены, нет никаких гарантий, что элементы будут в том же порядке спустя какое-то время. Операции добавления, удаления и поиска будут выполняться за константное время при условии, что хэш-функция правильно распределяет элементы по «корзинам» более углублённая информация(про время доступа до элементов находится у меня в голове если забуду в интернете а описывать про каждый немного долго).

1. Как можно увидеть в приведенной ниже таблице:

|  |  |  |
| --- | --- | --- |
| List | ArrayList | Список |
| LinkedList | Список |
| Vector | Вектор |
| Stack | Стек |

разновидности List подразделяются на список вектор и стек .

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. Зато в нем быстро реализованы взятие и изменение элемента – операции get, set, так как в них мы просто обращаемся к соответствующему элементу массива.

LinkedList реализован внутри по-другому. Он реализован в виде связного списка: набора отдельных элементов, каждый из которых хранит ссылку на следующий и предыдущий элементы. Чтобы вставить элемент в середину такого списка, достаточно поменять ссылки его будущих соседей. А вот чтобы получить элемент с номером 130, нужно пройтись последовательно по всем объектам от 0 до 130. Другими словами, операции set и get тут реализованы очень медленно.

Vector не сильно отличается от обычного массива по факту это тот же массив только динамического размера и может содержать значения разных типов. Каждый вектор пытается оптимизировать управление хранилищем, поддерживая a capacityи a capacityIncrement. Значение capacity всегда по крайней мере равно размеру вектора; обычно это больше, потому что, когда компоненты добавляются в вектор, память вектора увеличивается кусками до размера capacityIncrement. Приложение может увеличить емкость вектора перед вставкой большого количества компонентов; это уменьшает количество постепенного перераспределения. Очень важный момент в том, что vector потока безопасен.

Коллекции, созданные для того чтобы хранить элементы для дальнейшей обработки. Кроме базовых операций интерфейса Collection, очереди поддерживают дополнительные операции добавления, удаления и проверки состояния элемента. Обычно, но не обязательно очереди работают по принципу FIFO — первым пришел, первым ушел. Стэк — почти как очередь, но работает по принципу LIFO — последним пришел, первым ушел. Независимо от порядка добавления/удаления, голова очереди — это элемент, который будет удален при вызове методов remove() или poll(). Stack как и vector потока безопасен.

1. Как можно увидеть в приведенной ниже таблице:

|  |  |  |
| --- | --- | --- |
| Map | HashMap | Карта/Словарь |
| TreeMap | Карта/Словарь |
| SortedMap | Отсортированный словарь |
| Hashtable | Хеш-таблица |

Разновидности мэп подразделяются на словари, хэш-таблицу и отсортированный словарь.

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. Данная реализация не дает гарантий относительно порядка элементов с течением времени. Разрешение коллизий осуществляется с помощью метода цепочек.

Класс TreeMap расширяет класс AbstractMap и реализует интерфейс NavigatebleMap. Он создает коллекцию, которая для хранения элементов применяет дерево. Объекты сохраняются в отсортированном порядке по возрастанию. Время доступа и извлечения элементов достаточно мало, что делает класс TreeMap блестящим выбором для хранения больших объемов отсортированной информации, которая должна быть быстро найдена.

TreeMap основан на Красно-Черном дереве, вследствие чего TreeMap сортирует элементы по ключу в естественном порядке или на основе заданного вами компаратора.

HashMap работает строго быстрее TreeMap.

TreeMap реализован на красно-черном дереве, время добавления/поиска/удаления элемента — O(log N), где N — число элементов в TreeMap на данный момент.

У HashMap время доступа к отдельному элементу — O(1) при условии, что хэш-функция (Object.hashCode()) определена нормально (что является правдой в случае Integer).

Общая рекомендация — если не нужна упорядоченность, использовать HashMap. Исключение — ситуация с вещественными числами, которые в качестве ключей почти всегда очень плохи. Для них нужно использовать TreeMap, предварительно поставив ему компаратор, который сравнивает вещественные числа так, как это нужно в данной задаче.

Интерфейс SortedMap расширяет Map и создает отображение, в котором все элементы отсортированы в порядке возрастания их ключей. Все ключи в SortedMap должны реализовывать comparable интерфейс.

Hashtable:

этот класс реализует хеш-таблицу, которая отображает ключи на значения. Любой ненулевой объект может использоваться как ключ или как значение.

Чтобы успешно сохранять и извлекать объекты из хеш-таблицы, объекты, используемые в качестве ключей, должны реализовывать метод hashCode и метод equals.

Это похоже на HashMap, но синхронизируется.

Хэш-таблица хранит пару ключ / значение в хэш-таблице.

В Hashtable мы указываем объект, который используется в качестве ключа, и значение, которое мы хотим связать с этим ключом. Затем ключ хэшируется, и полученный хэш-код используется в качестве индекса, по которому значение сохраняется в таблице.

Если в map положить 2 значения с одинаковым ключом, то последнее значение перезапишет предыдущее.

1. В Java, каждая пара хранится в поле Map называемом Map.Entry. Map.entrySet() возвращает набор ключ-значений. Map.Entry изменяется при каждом изменение набора элементов в map.
2. Итератор — объект, позволяющий перебирать элементы коллекции. Например, foreach реализован с использованием итератора. Одним из ключевых методов интерфейса Collection является метод Iterator<E> iterator(). Он возвращает итератор — то есть объект, реализующий интерфейс Iterator. Пример использования Iterator: Iterator<String> iter = states.iterator(); while(iter.hasNext()){ System.out.println(iter.next()); } Основные отличия Iterator от ListIterator: Iterator может использоваться для перебора элементов Set, List и Map. В отличие от него, ListIterator может быть использован только для перебора элементов коллекции List. Iterator позволяет перебирать элементы только в одном направлении, при помощи метода next(). Тогда как ListIterator позволяет перебирать список в обоих направлениях, при помощи методов next() и previous(). При помощи ListIterator вы можете модифицировать список, добавляя/удаляя элементы с помощью методов add() и remove(). Iterator не поддерживает данного функционала. A насчет невалидности. Зависит от контейнера. например если он vector, после изменения контейнера все итераторы могут быть аннулированы. Однако, если это a list, итераторы, не имеющие отношения к модифицированному месту, остаются в силе. Итераторы вектора не действительны при добавлении перед ними элемента. Для deque любое добавление элемента не важно в начало или конец аннулирует итератор. list имеют важное свойство, что вставка не делают недействительными итераторы для перечисления элементов, и что даже удаление делает недействительными только итераторы, указывающие на удаляемые элементы. Map имеет важное свойство, что вставка нового элемента в элемент Map не делает недействительными итераторы, указывающие на существующие элементы. Стирание элемента с карты также не отменяет никаких итераторов, за исключением, конечно, для итераторов, которые фактически указывают на стираемый элемент. (то же самое для set, multiset и multimap)
3. Add()-добавление элемента в коллекцию.

Remove()-удаление элемента из коллекции.

sort ()-отсортировать коллекцию

min() и max()-получить минимальный/максимальный элемент коллекции.

revesre()-перевернуть список.

shuffle():-перемешать элементы коллекции

unmodifiableList()-создает из переданного списка его неизменяемый вариант. В него нельзя будет ни добавить, ни удалить элемент. И многие другие.

1. Ранние версии Java не включали в себя коллекцию Framework. Вместо этого он определил несколько классов и один интерфейс для хранения объектов. Когда появилась коллекция, эти классы были переработаны для поддержки интерфейсов Коллекции. Эти старые классы известны как унаследованные классы. Унаследованные классы известны как legacy classes. Legacy classes-Dictionary, HashTable, Properties, Stack, Vector Legacy interface- Enumeration.

• Enumeration ---Interface

• Dictonary ------Abstract class

• Hashtable -----Concrete class

• Properties -----Concrete class

• Vector -----Concrete class

• Stack -----Concrete class

Все эти классы были описаны сверху множество раз.