1. ArrayList/Vector

* Векторы синхронизированы, ArrayLists -нет.

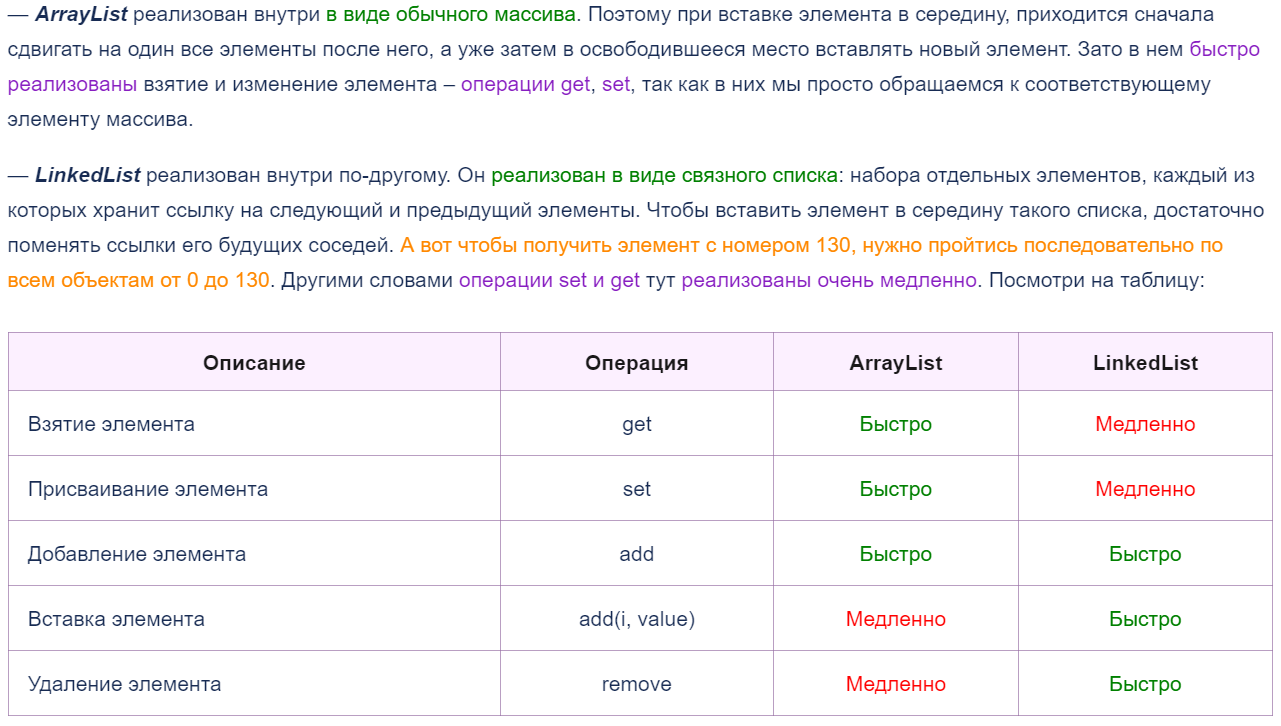
Несколько потоков могут работать с ArrayList одновременно, поэтому он считается несинхронизированным . В отличие от ArrayList, только один поток может работать с вектором одновременно; следовательно это называется Синхронизировано.

* Методы Роста Данных

Когда элемент вставляется в ArrayList или вектор, объект должен будет расширить свой внутренний массив, если у него закончится место. Вектор по умолчанию удваивает размер своего массива, в то время как ArrayList увеличивает его размер на 50 процентов.

1. Связный список

LinkedList — реализует интерфейс List. Является представителем двунаправленного списка, где каждый элемент структуры содержит указатели на предыдущий и следующий элементы

1. ArrayList/LinkedList
2. HashSet

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». HashSet не является структурой данных с встроенной синхронизацией, поэтому если с ним работают одновременно несколько потоков, и как минимум один из них пытается внести изменения, необходимо обеспечить синхронизированный доступ извне. Часто это делается за счет другого синхронизируемого объекта, инкапсулирующего HashSet.

1. TreeSet

Класс TreeSet в Java обеспечивает реализацию интерфейса Set, который использует древо для хранения. Объекты хранятся в отсортированном и возрастающем порядке. Время доступа и поиска довольно быстрое, что делает TreeSet отличным выбором при хранении большого количества отсортированной информации, которая должна быть найдена быстро.

1. HashMap

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.

— Ключи и значения могут быть любых типов, в том числе и null. Для хранения примитивных типов используются соответствующие классы-оберки;

— Не синхронизирован.

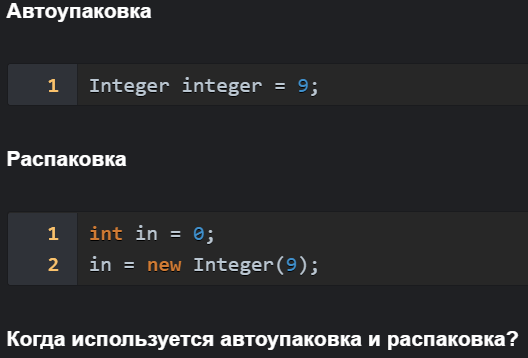
1. TreeMap



1. Затирание типов в дженериках

Обратите внимание, что во время компиляции компилятор имеет полную информацию о типе, но эта информация *обычно намеренно отбрасывается при* создании байтового кода в процессе, известном как *стирание типа* . Это делается таким образом из-за проблем совместимости: целью разработчиков языка было обеспечение полной совместимости исходного кода и полной совместимости байтового кода между версиями платформы. Если бы он был реализован по-другому, вам пришлось бы перекомпилировать устаревшие приложения при переходе на более новые версии платформы. Таким образом, все сигнатуры методов сохраняются (совместимость с исходным кодом), и вам не нужно ничего перекомпилировать (двоичная совместимость).

1. Ещё про дженерики: Дженерики (обобщения) — это особые средства языка Java для реализации обобщённого программирования: особого подхода к описанию данных и алгоритмов, позволяющего работать с различными типами данных без изменения их описания. Цели более сильная проверка типов во время компиляции и устранение необходимости явного приведения.
2. Авто упаковка

В версиях ниже JDK 1.5 было не легко преобразовывать примитивные типы данных, такие как int, char, float, double в их классы оболочки Integer, Character, Float, Double. Начиная с версии JDK 5 эта функция, преобразования примитивных типов в эквивалентные объекты, реализована автоматически.

Это свойство известно как Автоупаковка (Autoboxing). Обратный процесс соответственно – Распаковка (Unboxing) т.е. процесс преобразования объектов в соответствующие им примитивные типы.

1. NIO/IO

Данные получаются в буффер, которым можно оперировать и за которым надо следить / Данные читаются из стрима (набора байт) и нигде не кэшируются