- Интерфейсы Comparable Comparator

Comparable

Comparable - автоматическая сортировки при условии реализации коллекции (сортировка по умолчанию)

С английского "Comparable"переводится как "сравнимый". Имплементируя этот интерфейс мы как бы говорим "Эй, теперь объекты этого класса можно сравнивать между собой! И я знаю, как это сделать!"А до этого было нельзя

- ноль, если два объекта равны;
- число >0, если первый объект (на котором вызывается метод) больше, чем второй (который передается в качестве параметра);
- число <0, если первый объект меньше второго.

```
public class House implements Comparable<House>{
    int area;
    int price;
    String city;
    boolean hasFurniture;

    public House(int area, int price, String city, boolean hasFurniture)
    {
        this.area = area;
        this.price = price;
        this.city = city;
        this.hasFurniture = hasFurniture;
    }

    public int compareTo(House anotherHouse)
    {
        if (this.area == anotherHouse.area) {
            return 0;
        } else if (this.area < anotherHouse.area) {
            return -1;
        } else {
            return 1;
        }
    }
}
```

/

Comparator

Comparator - сортировка по уникальному критерию. Невстрое и прописывается отдельно

Итак, нестандартная сортировка. Допустим, мы все согласны что логичнее всего сравнивать дома по площади. Ну а если их нужно отсортировать, например, по цене?

Для этой цели мы можем создать отдельный класс, который реализует интерфейс Comparator.

```java
public class PriceComparator implements Comparator<House> {

    public int compare(House h1, House h2) {
        if (h1.price == h2.price) {
            return 0;
        }
        if (h1.price > h2.price) {
            return 1;
        }
        else {
            return -1;
        }
    }
}
```

- Коллекции Для хранения групп объектов одинакового типа в языке программирования Java был разработан Collections Framework.

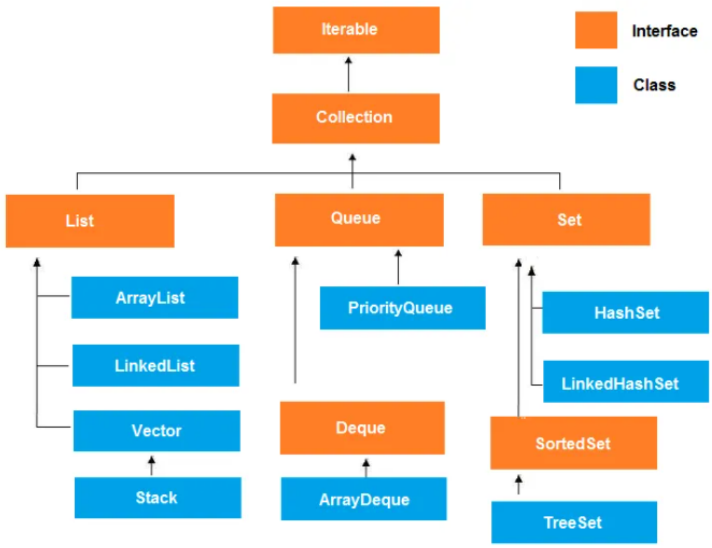
  К нему были предъявлены следующие требования:

  – Должен позволять работать с разными типами данных одинаково и с высокой степенью совместимости.

  – Должен иметь высокую производительность и эффективно реализовывать фундаментальные структуры данных такие, как связный список, деревья, хэш-таблицы и динамические массивы.

  – Должен позволять легко создавать собственные коллекции для специфических задач.

  Для реализации этих целей, был разработан набор интерфейсов, на базе которых были созданы готовые коллекции, позволяющие разрабатывать коллекции собственные.



Интерфейсы

– Collection — это основа, на которой построена структура коллекций в Java. Он объявляет основные методы, которые будут

иметь все коллекции: удаление, добавление, проверка на наличие, очистка, размер.

- List – Список объекты хранятся в порядке их добавления в список. Доступ к элементам списка осуществляется по индексу.
  - Элементы могут быть вставлены или доступны по их позиции в списке, используя индекс на основе нуля.
  - Список может содержать повторяющиеся элементы.
- Queue- определяет поведение класса в качестве однонаправленной очереди FIFO - первый пришел, первый ушёл.
- Deque - двунаправленная очередь
- Set – множество неповторяющихся объектов. В коллекции этого типа разрешено наличие только одной ссылки типа null.
- SortedSet - расширяет Set и объявляет поведение набора, отсортированного по возрастанию.

Классы, реализованные в Коллекциях

- ArrayList - это класс, который является реализацией динамического массива, т.е. массива, который при необходимости увеличивает свой размер.
- LinkedList - класс, который является реализацией такой структуры данных, как связный список. Является представителем двунаправленного списка, где каждый элемент структуры содержит указатели на предыдущий и следующий элементы.
- Vector реализует динамический массив, однако в отличии от ArrayList он синхронизирован. Если несколько потоков одновременно обращаются к массиву ArrayList, мы должны синхронизировать блок кода, который модифицирует список либо структурно, либо просто модифицирует элемент. У Vector некоторые методы синхронизированы и поэтому они медленные.
- Stack - это подкласс Vector, который реализует стандартный стек last-in, first-out.
- PriorityQueue - Элементы упорядочиваются и в зависимости от компаратора, т. Е. не только стандартным порядком. Реализация на основе кучи, а хранилище фактически массив. LIFO
- ArrayDeque - Этот класс представляют обобщенную двунаправленную очередь, наследуя функционал от класса AbstractCollection и применяя интерфейс Deque.
- HashSet - Он создает коллекцию, которая использует хеш-таблицу для хранения.

  Хэш-таблица хранит информацию с помощью механизма, называемого хешированием. В хэшировании информационный контент ключа используется для определения уникального значения, называемого его хэш-кодом.

  Хэш-код затем используется как индекс, в котором хранятся данные, связанные с ключом. Преобразование ключа в его хэш-код выполняется автоматически. Итог - хэш - ключ

– LinkedHashSet - Класс LinkedHashSet наследует класс HashSet. Этот класс запоминает порядок добавления элементов.

  Поэтому, когда мы захотим увидеть содержимое структуры, то элементы будут выведены в том порядке, в котором они были добавлены, а не упорядочены, как в случае с HashSet.

– TreeSet - 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.

Интерфейс map и его реализации

Map - это структура данных, которая содержит набор пар "ключ-значение". По своей структуре данных напоминает словарь, поэтому ее часто так и называют.

Классы

– HashMap - в Java использует хэш-таблицу для реализации интерфейса Map. Это позволяет времени для выполнения основных операций, таких как get () и put (), оставаться постоянным даже для больших множеств.

– TreeMap обеспечивает эффективное средство хранения пар ключ/значение в отсортированном порядке и позволяет быстро извлекать данные. Следует отметить, что, в отличие от хэш-карты, карта деревьев гарантирует, что ее элементы будут отсортированы в порядке возрастания ключа.

– LinkedHashMap поддерживает связанный список записей на Map в том порядке, в котором они были вставлены. Это позволяет итерации ввода-вывода по Map. То есть при итерировании LinkedHashMap элементы будут возвращены в том порядке, в котором они были вставлены.

  Вы также можете создать LinkedHashMap, который возвращает свои элементы в том порядке, в котором они были в последний раз.

Параметризованные типы

По существу, обобщения - это параметризованные типы. Такие типы важны, поскольку они позволяют объявлять классы, интерфейсы и методы, где тип данных, которыми они оперируют, указан в виде параметра. Используя обобщения, можно, например, создать единственный класс, который будет автоматически обращаться с разнотипными данными. Классы, интерфейсы или методы, оперирующие параметризованными типами, называются обобщенными..

– Строгая проверка типов, на этапе компидяции

- Информация о типах стирается -> код универсален
- Не нужен явный cast инг
- Работает только с объектами
- Когда объявляется экземпляр обобщенного типа, аргумент, передаваемый в качестве параметра типа, должен относиться к ссылочному типу, но ни в коем случае не к примитивному типу вроде int или char.

Обобщения автоматически гарантируют типовую безопасность во всех операциях, где задействован обобщенный класс. В процессе его применения исключается потребность в явном приведении и ручной проверке типов в прикладном код. Из за чего могут возникать ошибки

- Не Generic версия

```
public class Box {

    private Object value;

    public Object getValue() {
        return value;
    }

    public void setValue(Object value) {
        this.value = value;
    }
}
```

- class name<T1, T2, ..., Tn> {}

```
public class Box<T> {

    private T value;

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }
}
```

Названия параметров типа

- E - элемент коллеции(java api)
- K - ключ
- V - значение
- N - число
- T - первый параметр типа
- S,U,V - 2 3 4 параметры

Ограничения Generics

- Нельзя создавать объекты генерик класса
- Создавать статичные поля с параметром типа
- Нельзя инстанцироваться с примитивными типами /инстансоф - ф-я которая проверяет является ли объект слева наследником объека справа
- Нельзя содавать массив параметров типа /это связано с выделением памяти
- Не может быть наследником класса Throwable

Bounded type parameters - явное ограничения на те типы, с которыми мы можем работать

```
public class Box<T> {

    private T t;

    public void set(T t) {
        this.t = t;
    }

    public <S extends Number> void run(S u){
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }

    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.set(new Integer( value: 10));
        integerBox.run("message"); // Ошибка: не принимаем строки
    }
}
```

WildCard

- Обозначаются как <?>
- Представляет неизвестный тип
- Используется как
  - Тип: параметра, поля, локальной переменной
- Никогда не используется как
  - Как супертип

```
public void run(Box<? extends Number> number) {
    System.out.println(number);
}
```

- run(new Box<Integer>)
- run(new Box<Foo>), где Foo extends Number

```
public static void addNumbers(List<? super Integer> list) {
    for (int i = 1; i <= 10; i++) {
        list.add(i);
    }
}
```

PECS - producer - extends, consumer - super

PECS - это с точки зрения коллекции. Если вы только вытаскиваете элементы из общей коллекции, это производитель, и вы должны использовать extends если вы только набиваете элементы, это потребитель, и вы должны использовать super. Если вы делаете и то, и другое с одной и той же коллекцией, вы не должны использовать ни extends, ни super.

Классы-оболочки. Назначение, область применения, преимущества и недостатки. Автоупаковка

Классы оболочки - созданы для того чтобы приводить примитивные типы, которые не являются объектами к объекатам, у которых больший потенциал - методы, коллекции.

- Примитивный тип
  - передача по значению - передаете копию текущего значения

    * не является потомком класса Object
    * не имеет методов
    * арифметические операции
- Ссылочный тип
    * передача по ссылке - передаете копию ссылки
    * потомок класса Object
    * есть методы

Виды оболочек

- Number - абстрактный класс
  Имеет методы типа int intValue()
  Потомки - Byte, Short, Integer, Long, Float, Double
- Character
- Boolean

У всех оболочек есть методы - minValue, maxValue, hashCode, equals, toString. Также все они реализуют интерфейс Comparable

Методы преобразования

- примитив <-> объект
    * static Integer valueOf(int)
    * int intValue()
- строка <-> объект
    * static Integer valueOf(String)
    * String toString()
- строка <-> примитив
    * static int parseInt(String)
    * + int

Автоупаковка/автораспаковка

Мы можем сократить код и джава сама обернет примитив в оболочку примеры

- Integer a = Integer.valueOf(5); <-> Integer a = 5;
- int i = a.intValue(); <-> int i = a;

Однако если у нас происходит работа в цикле, то для оптимизации следует работать с примитивами и после выполнения производитьь обертку.

Работа с файлами в Java. Класс java.io.File.

Чтение происходит побайтово

В отличие от большинства классов ввода/вывода, класс File работает не с потоками, а непосредственно с файлами. Данный класс позволяет получить информацию о файле: права доступа, время и дата создания, путь к каталогу. А также осуществлять навигацию по иерархиям подкаталогов.

Класс File, определенный в пакете java.io, не работает напрямую с потоками. Его задачей является управление информацией о файлах и каталогах. Хотя на уровне операционной системы файлы и каталоги отличаются, но в Java они описываются одним классом File.

В зависимости от того, что должен представлять объект File - файл или каталог, мы можем использовать один из конструкторов для создания объекта:

По сути File это

- Это абстрактное представление путей к файлам и каталогам.
- Имя пути, абстрактное или строковое, может быть абсолютным или относительным.
- Родитель абстрактного пути может быть получен путем вызова метода getParent () этого класса.
- Прежде всего, мы должны создать объект класса File, передав ему имя файла или имя каталога. Файловая система может устанавливать ограничения на определенные операции с фактическим объектом файловой системы, такие как чтение, запись и выполнение. Эти ограничения в совокупности известны как разрешения на доступ.
- Экземпляры класса File неизменяемы; то есть, однажды созданный абстрактный путь, представленный объектом File, никогда не изменится.
- File(String путь к каталогу)
- File(String путь к каталогу, String имя файла)
- File(File каталог, String имя файла)

Методы, реализуемые классом File:

Пакет java.nio - назначение, основные классы и интерфейсы.

Содержит классы для ввода вывода. Java NIO - часть стандратной библиотеки Java, предназначенная для работы с вводом-выводм Отличия от IO

- Неблокирующий - При работе с обычными потоками ввода-вывода вы обычно вызываете метод read (write) для получения очередной порции данных. Что же происходит с вашей программой в этот момент? Она останавливается до тех пор, пока эти данные не появятся в потоке.

  NIO - не ожидает появления данных в канале (см. далее), а получает уже имеющиеся в нём данные, а если их нет - не получает ничего, после этого программа может перейти к исполнению следующих инструкций.

- Асинхронный - В IO read/write инициирует процесс получения данных: например, чтение из файла на жёстком диске. В NIO же данные в канал поступают независимо от программы - достаточно связать канал с источником данных, и при их появлении они будут "кэшироваться"в нём. Операция чтения из канала просто получит уже имеющиеся в нём данные.

- Если в IO работа сданными происходит побайтово/посимвольно (или с массивами байт/символов), то NIO работает с буферами, которые можно воспринимать как высокоуровневые обёртки над массивом байт. В отличии от буферизации в том же BufferedInputStream'е (и прочих Buffered*), программисту даётся полный контроль над буфером: существует большое число методов по модификации его содержимого, навигации и т.д

Основные классы

- java.nio.Buffer - Абстрактный класс контейнер для хранения данных. Имеет campasiti - макс.размер, limit - сколько можно записать в данный момент, position - текущая позиция

  Буферы в Java NIO можно рассматривать как простой объект, который действует как контейнер с фиксированным размером блоков данных, которые можно использовать для записи данных в канал или чтения данных из канала, чтобы буферы действовали как конечные точки для каналов.

  Он предоставляет набор методов, которые делают более удобным обращение с блоком памяти для чтения и записи данных в и из каналов.

  Буферы делают пакет NIO более эффективным и быстрым по сравнению с классическим вводом-выводом, поскольку в случае ввода-вывода данные обрабатываются в виде потоков, которые не поддерживают асинхронный и параллельный поток данных. Также ввод-вывод не позволяет выполнять данные в чанке или группе байтов. ,

  - Создание буфера: allocate(capacity), wrap(array[])

  Методы

  - limit(lim) и position(pos)
  - mark() и reset() mark <-> position
  - clear() - переход в режим записи - позиция = 0, граница = емкость
  - compact - все недочитанное - в начало буфера
  - flip() - переход в режим чтения - граница = позиция, позиция = 0
  - rewind() - повторное чтение - позиция = 0

- Класс Charset - перевести из одной кодировки в другую

- Channels - ввод/вывод. Делятся на Файловые каналы и сетевые каналы Отличие от потока

  Определяет каналы, которые представляют подключения к объектам, способным выполнять операции ввода-вывода, например к файлам и сокетам; определяет селекторы для мультиплексированных неблокирующих операций ввода-вывода.

  - Можно читать из канала и писать в канал одновременно
  - Можно читать из канала и писать в канал асинхронно
  - Можно читать из канала в буфер и писать в канал из буфера

- java.nio.file.Path

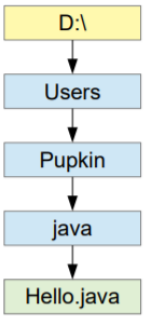
  Как следует из названия, Path — это конкретное местоположение объекта, такого как файл или каталог в файловой системе, чтобы можно было искать и получать к нему доступ в этом конкретном месте.

  В общем случае путь объекта может быть двух типов: один — абсолютный путь, а другой — относительный путь. Поскольку имя обоих путей предполагает, что абсолютный путь — это адрес местоположения от корня до объекта, в котором он находится, а относительный путь — это адрес местоположения. который относится к какому-либо другому пути. В своем определении Path использует разделители как « для Windows и «/» для операционных систем Unix.

  Чтобы получить экземпляр Path, мы можем использовать статический метод класса java.nio.file.Paths get (). Этот метод преобразует строку пути или последовательность строк, которые при соединении образуют строку пути, в экземпляр Path Этот метод также генерирует InvalidPathException во время выполнения, если переданные аргументы содержат недопустимые символы.

  - абсолютный / относительный
  - пустой — текущая директория
  - Методы:

```
Path getRoot()       // D:\
int getNameCount()   // 4
Path getName(1)      // Pupkin
Path getParent()     // D:\Users\Pupkin\java
Path getFileName()   // Hello.java
Path subpath(1,3)    // Pupkin/java/Hello.java
```



- FileChannel

  В FileChannel реализована реализация канала Java NIO для доступа к свойствам метаданных файла, включая создание, изменение, размер и т. Д. Наряду с этим каналы являются многопоточными, что снова делает Java NIO более эффективным, чем Java IO.

  В общем, мы можем сказать, что FileChannel — это канал, который подключен к файлу, по которому вы можете читать данные из файла и

записывать данные в файл. Другая важная характеристика FileChannel заключается в том, что его нельзя перевести в неблокирующий режим. и всегда работает в режиме блокировки.

Мы не можем получить объект файлового канала напрямую, объект файлового канала получается либо —

– getChannel () — метод для любого FileInputStream, FileOutputStream или RandomAccessFile.

  open () — метод File channel, который по умолчанию открывает канал.

Тип объекта канала File зависит от типа класса, вызываемого при создании объекта, т. Е. Если объект создается путем вызова метода getchannel из FileInputStream, то канал File открывается для чтения и выдает исключение NonWritableChannelException в случае попытки записи в него.

Основные Интерфейсы