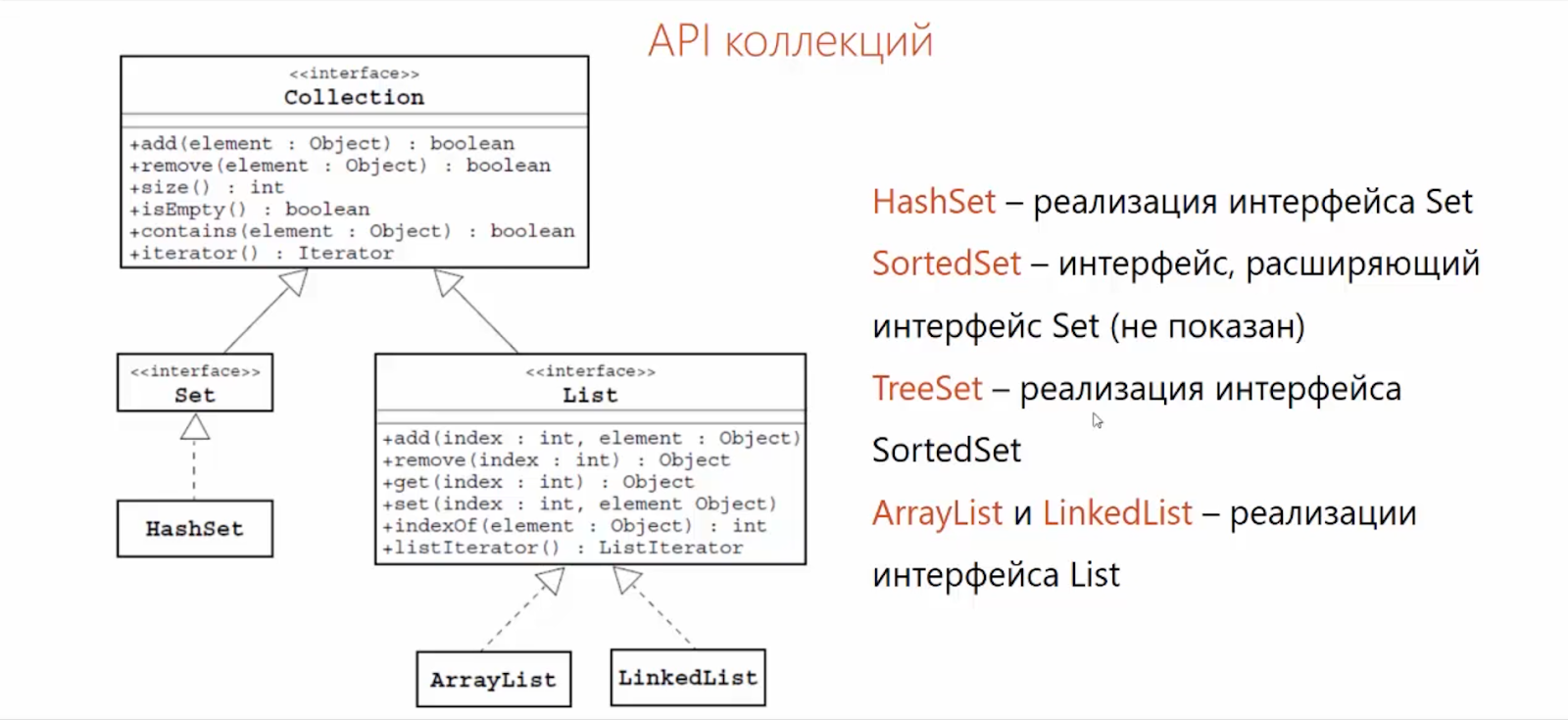
0) Иерархия коллекций. Сортировка элементов коллекции.

<https://javastudy.ru/interview/collections/>

1) Коллекции. Интерфейсы java.util.Comparable и java.util.Comparator.



Коллекции бывают гомогенные/generic(состоящие из объектов одного типа) и гетерогенные(из объектов разных типов)

Collection - группа объектов(возможно, повторяющихся) без определенного порядка

Set - неупорядоченная коллекция без дубликатов

Реализации Set:

-HashSet - неупорядоченный Set, основанный на HashMap

-LinkedHashSet - упорядоченный в порядке добавления Set, основанный на LinkedHashMap

-TreeSet - упорядоченный по возрастанию Set, с облегченным поиском элементов

List - упорядоченная коллекция, дубликаты возможны

Реализации List:

-Vector и Stack - синхронизированные динамические массивы

-ArrayList - обычный динамический массив. При достижении вместимости (capacity = 10 по дефолту; либо указать в конструкторе) в памяти создается новый массив размера 1,5\*capacity+1. При добавлении элемента внутрь массива все правые элементы сдвигаются на одну ячейку вправо.

-LinkedList - реализация интерфейсов List и Deque, в котором у каждого элемента есть ссылка на предыдущий и следующий, что позволяет свободно передвигаться по массиву в обе стороны. Имеет методы с окончаниями ...last() и ...first(), которые позволяют работать с обеими концами массива.

Queue - очередь типа FiFo, имеет методы для просмотра головного(нулевого) элемента очереди (element() и peek() не удаляющие его; poll() и remove() удаляющие его). Реализация Queue:

-PriorityQueue - очередь с возможностью сортировки компаратором или компэрэблом. Стандартная вместимость - 11.

Deque - двунаправленная очередь. Имеет методы с окончаниями ...last() и ...first(), которые позволяют работать с обеими концами очереди. Реализация Deque:

-ArrayDeque

Map - группа пар ключ:значение, не является наследником Collection.

Ключи являются уникальными(что обеспечивают методы hashcode и equals)

Реализации Map:

-HashTable - синхронизированный(медленный) аналог HashMap

-HashMap - неупорядоченный список. Порядок добавления ключей вычисляется с помощью хэш-кода, а именно: HashMap создает в памяти ArrayList с вместимостью 16, и при добавлении помещает ключ в ячейку с номером (hashcode(key)&(capacity-1)). Если несколько элементов имеют одинаковый хэш-код(и соответственно помещаются в одну ячейку), то эта ячейка превращается в LinkedList(если <6 элементов)/TreeSet с меньшим временем доступа.

-LinkedHashMap - упорядоченный в порядке добавления список

-TreeMap - упорядоченный в порядке возрастания список, в котором поиск элемента осуществляется на основе красно-черного дерева

-EnumMap - карта, у которой ключами являются enum’ы

-IdentityHashMap - карта, в которой уникальность ключей сравнивается не по equals, а ссылке на ключ

-WeakHashMap - использует слабые ссылки

Интерфейсы Comparable и Comparator нужны для сортировки сложных объектов в коллекции. Comparable имплиментируется непосредственно в сравниваемом классе, где нужно будет реализовать метод compareTo(Obj o). Когда у нас, например, нет доступа к нужному классу, создается класс-помощник с интерфейсом Comparator и методом compare(Obj o1, Obj o2) - объект класса-помощника нужно будет передать аргументом в коллекцию для сортировки. Сама сортировка осуществляется двумя способами: Collections.sort(.л..) или автоматически в коллекциях типа TreeSet и TreeMap.

public int compareTo(Person p){

return name.compareTo(p.getName());

}

Интерфейс компаратора определяет специальный метод по умолчанию thenComparing, который позволяет использовать цепочки компараторов для сортировки набора:

Comparator<Person> pcomp = new PersonNameComparator().thenComparing(new PersonAgeComparator());

2) Категории коллекций - списки, множества. Интерфейс java.util.Map и его реализации.

3) Отличие методов в коллекциях List, Set, Queue.

4) Отличие ArrayList и LinkedList. Устройство в памяти ArrayList.

5) HashMap

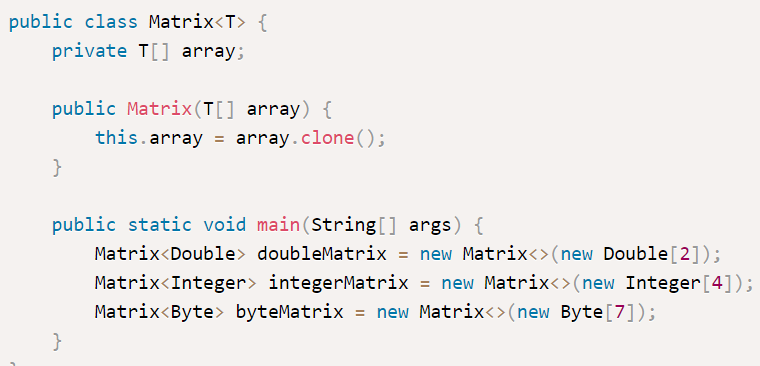
6) Параметризованные типы. Создание параметризуемых классов. Wildcard-параметры.

<https://www.examclouds.com/ru/java/java-core-russian/generics-russian>

Дженерики (или обобщения) - это параметризованные типы.

Параметризованные типы позволяют объявлять классы, интерфейсы и методы, где тип данных, которыми они оперируют, указан в виде параметра. Используя дженерики, можно создать единственный класс, например, который будет автоматически работать с разными типами данных.

ПАРАМЕТРИЗИРОВАННЫЕ КЛАССЫ



Дженерики работают только с объектами

Т обозначает имя параметра типа.

ОГРАНИЧИТЕЛЬНЫЕ ТИПЫ

Указывая параметр типа, можно наложить ограничение сверху в виде верхней границы, где объявляется супер класс, от которого должны быть унаследованы все аргументы типов. С этой целью вместе с параметром указывается ключевое слово extends:

class Gen <Т extends Superclass>

Параметр типа Т может быть заменен только указанным супер классом или его подклассами.

В виде ограничения можно накладывать не только тип класса, но и тип интерфейса:

public class MyClass<T extends Serializable>

Ограничение может включать в себя как тип класса, так и типы одного или нескольких интерфейсов:

class Gen <T extends MyClass & MyInterface1 & MyInterface2>

Тип класса должен быть задан первым. Накладывая на обобщенный тип ограничение, состоящее из класса и одного или нескольких интерфейсов, для их объединения следует воспользоваться логической операцией &: Таким образом, любой тип, передаваемый параметру Т, должен быть подклассом, производным от класса MyClass и реализующим интерфейсы MyInterface1 и MyInterface2.

МЕТАСИМВОЛЬНЫЕ АРГУМЕНТЫ

boolean sameAvg(Average<T> ob) {

return average() == ob.average();

} – Это не сработает так как в этом случае тип передаваемого объекта должен будет совпадать с this объектом

boolean sameAvg(Average<?> ob) { - ? представляет собой неизвестный тип, также может быть ограничен.

ПАРАМЕТРИЗИРОВАННЫЕ МЕТОДЫ И КОНСТРУКТОРЫ

public static <T, V> boolean isIn(T x, V[] array) {

public <T extends Number> GenConstructor(T arg) {

ПАРАМЕТРИЗИРОВАННЫЕ ИНТЕРФЕЙСЫ

public interface MyInterface<T> {

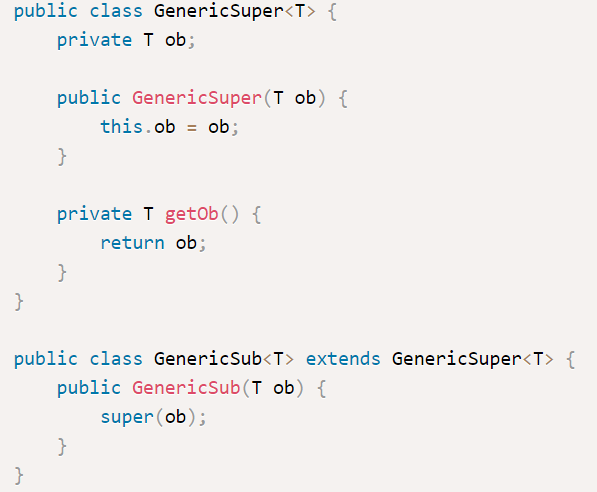
T someMethod(T t);

}

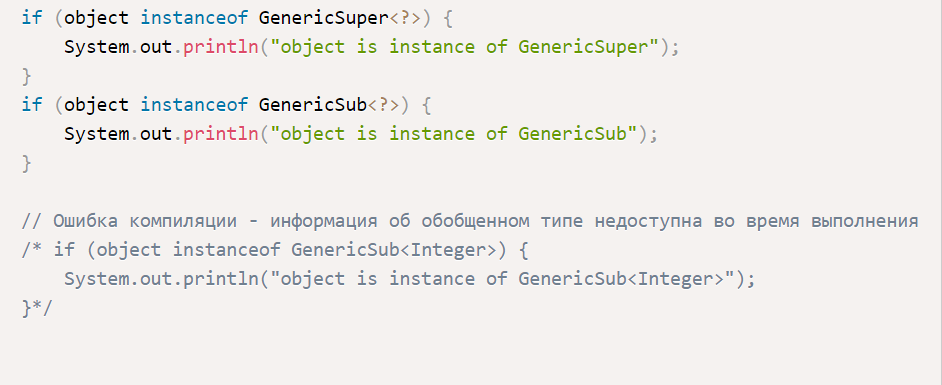
public class MyClass<T> implements MyInterface<T> {

ИЕРАРХИЯ ПАРАМЕТРИЗИРОВАННЫХ КЛАССОВ

Ключевое отличие между параметризованными и не параметризованными иерархиями состоит в том, что в параметризованной иерархии любые аргументы типов, необходимые параметризованному супер классу, всеми подклассами должны передаваться по иерархии вверх.

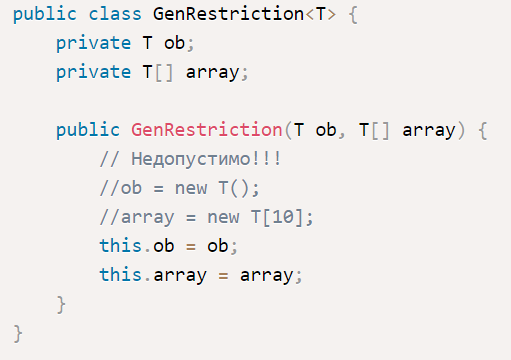


ОПЕРАТОР INSTANCEOF

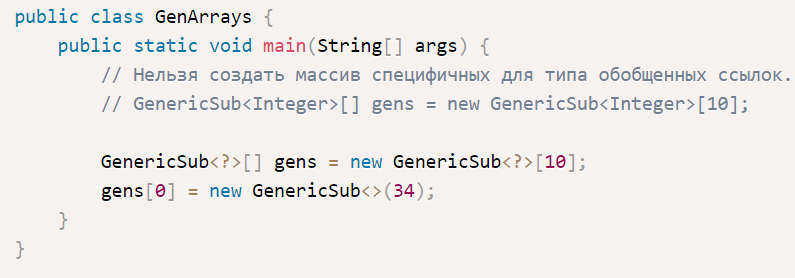


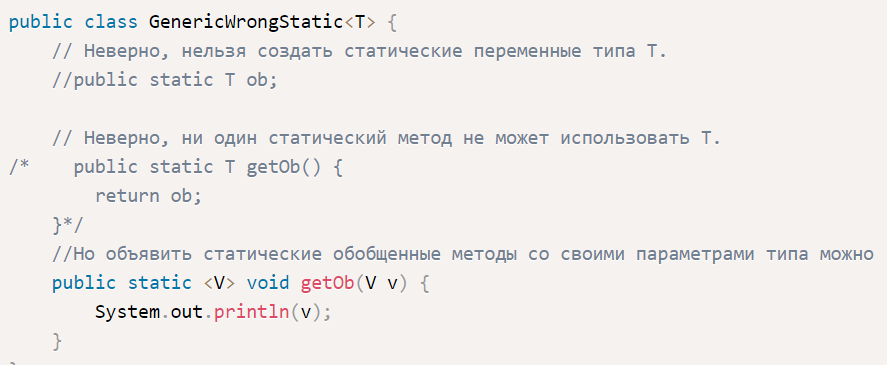
ОГРАНИЧЕНИЯ ОБОБЩЕНИЙ

1. Нельзя создавать экземпляр по параметру типа. Ни обычный объект, ни массив.



2. Нельзя создать массив специфических для типа обобщенных ссылок:



3. Нельзя создавать обобщенные статические переменные и методы. Но объявить статические обобщенные методы со своими параметрами типа все же можно

ДОПОЛНЕНИЕ

который “оперирует” объектами типов T1,..., т.е. конструктор ожидает не конкретный класс, а абсолютно любой, но в таком случае мы должны обрабатывать этот объект универсальными для любого класса методами .getClass и тд.

Т.к. Collection<Animal> != Collection<Dog> (при Dog extends Animal), то на помощь приходят так называемые wildcards. Это конструкция вида Collection<? extends Animal> или Collection<? super Dog>. Первая приравнивается ко всем дженерик коллекциям типов-наследников Animal, вторая приравнивается ко всем дженерикам типов-предков класса Dog.

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

Обертка — это специальный класс, который хранит внутри себя значение примитива

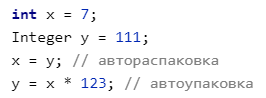
Классы оболочки обладают методами, которых нет у примитивов.

Так как параметры методам передаются по ссылке (или копии примитивов), то классы оболочки можно использовать там.

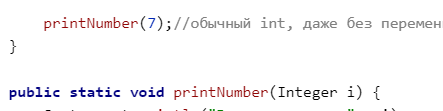
Integer.MAX(MIN)\_VALUE

Integer a = 1 (создается новый объект). Иммутабельность (неизменяемость).

АВТОУПАКОВКА, АВТОРАСПАКОВКА.



Именно поэтому добавляется слово "авто": для присваивания ссылок-примитивов объектам их классов-оберток (и наоборот) не требуется ничего делать, все происходит автоматически.

и наоборот

**Примитивы:**

они занимают мало памяти, за счет чего повышается эффективность работы программы, и четко разделены по диапазонам значений.

**Обертки:**

Позволяют не нарушать принцип “все является объектом”, благодаря чему числа, символы и булевы значения true/false не выпадают из этой концепции

Расширяют возможности работы с этими значениями, предоставляя удобные методы и поля

Необходимы, когда какой-то метод может работать исключительно с объектами

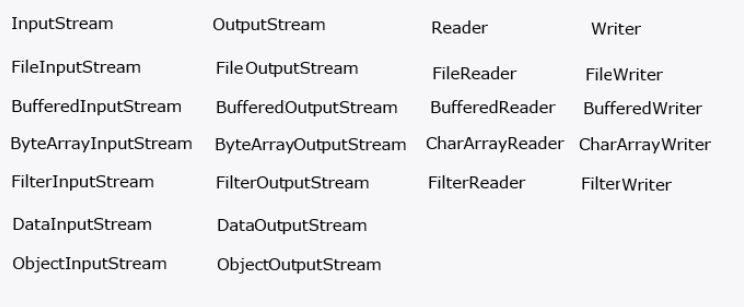
8) Потоки ввода-вывода в Java. Байтовые и символьные потоки. "Цепочки" потоков (Stream Chains).

<https://metanit.com/java/tutorial/6.8.php>

[https://studfile.net/preview/2090788/page:2/#10](https://studfile.net/preview/2090788/page:2/%2310)

В основе всех классов, управляющих потоками байтов, находятся два абстрактных класса: InputStream (представляющий потоки ввода) и OutputStream (представляющий потоки вывода)

Но поскольку работать с байтами не очень удобно, то для работы с потоками символов были добавлены абстрактные классы Reader (для чтения потоков символов) и Writer (для записи потоков символов).



Класс InputStream

Класс InputStream является базовым для всех классов, управляющих байтовыми потоками ввода. Рассмотрим его основные методы:

int available(): возвращает количество байтов, доступных для чтения в потоке

void close(): закрывает поток

int read(): возвращает целочисленное представление следующего байта в потоке. Когда в потоке не останется доступных для чтения байтов, данный метод возвратит число -1

int read(byte[] buffer): считывает байты из потока в массив buffer. После чтения возвращает число считанных байтов. Если ни одного байта не было считано, то возвращается число -1

int read(byte[] buffer, int offset, int length): считывает некоторое количество байтов, равное length, из потока в массив buffer. При этом считанные байты помещаются в массиве, начиная со смещения offset, то есть с элемента buffer[offset]. Метод возвращает число успешно прочитанных байтов.

long skip(long number): пропускает в потоке при чтении некоторое количество байт, которое равно number

Класс OutputStream

Класс OutputStream является базовым классом для всех классов, которые работают с бинарными потоками записи. Свою функциональность он реализует через следующие методы:

void close(): закрывает поток

void flush(): очищает буфер вывода, записывая все его содержимое

void write(int b): записывает в выходной поток один байт, который представлен целочисленным параметром b

void write(byte[] buffer): записывает в выходной поток массив байтов buffer.

void write(byte[] buffer, int offset, int length): записывает в выходной поток некоторое число байтов, равное length, из массива buffer, начиная со смещения offset, то есть с элемента buffer[offset].

Абстрактные классы Reader и Writer

Абстрактный класс Reader предоставляет функционал для чтения текстовой информации. Рассмотрим его основные методы:

absract void close(): закрывает поток ввода

int read(): возвращает целочисленное представление следующего символа в потоке. Если таких символов нет, и достигнут конец файла, то возвращается число -1

int read(char[] buffer): считывает в массив buffer из потока символы, количество которых равно длине массива buffer. Возвращает количество успешно считанных символов. При достижении конца файла возвращает -1

int read(CharBuffer buffer): считывает в объект CharBuffer из потока символы. Возвращает количество успешно считанных символов. При достижении конца файла возвращает -1

absract int read(char[] buffer, int offset, int count): считывает в массив buffer, начиная со смещения offset, из потока символы, количество которых равно count

long skip(long count): пропускает количество символов, равное count. Возвращает число успешно пропущенных символов

Класс Writer определяет функционал для всех символьных потоков вывода. Его основные методы:

Writer append(char c): добавляет в конец выходного потока символ c. Возвращает объект Writer

Writer append(CharSequence chars): добавляет в конец выходного потока набор символов chars. Возвращает объект Writer

abstract void close(): закрывает поток

abstract void flush(): очищает буферы потока

void write(int c): записывает в поток один символ, который имеет целочисленное представление

void write(char[] buffer): записывает в поток массив символов

absract void write(char[] buffer, int off, int len) : записывает в поток только несколько символов из массива buffer. Причем количество символов равно len, а отбор символов из массива начинается с индекса off

void write(String str): записывает в поток строку

void write(String str, int off, int len): записывает в поток из строки некоторое количество символов, которое равно len, причем отбор символов из строки начинается с индекса off

Функционал, описанный классами Reader и Writer, наследуется непосредственно классами символьных потоков, в частности классами FileReader и FileWriter соответственно, предназначенными для работы с текстовыми файлами.

FileInputStream, FileOutputStream работают с байтами, в основном предназначены именно для этого

МЕТОД FLUSH()

метод flush() выбрасывает всё из буфера в соответствующий поток. Если поток PrintWriter создан на базе стандартного потока вывода System.out, то записанные в него данные появятся на консоли только после выполнения сброса буфера (автоматического или сделанного методом flush(). Поправьте кто, если я не прав, но метод println() класса PrintWriter помещает значение в буфер самого PrintWriter, а уже при flush() толкает в System.out

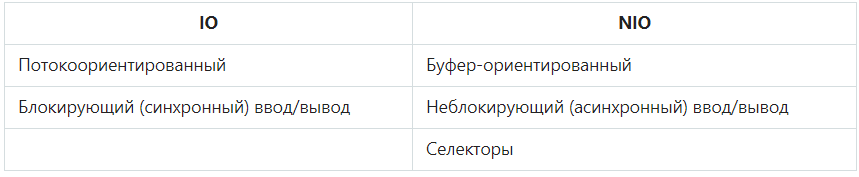
9) Работа с файлами в Java. Класс java.io.File.

<https://metanit.com/java/tutorial/6.11.php>

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.

10) Отличие блокирующих и неблокирующих IO.

<https://habr.com/ru/post/235585/>



Каналы и селекторы: NIO поддерживает различные типы каналов. Канал является абстракцией объектов более низкого уровня файловой системы (например, отображенные в памяти файлы и блокировки файлов), что позволяет передавать данные с более высокой скоростью. Каналы не блокируются и поэтому Java предоставляет еще такие инструменты, как селектор, который позволяет выбрать готовый канал для передачи данных, и сокет, который является инструментом для блокировки.

Буферы: в Java 4 была введена буферизация для всех классов-обёрток примитивов (кроме Boolean). Появился абстрактный класс Buffer, который предоставляет такие операции, как clear, flip, mark и т.д. Его подклассы предоставляют методы для получения и установки данных.

Кодировки: в Java 4 появились кодировки (java.nio.charset), кодеры и декодеры для отображения байт и символов Unicode.

11) Пакет java.nio - назначение, основные классы и интерфейсы. Устройства и преимущества пакетов IO и NIO.

<http://www.quizful.net/post/java-nio-tutorial>

<https://docs.oracle.com/javase/8/docs/api/index.html>

Буфферы. Временные хранилища фиксированного размера для транспортируемых данных. Именно буферизация – основное отличие неблокирующего чтения от java.io.

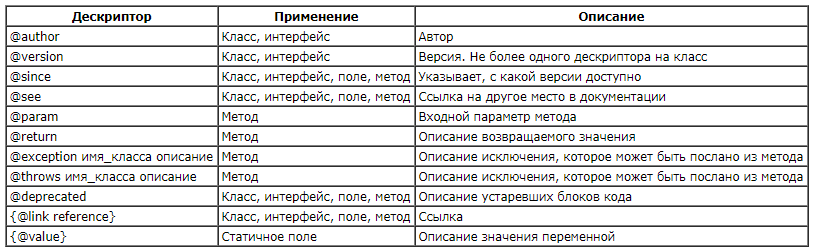
Каналы. Реализации интерфейса Channel – сущности, представляющие соединения между разными участниками ввода-вывода (файлы, сокеты, консоль).

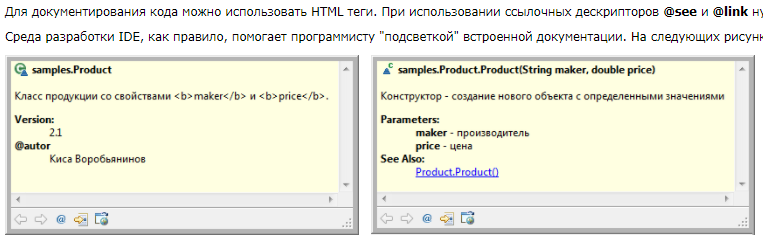
Селекторы. Наследники класса Selector. «Мультиплексоры» каналов – комбинируют несколько каналов в один. Регистрация канала в селекторе возвращает SelectionKey, который содержит ссылку на сам канал, и ряд его атрибутов. Селектор позволяет выбрать из набора зарегистрированных каналов подмножество готовых к работе, при необходимости блокируя выполнение на время ожидания. Каналы и селекторы располагаются в пакете java.nio.channels. Полный пример использования селекторов можно найти в статье на baeldung.

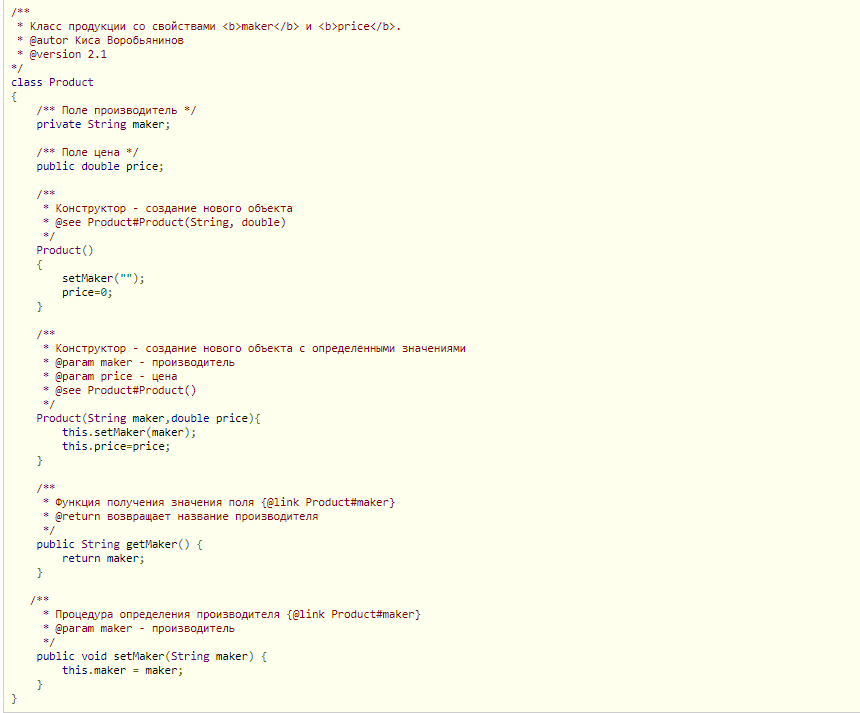
Кодировки. Charset – то, как бинарные данные будут конвертироваться в родные для Java символы UTF-16 и обратно. Классы для работы с кодировками хранятся в пакете java.nio.charset.

12) Утилита javadoc. Особенности автоматического документирования кода в Java.

javadoc — это генератор документации в HTML-формате из комментариев исходного кода Java и определяет стандарт для документирования классов Java. Для создания доклетов и тэглетов, которые позволяют программисту анализировать структуру Java-приложения, javadoc также предоставляет API. В каждом случае комментарий должен находиться перед документируемым элементом.







<http://java-online.ru/java-javadoc.xhtml>

13) dev/\*

/dev/null - специальный файл в системах класса UNIX, представляющий собой так называемое «пустое устройство». Запись в него происходит успешно, независимо от объёма «записанной» информации. Чтение эквивалентно считыванию конца файла.

/dev/zero - источник нулевых байтов. При чтении этого файла никогда не достигается его конец. Любые данные, записанные в /dev/zero, будут игнорированы, а сама запись завершается успешно.

/dev/full - Запись в /dev/full завершается успешно только для нулевого количества байт. Это устройство используется для добавления проверок того, что программа ничего не выводит. Представляет собой полное устройство, чтение как у /dev/zero

/dev/random — генератор случайных чисел.

/dev/urandom — генератор псевдослучайных чисел.

Они предоставляют интерфейс к системному генератору случайных чисел, который выводит шум из драйверов устройств и других источников в «хаотичный» пул (англ. entropy pool). Генератор также сохраняет необходимое количество битов шума в этом пуле и формирует из него случайную либо псевдослучайную последовательность чисел.

14) ZonedTimeDate, LocalTimeDate, Instant

<https://docs.oracle.com/javase/8/docs/api/java/time/ZonedDateTime.html>

<https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html>

<https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html>

Про перевод из одного в другое смысла наверное нет.

15) API, Arrays.

Java API — библиотека компонентов ПО доступных в готовом виде всем, кто установил Java Development Kit.

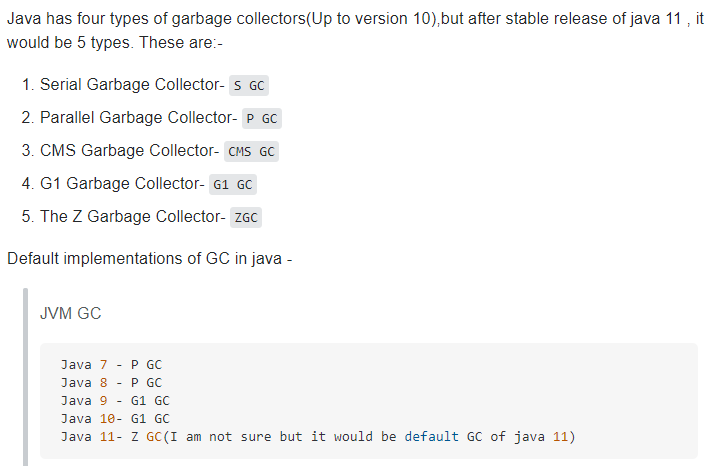
class Arrays (java.util)

This class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists.

Методы: sort, binarySearch, copyOf, equals и т.д.

16) Garbage Collectors

<https://habr.com/ru/post/269621/>



Виды ссылок

Сильные - обычные ссылки на объект без доступа к GarbageCollector

Мягкие(SoftReference) - когда теряем ссылку на объект, то сам объект будет удален из памяти когда JVM будет ее нехватать

Слабые(WeakReference) - когда теряем ссылку на объект, GC сразу же удаляет его из памяти

17) Serializable, сериализация.

Сериализация (Serialization) — это процесс, который переводит объект в последовательность байтов, по которой затем его можно полностью восстановить. Восстановить можно на любой ОС.

Десериализация — это процесс восстановления объекта из этих байт.

Интерфейс Serializable не включает ни одного метода, он нужен лишь для того, чтобы передать JVM необходимую информацию. Это так называемый интерфейс-маркер, который помечает классы дополнительной инфой.

SerialVersionUID используется для указании версии сериализованных данных.

Когда мы пытаемся провести десериализацию, то есть восстановить объект из набора байт, значение serialVersionUID сравнивается со значением serialVersionUID класса в нашей программе. Если значения не совпадают, будет выброшено исключение java.io.InvalidClassException.

В значение serialVersionUID можно записать любое число.

Во время сериализации, среда выполнения Java создает номер версии для класса, так что она может десереализировать его позже. В Java этот номер версии известен как SerialVersionUID. Если во время десериализации, SerialVersionUID не соответствует, то процесс завершится с исключением InvalidClassException в потоке "main" java.io.InvalidClassException, а также напечатает имя класса и соответствующий SerialVersionUID. Быстрое решение для исправления этой проблемы – скопировать SerialVersionUID и определить его как константу типа private static final long в вашем классе.

При сериализации объекта сериализуются все объекты, на которые он ссылается в своих переменных экземпляра. И если те объекты тоже ссылаются на третьи объекты, они тоже сериализуются. И так до бесконечности. Все классы в этой цепочке должны быть Serializable, иначе их невозможно будет сериализовать и будет выброшено исключение.

Проблемы такого рода решаются в Java при помощи ключевого слова transient. Если добавить к полю класса это ключевое слово — значение этого поля не будет сериализовано.

Заодно мы получили ответ на вопрос, какое же значение будет присвоено transient-полю. Ему присваивается значение по умолчанию. В случае с объектами это null.

<https://javarush.ru/groups/posts/2022-serializacija-i-deserializacija-v-java>

Первое отличие двух вариантов, которое бросается в глаза это размер кода. При реализации интерфейса Externalizable нам необходимо переопределить два метода: writeExternal() и readExternal(). В методе writeExternal() мы указываем какие поля будут сериализованы и как, в readExternal() как их прочитать.