**Теория по 5-й лабораторной.**

Оглавление

[Коллекции. Сортировка элементов коллекции. Интерфейсы java.util.Comparable и java.util.Comparator. 2](#_Toc107054206)

[Параметризованные типы. Создание параметризуемых классов. Wildcard-параметры. 4](#_Toc107054207)

[Классы-оболочки. Назначение, область применения, преимущества и недостатки. Автоупаковка и автораспаковка. 4](#_Toc107054208)

[Потоки ввода-вывода в Java. Байтовые и символьные потоки. "Цепочки" потоков (Stream Chains). 6](#_Toc107054209)

[Работа с файлами в Java. Класс java.io.File. 6](#_Toc107054210)

[Пакет java.nio - назначение, основные классы и интерфейсы. 10](#_Toc107054211)

# Коллекции. Сортировка элементов коллекции. Интерфейсы java.util.Comparable и java.util.Comparator.

Под коллекциями в программировании подразумевают объекты Которые хранят внутри себя какой-либо набор значений и предоставляют набор методов для обращения к этим значениям. В java выделяют три вида коллекций: list(списки), set(наборы), map(словари).

**Списки**

Они в java реализуются с помощью интерфейса List, который расширяет интерфейс **Collection.** Список позволяет хранить любые значения, в том числе и повторяющиеся. Итерация идет в порядке добавления командой add.

List<String> list = **new** ArrayList<>();  
list.add(**"яблоко"**);  
list.add(**"ананас"**);  
list.add(**"яблоко"**);  
System.out.println(list); *// На экране увидим: [яблоко, ананас, яблоко]*

Списки делятся на два класса: **ArrayList, LinkedList.**

**Первый класс** построен на базе массива, потому обращение по индексу идет быстро. А добавление элементов в середину затратно, так как надо двигать вправо каждый элемент, который идет после добавляемого. Кроме того, массив, лежащий в основе этой структуры, имеет конечное количество свободных ячеек. Если перестаёт хватать их, то придётся создавать новый массив большего размера и переносить туда данные из исходного.

**Второй класс** представляет собой цепочку элементов, в который каждый элемент имеет ссылку на предыдущий элемент и на следующий. Также имеется ссылка на начало и на конец списка, что позволяет быстро получать доступ к первому и последнему элементу. Но в отличии от другого класса для доступа по индексу придется проходится последовательно по всей цепочки, потому время доступа прямо пропорционально размеру списка. Однако процесс добавления и удаления элемента прост.

**Наборы (set)**

Интерфейс **Set** представляет собой набор уникальных значений и тоже наследуется от Collection. У этого интерфейса есть несколько реализаций, но каждая из них гарантирует, что каждое значение в наборе уникально. Сравнение происходит с помощью методов **equals()** и **hashCode()** из базового класса **Object.**

Есть три основные реализации интерфейса **Set**: HashSet, LinkedHashSet, TreeSet.

**HashSet.** Порядок зависит от хэш-функций каждого элемента. Также не добавляет элемент в набор, если такой уже имеется.

Следующая реализация - это **LinkedHashSet**, которая расширяет предыдущую. Основное различие заключается в том, что при обходе элементов мы будем видеть их в порядке добавления.

Ну а третья реализация под названием **TreeSet** имеет в своей основе структуру данных «красно-чёрное дерево», что позволяет сортировать элементы автоматически.

**Map**

Интерфейс представляет собой набор из пар элементов типа «ключ-значение». Словарь гарантирует, что каждому ключу соответствует одно и только одно значение.

**Put(K k, V v)** - помещает в коллекцию новый объект с ключом k и значением v. Если в коллекции уже есть объект с подобным ключом, то он перезаписывается. После добавления возвращает предыдущее значение для ключа k, если он уже был в коллекции. Если же ключа еще не было в коллекции, то возвращается значение null.

**Get(Oject k)** - возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение null

**Set<Map.Entry<K, V>> entrySet(): в**озвращает набор элементов коллекции. Все элементы представляют объект Map.Entry

V getOrDefault(Object k, V defaultValue): возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение defaultVlue

Есть три реализации: **HashMap, LinkedHashMap, TreeMap**

Первая реализация - это **HashMap**, которая не гарантирует никакого порядка элементов при обходе. Обратите внимание, что при повторном добавлении элемента с тем же ключом, мы теряем первое значение.

Ещё одна реализация - это **LinkedHashMap**, которая сохраняет порядок добавления.

Ну и третья популярная реализация интерфейса Map - это **TreeMap**, которая сортирует ключи по порядку.

Поэтому если хотите сохранять порядок добавления элементов - используйте LinkedHashMap, а если хотите получить отсортированный по ключам набор - тогда используйте TreeMap.

**Queue**

Коллекция предназначена для хранения элементов перед обработкой.

Помимо основных Collection операция, queue обеспечивают дополнительные операции вставки, извлечения и проверки. Он расширяет базовый интерфейс и определяет поведение класса в качестве однонаправленной очереди ( двигаться можно по одному направлению «голова – хвост»).

**Deque**

Этот интерфейс расширяет выше описанный интерфейс и определяет поведение двунаправленной очереди

**Сортировка.** Для сортировки в качетсве помощи выступают два интерфейса: Comparable и Comparator. Первый интерфейс сортирует объекты по возрастанию. Интерфейс Comparable является generic’ом – т.е. типом с параметром. У него всего один generic-метод – compareTo(T o). В этом методе и происходит сравнение переданного объекта (o) и текущего (this). Т.е. надо переопределить этот метод в своем классе и сравнить в нем текущий объект (this) с переданным.

**Comparator** используется, например, для сортировки колонок таблиц. У него есть метод **compare,** который на вход принимает два значения типа int. При использовании интерфейса логика сравнения пары объектов не прячется внутрь, а реализуется в отдельном классе. Вызов: **Collections.sort.**

# Параметризованные типы. Создание параметризуемых классов. Wildcard-параметры.

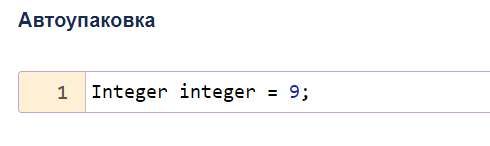
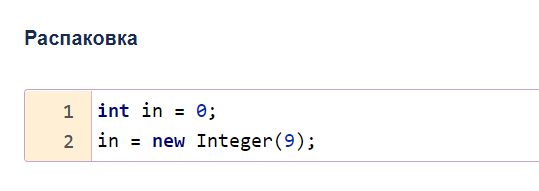
**Параметризованным** называется **тип**, описывающий сложную структуру данных (коллекцию), состоящую из однотипных элементов - например стек, дерево, граф, очередь, хеш-таблица, - в котором тип элементов является параметром, например Stack <t> (в синтаксисе языка Java).

**Wildcard –** специальные вопросительные знаки, отмечающие место, где должно оказаться название параметра типа. Они являются средством выражения ограничений в условиях неизвестности какого -либо типа.

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

Класс называется «оболочкой» потому, что он, по сути, копирует то, что уже существует, но добавляет новые возможности для работы с привычными типами. Имена таких классов несложно запомнить, ведь они повторяют имена примитивных типов: Byte, Short, Integer, Long, Float, Double, character, Boolean. Примитивы и классы-оболочки существуют параллельно, потому что у каждого есть преимущества. Например, обычный int занимает меньше места, и если нет необходимости проводить над ним особые операции, компьютер будет работать. В свою очередь, с помощью класса-оболочки Integer можно выполнять специальные операции – например, перевести текст в число с помощью команды parseInt().

**Автоупаковка** и **автораспаковка –** это функция преобразования примитивных типов в объектные(классы оболочки) и наоборот.

**Автоупаковка применяется компилятором Java в следующих условиях:**

1. Когда значение примитивного типа передается в метод в качестве параметра метода, который ожидает объект соответствующего класса-оболочки
2. Когда значение примитивного типа присваивается переменной соответствующего класса-оболочки.

**Автораспаковка применяется компилятором в Java в следующих условиях:**

1. Когда объект передается в качестве параметра метода, который ожидает соответствующий примитивный тип.
2. Когда объект присваивается переменной соответствующего примитивного типа.

Автоупаковка и автораспаковка позволяют разработчику писать код, который легко читается и понятен.

Автоупаковка и распаковка выполняется при перегрузке метода на основании следующих правил:

1. Расширение "побеждает" упаковку в ситуации, когда становится выбор между расширением и упаковкой, расширение предпочтительней.
2. Расширение побеждает переменное количество аргументов в ситуации, когда становится выбор между расширением и переменным количеством аргументов, расширение предпочтительней.
3. Упаковка побеждает переменное количество аргументов в ситуации, когда становится выбор между упаковкой и переменным количеством аргументов, упаковка предпочтительней.

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

Потоки ввода-вывода: InputStream, OutputStream, Reader, Writer и тд.

Никакого движения данных по цепочке потоков нет, пока не начнем вызывать методы тпа read() или readLine().

**Байтовые потоки**

Это универсальные классы, поэтому они поддерживают самый распространённый тип данных – byte. В OutputStream можно записать байты, а из объекта InputStream можно читать байты. Все – никакие другие типы данных они не поддерживают. Особенность этих потоков в том, что данные из них можно читать только последовательно. Вы не можете прочитать данные из середины потока, не прочитав все данные перед ними. Именно так работает чтение клавиатуры через класс Scanner. Запись данных в поток OutputStream тоже происходит последовательно. Хороший пример том – вывод на экран.

**Символьные строки**

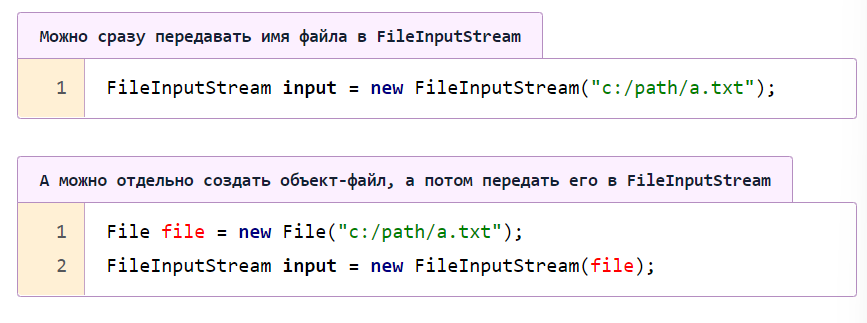
К ним относят классы Reader и Writer.

Класс Reader — это аналог класса InputStream, только его метод read() читает не байты, а символы — char. Класс Writer соответствует классу OutputStream, и так же, как и класс Reader, работает с символами (char), а не байтами.

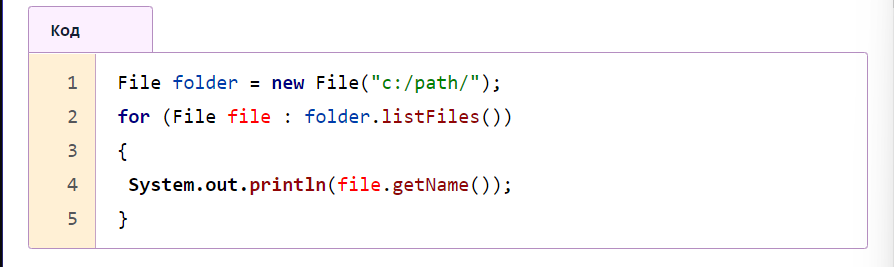
# Работа с файлами в Java. Класс java.io.File.

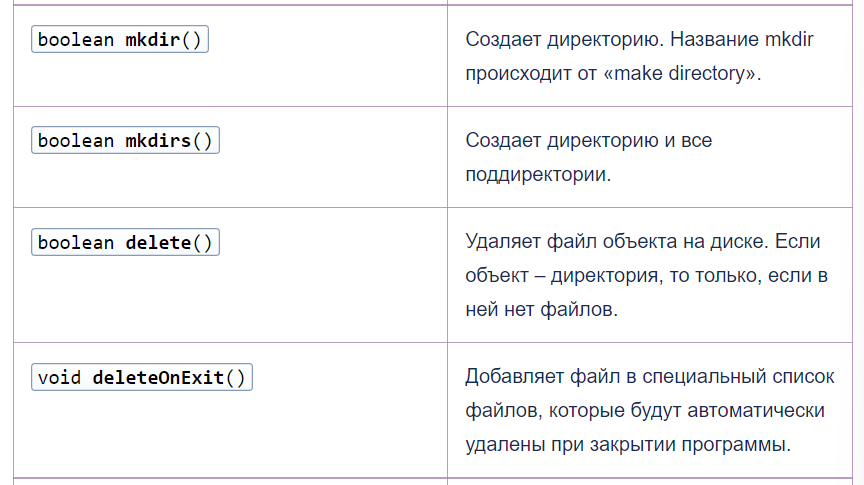
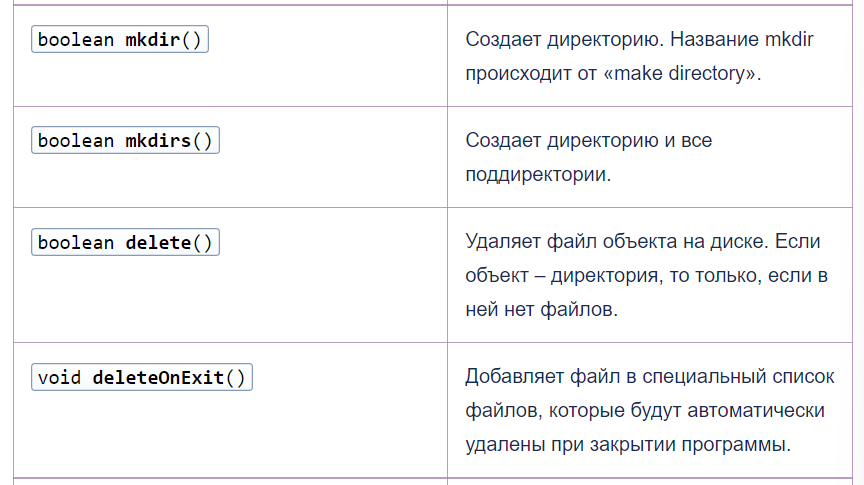
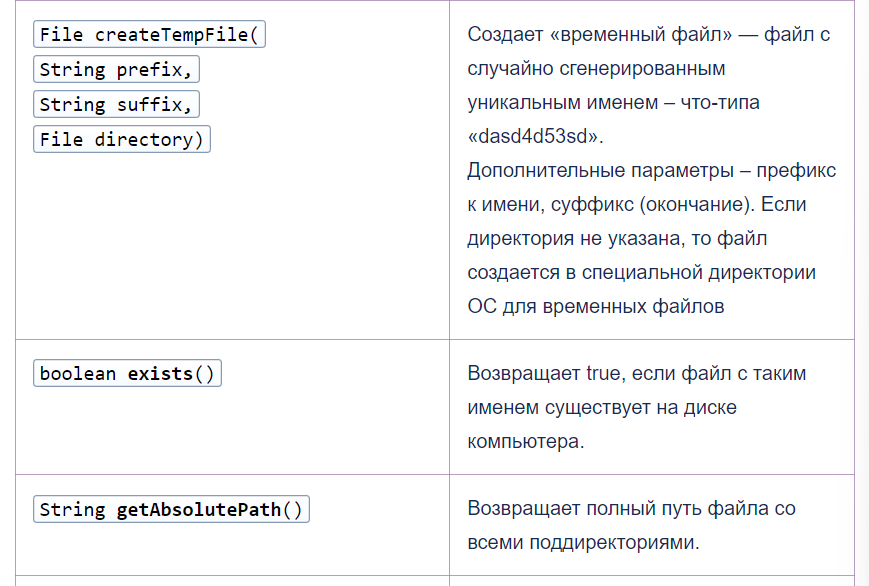
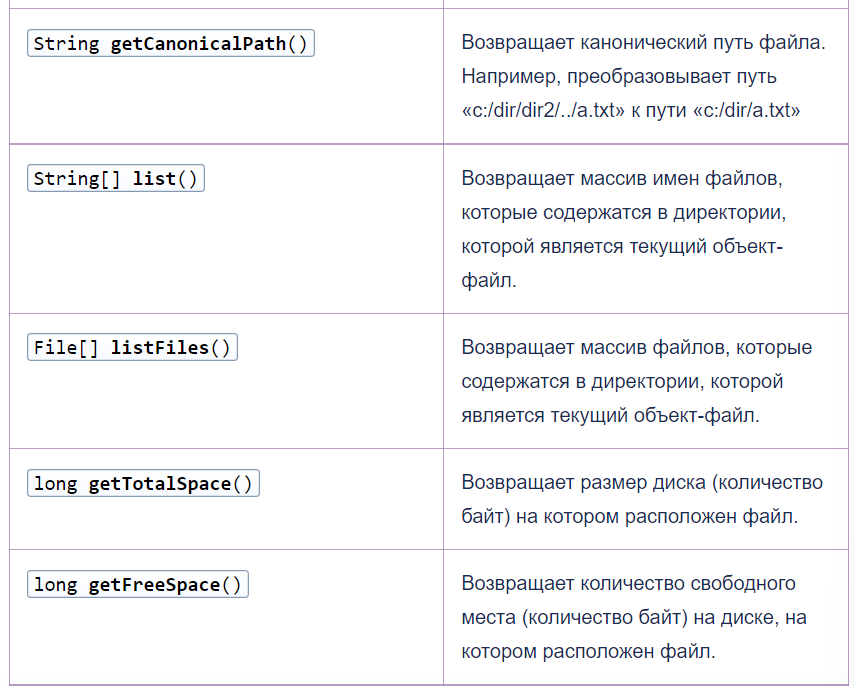
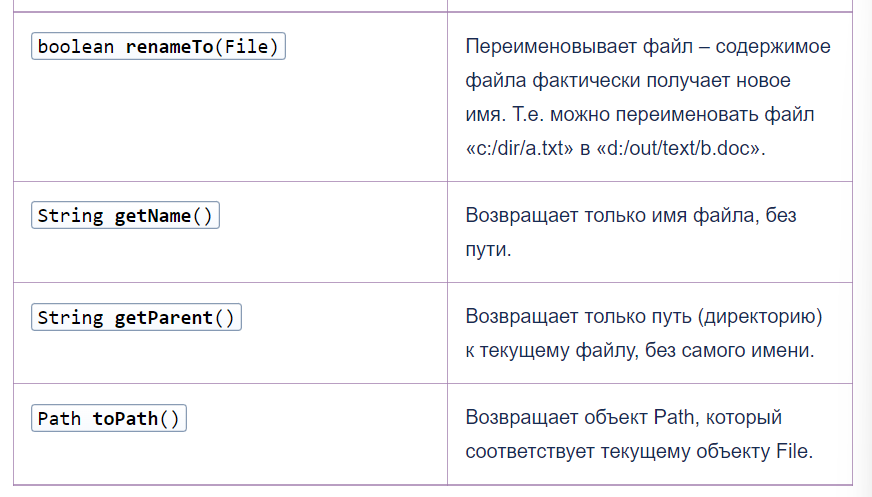
В Java есть специальный класс File, с помощью которого можно управлять фвйлами на диске компьютера. Для того, чтобы управлять содержимым фацла, есть другие классы: FileInputStream, FileOutputStream …

Почти во все классы такие можно передвать объекты класса File



listFiles() – метод, который возвращает список файлов в папке с заданным именем. Пример:



Вот остальные методы класса File: 8     

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

NIO – коллекция прикладных программных интерфейсов для языка Java, предназначенных для реализации высокопроизводительных операций ввода-вывода. Центральными абстракциями API-интерфейсов NIO являются:

* Буферы, которые являются контейнерами для данных;
* Кодировки и связанные с ними декодеры и кодировщики, которые переводят между байтами и символами Юникода;
* Каналы различных типов, которые представляют соединения с объектами, способными выполнять операции ввода-вывода; и
* Селекторы и ключи выбора, которые вместе с выбираемыми каналами определяют мультиплексированное, неблокирующее средство ввода-вывода.

Ключевое свойство nio – неблокирующий ввод-вывод.