Часть 1 (Апплеты)

Апплеты

Апплет — это (небольшая) прикладная программа, хранящаяся на веб- сервере, и запускающаяся при интерпретировании веб- документа.
Подключение апплетак веб- документу выполняется средствами языка разметки.
Апплеты исполняются в специально сформированной для них песочнице.
Апплеты имеют следующую структуру кода:

```
import java.awt.*             // какая- л/ граф. библиотека
import java.applet.*  // java.applet
public class ClassName extends Applet // класс инкапс. код апплета
{ public void FuncName ( … ) { … }}
```

апплеты традиционно оснащаются средствами GUI (и ввод- вывод в апплетах осуществляется средствами графических библиотек, без использования потоков ввода-вывода); кроме того, они используют возможности `java.applet` (и этого требует песочница).
Класс, инкапсуллирующий апплет, как и функция — точка входа в него, должен быть `public`, дабы быть доступным из песочницы. Функция — точка входа не должна называться `main()`.

Класс Applet

Методы класса Applet

`destroy()`

используется для уничтожения апплета; параметров не принимает; этот метод не делает ничего — он служит для переопределения (если вообще используется, и должен выполнять пользовательские действия (они производятся перед ничтожением апплета песочницей));
возвращает ничтоже;

`getAccessibleContext()`

используется для возврата контекста доступности вызывающего объекта; параметров не принимает; возвращает контекст доступности вызывающего объекта; возвращает объект `AccessibleContext`;

`getAppletcontext()`

используется для возврата контекста апплета; параметров не принимает; возвращает контекст апплета;
возвращает объект `AppletContext`;

`getAppletInfo()`

используется для получения информации об апплете; параметров не принимает; этот метод не делает ничего — он служит для переопределения (если вообще используется, и должен возвращать строку с информацией об апплете, в оригинальной версии метод возвращает значение `null`);
возвращает объект `String`;

`getAudioClip()`

используется для возврата указанного аудиоклипа; принимает параметры — адрес указанного аудиоклипа (в виде `URL`), либо адрес и строку (используетсякак имя аудиоклипа); возвращает указанный аудиоклип;
возвращает объект `AudioClip` (инкапсуллирует указанный аудиоклип);

`newAudioClip()`

используется для возврата указанного аудиоклипа; принимает параметры — адрес указанного аудиоклипа (в виде `URL`);
возвращает указанный аудиоклип; возвращает объект `AudioClip` (инкапсуллирует указанный аудиоклип);
этот метод — синоним метода `getAudioClip()`, за тем исключением, что принимает только один параметр, и является статическим и завершенным;

`getCodeBase()`

используется для возврата значения атрибута `codebase` соответствующего тега; параметров не принимает;
возвращает значение указанного атрибута; возвращает `URL`;

`getDocumentBase()`

используется для возврата значения адреса веб- документа, к которому подключен апплет; параметров не принимает; возвращает адрес веб- документа к которому подключен апплет;
возвращает `URL`;

`getImage()`

используется для возврата указанного изображения; принимает параметры — `URL` (целевого изображения), либо `URL` и строку (исполльзуется как имя целевого изображения);
возвращает указанное изображение;
возвращает объект `Image`;

`getLocale()`

используется для возврата объекта `Locale` (инкапсуллирует настройки локализации); параметров не принимает;
возвращает объект `Locale`; возвращает объект `Locale`;

`getParameter()`

используется для возврата параметра с указанным именем; принмает параметр — строку (используется как имя параметра); возвращает указанный параметр; возвращает строку;

`getParameterInfo()`

используется для возврата сведений о параметре; параметров не принимает; этот метод не делает ничего, он служит для последующего переопределения (если вообще используется, и должен возвращать массив строк (двумерный массив, он будет использоваться как таблица, каждаязапись таблицы должна состоять из трех строк - «имя параметра», «тип параметра, экстремумы значений», «комментирующие сведения»);
оригинальная версия метода возвращает значение `null`; возвращает массив строк;

`init()`

используется песочницей для запуска апплета; параметров не принимает; выполняет запуск апплета;
возвращает ничтоже;

`isActive()`

используется для проверки, активен ли апплет (исполняется ли он); параметров не принимает; проверяет, активен ли апплет;
возвращает логическое значение (`true` – активен, `false` – нет);

`isValidateRoot()`

используется для проверки, проверяет ли апплет корневой каталог; параметров не принимает; выполняет проверку, проверяет ли апплет корневой каталог; возвращает логическое значение (`true` – проверяет, `false` – нет);

`play()`

используется для воспроизведения аудиоклипа; принимает параметры — `URL` (аудиоклипа), либо `URL` и строку (используется как имя аудиоклипа); воспроизводит указанный аудиоклип; возвращает ничтоже;

`resize()`

используется для изменения размеров отображаемого апплета; принимает параметры — объект `Dimensions` (содержит целочисленные `DimensionsObjectName.width` и `DimensionsObjectName.height` – указывают размеры; класс `Dimensions` располагается в `java.awt`), либо число (целочисленное, ширина) и число (целочисленное, высота); руководствуясь переданными параметрами, изменяет габариты отображаемого апплета; возвращает ничтоже;

`setStub()`

используется для установки заглушки апплета (заглушка апплета — это объект `AppletStub`, он инкапсуллирет код, отвечающий за взаимодействие апплета с браузером); принимает параметр — объект заглушки апплета; устанавливает целевую заглушку апплета (этот метод вызывается самой песочницей, и не доступен для вызова программистом); возвращает ничтоже;
этот метод объявлен как завершенный;

`showStatus()`

используется для вывода в строке состояния браузера; принимает параметр — строку (она и будет выведена); выводит указанную строку в строке состояния браузера; возвращает ничтоже;

`start()`

используется для начала работы апплета (после инициализации апплета, либо при его возобновлении); параметровне принимает; выполняет старт апплета (этот метод вызывается самой песочницей) возвращает ничтоже;

`stop()`

используется для приостановки апплета; параметров не принимает; выполняет приостановку апплета (этот метод вызывается самой песочницей); возвращает ничтоже;
чтобы апплет был уничтожен, он сперва приостанавливается;

Переменные класса Applet

О них — в справочнике.

Прочие вопросы

прочие вопросы касательно апплетов — это уже вопрос написания апплетов, и поэтому в этих записях не рассматриваются.

Часть 2 (События)

События

При программиовании на Java события традиционно применяются в хоть сколь- бы то ни было сложном ПО (если речь идет о коммерческой программе, оснащенной GUI, то это ПО несомненно задействует события).
Поддержка событий реализована в следующих пакетах: java.event, java.awt, java.util.
Процесс обработки событий определяется так называемой «моделью делегирования событий». Модель, в общих чертах, предписывает следующее.
В рамках этой модели, события должны генерироваться и обрабатываться штатными для этого средствами (далее — «источниками» и «приемниками»). Таким образом, обработка событий делегируется этим средствам — за это модель и получила свое название.
Источник генерирует определенное событие (это объект, описывающий собственно событие), и извещает об этом целевой приемник (целевыхприемников может быть и несколько). По факту извещения, приемник обрабатывает это событие, и возвращает управление. Приемник становится целевым для источника по факту его регистрации источником в качестве целевого.
Событие целесообразно трактовать как объект, описывающий изменение состояния его источника. События могут наступать в результате различных процессов (действий пользователя с GUI, и прочих процессов). События могут быть языковыми и пользовательскими (разработчик языка считает, что поддержка пользовательских событий необходима, так как пользовательские события могут быть более подходящими для применения в каких- либо конкретных программах). Источником события является тоже объект (что логично для объектно- ориентированных языков), по изменению его состояния и возникает событие.
Приемники событий — это методы своих объектов; они имеют имя вида:
`addEventNameListener()`
эти методы объявлены как публичные (так как должны быть доступны), и возвращающие ничтоже (так как принято считать, что приемник должен делать всю работу сам, и не вообще делегировать ее доделывать (действительно, приемник и есть тот модуль, которому делегируется обработка события)). Эти методы принимают следующие параметры:
`(EventName ObjectName, … )`

источник событий поставляет метод снятия приемника с регистрации, это метод, имя которого имеет вид:
`removeEventNameListener()`

этот метод тоже публичный, и тоже возвращающий ничтоже — по тем же причинам. Область параметров имеет тот же вид - `(EventName ObjectName, … )`.
Строго говоря, приемник — это не сам метод, а объект, уведомляемыйо событии. Он должен быть зарегистрирован соответствующим методом источника;также, он должен задействовать средства пакетов, осуществляющих поддержку событий (расширять соответствующие интерфейсы; это интерфейсы с именами `EventName`, и представляющие специфические для события средства).
События, представленные в их классах, являются ядром механизма поддержки событий.
Класс `EventObject` – это вершина иерархии классов событий, он располагается в `java.util`. Этот класс имеет конструктор объектов (`EventObject( … )`, в параметрах прописывается объект- источник); и методы `getSource()` - возвращает объект- источник, и `toString()` - возвращает строковое представление своего объекта.
Класс `AWTEvent` (`java.awt`) является производным от `EventObject`, и вершиной иерархии классов событий библиотеки `AWT`. Метод `AWTEvent.getID()` позволяет запросить тип события.
Пакет `java.awt.event` содержит ряд классов событий, в частности следующие классы.

`ActionEvent`

это событие наступает по действию с GUI (клик по кнопке, двойной клик по элементу списка, клик по пункту меню);

`AdjustmentEvent`

это событие наступает при пользовании полосой прокрутки;

`ComponentEvent`

наступает при событиях, значимых для компонента (скрытие, перемещение, изменение габаритов, становление доступным);

`ContainerEvent`

это событие наступает по добавлению/ удалению элемента из контейнера;

`FocusEvent`

это событие наступает по изменению фокуса вода с клавиатуры;

`InputEvent`

это событие наступает по вводу данных в компонентах (а сам этот класс абстр.);

`ItemEvent`

это событие наступает по действию с элементами интерфейса, представленными множеством пунктов (флажок, элемент списка, пункт меню);

`KeyEvent`

это событие наступает по вводу с клавиатуры;

`MouseEvent`

это событие наступает при пользовании мышью (перевод указателя, клики);

`MouseWheelEvent`

это событие наступает при пользовании колесиком мыши (прокрутка);

`TextEvent`

это событие наступает по изменению текстовой области/ поля;

`WindowEvent`

это событие наступает по действиям, значимым для окна (изменение активности, изменение развернутости, и закрытии окна);

Класс ActionEvent

Конструктор класса ActionEvent

`ActionEvent()`

конструктор класса `ActionEvent`; принимает параметры — объект (источник), целочисленное (тип события), и строку (командная строка), либо то же и целочисленное (обозначает нажатие модифицирующих клавиш Alt, Ctrl, Shift), либо то же (что и в первом случае) и целочисленное (момент наступления

события) и целочисленное (обозначает нажатие модифицирующих клавиш); создает объект своего класса; возвращает ничтоже;

Методы класса ActionEvent

`getActionCommand()`

используется для возврата командной строки; параметров не принимает; возвращает командную строку своего события;

`getModifiers()`

используется для определения нажатия модифицирующих клавиш на момент наступления события; параметров не принимает; возвращает запрошенноые сведения; возвращает целочисленное;

`getWhen()`

используется для возврата момента наступления события; параметров не принимает; возвращает временную метку наступления своего события; возвращает целочисленное;

Переменные класса ActionEvent

`ALT_MASK`

константа; целочисленное; обозначает нажатие соответствующей клавиши;

`CTRL_MASK`

константа; целочисленное; обозначает нажатие соответствующей клавиши;

`SHIFT_MASK`

константа; целочисленное; обозначает нажатие соответствующей клавиши;

`META_MASK`

константа; целочисленное; обозначает нажатие соответствующей клавиши;

ACTION_PERFORMED

константа; целочисленное; обозначает нажатие соответствующей клавиши;

Класс AdjustmentEvent

Конструктор класса AdjustmentEvent

AdjustmentEvent()

конструктор класса AdjustmentEvent; принимает параметры — объект (источник), целочисленное (идентификатор события настройки), целочисленное (конкретный тип события настройки), и целочисленное (связанное с событием); создает объект своего класса; возвращает ничтоже;

Методы класса AdjustmentEvent

getAdjustable()

используется для возврата объекта- источника; параметров не принимает; возвращает объект- источник; возвращает объект;

getAdjustmentType()

используется для возврата типа события настройки; параметров не принимает; возвращает тип события настройки (соответствующую константу); возвращает целочисленное;

getValue()

используется для возврата значения события настройки; параметров не принимает; возвращает значение события настрйки (из соответствующей константы);
возвращает целочисленное;

Переменные класса AdjustmentEvent

BLOCK_INCREMENT

константа; целочисленное; обозначает щелчек по полосе прокрутки для увеличения прокручиваемого значения;

BLOCK_DECREMENT

константа; целочисленное; обозначает щелчек по полосе прокрутки для уменьшения прокручиваемого значения;

UNIT_INCREMENT

константа; целочисленное; обозначает щелчек по кнопке инкремента прокручиваемого значения;

UNIT_DECREMENT

константа; целочисленное; обозначает щелчек по кнопке декремента прокручиваемого значения;

TRACK

константа; целочисленное; обозначает перемещение ползунка;

ADJUSTMENY_VALUE_CHANGED

константа; целочисленное; обозначает изменение события;

Класс ComponentEvent

Конструктор класса ComponentEvent

ComponentEvent()

конструктор класса `ComponentEvent`; принимает параметры — объект (источник), и целочисленное (тип события); создает объект своего класса; возвращает ничтоже;
этот класс является суперклассом для классов `ContainerEvent`, `FocusEvent`, `KeyEvent`, `MouseEvent`, и `WindowEvent` (и ряда прочих);

Методы класса ComponentEvent

getComponent()

используется для возврата компонента (вызвавшего событие); параметров не принимает; возвращает компонент, вызвавший событие; возвращает объект;

Переменные класса ComponentEvent

COMPONENT_MOVED

целочисленное; константа; обозначает, что компонент перемещен;

COMPONENT_RESIZED

целочисленное; константа; обозначает, что компонент изменен в размерах;

COMPONENT_SHOWN

целочисленное; константа; обозначает, что компонент стал доступен;

COMPONENT_HIDDEN

целочисленное; константа; обозначает, что компонент сокрыт;

Класс ContainerEvent

Конструктор класса ContainerEvent

ContainerEvent()

конструктор класса `ContainerEvent`; принимает параметры — объект (источник), целочисленное (конкретный тип события), объект (компонент (удаляемый/ вводимый)); создает объект своего класса; возвращает ничтоже;

Методы класса ContainerEvent

getContainer()

используется для возврата контейнера; параметров не принимает; возвращает контейнер (к которому подключено это событие); возвращает объект;

getChild()

используется для возврата вложенного компонента контейнера; параметров не принимает; возвращает вложенный компонент своего контейнера (с которым произошло доступное ему событие); возвращает объект;

Переменные класса ContainerEvent

COMPONENT_ADDED

константа; целочисленное; обозначает одноименное себе событие;

COMPONENT_REMOVED

константа; целочисленное; обозначает одноименное себе событие;

Класс FocusEvent

Конструктор класса FocusEvent

FocusEvent()

конструктор класса `FocusEvent`; принимает параметры — объект (источник) и целочисленное (конкретный тип события), либо то же и логическое значение (показывает, есть ли еще фокус), либо то же (что и предыдущее) и объект (компонент, на который переходил фокус со своего компонента этого метода); создает объект своего класса; возвращает ничтоже;

Методы класса FocusEvent

getOppositeComponent()

используется для возврата компонента (на который переходил фокус); параметров не принимает; возвращает противоположный компонент (компонент, на который переходил фокус); возвращает объект;

isTemporary()

используется для определения, является ли временным переход фокуса; параметров не принимает; возвращает результат (логическое значение); возвращает логическое значение;

Переменные класса FocusEvent

FOCUS_GAINED

константа; целочисленное; обозначает наступление фокуса на своем объекте-компоненте;

`FOCUS_LOST`

константа; целочисленное; обозначает переход фокуса со своего объекта-компонента;

Класс InputEvent

Конструктор класса InputEvent

`InputEvent()`

строго говоря, этот конструктор не существует (так как класс абстрактный);

Методы класса InputEvent

это абстрактный класс, его методы малоинтересны;

Переменные класса InputEvent

Это абстрактный класс, его переменные малоинтересны.

Класс ItemEvent

Конструктор класса ItemEvent

`ItemEvent()`

констрктор класса `ItemEvent`; принимает параметры — объект (источник), целочисленное (конкретный тип события) объект (объект- элемент), и целочисленное (состояние события); руководствуясь переданными параметрами, создает объект своего класса;
возвращает ничтоже;

Методы класса ItemEvent

`getItem()`

используется для возврата ссылки на объект (элемент); параметров не принимает; возвращает ссылку на элемент;
возвращает объект;

`getItemSelectable()`

используется для возврата ссылки на объект (элемент); параметров не принимает; возвращает ссылку на элемент; возвращает объект;

`getStateChanged()`

используется для возврата состояния события; параметров не принимает; возвращает состояние события; возвращает целочисленное;

Переменные класса ItemEvent

`SELECTED`

константа; целочисленное; обозначает выбор определенного пункта;

`DESELECTED`

константа; целочисленное; обозначает отмену выбора;

`ITEM_STATE_CHANGED`

константа; целочисленное; обозначает состояние события;

Класс KeyEvent

Конструктор класса KeyEvent

`KeyEvent()`

конструктор класса `KeyEvent`; принимает параметры — объект (источник), целочисленное (конкретный тип события), целочисленное (время наступления события), целочисленное (клавиши- модификаторы), целочисленнное (код клавиши), символ (символ — для символьных клавиш); создает объект своего класса; возвращает ничтоже;

Методы класса KeyEvent

getKeyChar()

используется для возврата символа; параметров не принимает; возвращает символ (если несимвольная клавиша — то специальное значение `CHAR_UNDEFINIED`); возвращает символ;

getKeyCode()

используется для возврата кода клавиши; параметров не принимает; возвращает код клавиши (при нажатии символьной клавиши возвращает специальное значение `VK_UNDEFINIED`); возвращает целочисленное;

Прочие методы

прочие методы безынтересны;

Переменные класса KeyEvent

KEY_PRESSED

константа; целочисленное; обозначает одноименное событие; это событие несимвольных клавиш;

KEY_RELEASED

константа; целочисленное; обозначает одноименное событие; это событие несимвольных клавиш;

KEY_TYPED

константа; целочисленное; обозначает одноименное событие; это событие символьных клавиш;

Прочие переменные — несимвольные клавиши

это константы, целочисленные; это следующие константы:
`VK_ALT, VK_CONTROL, VK_SHIFT,`
`VK_ESCAPE, VK_ENTER,`
`VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, VK_PAGE_DOWN, VK_PAGE_UP;`

`Прочие переменные — символьные клавиши`

это константы, целочисленные; это следующие константы: `VK0_VK9`, `VKA-VKZ`;

Класс MouseEvent

Конструктор класса MouseEvent

`MouseEvent()`

конструктор класса `MouseEvent`; принимает параметры — объект (источник), целочисленное (конкретный тип события), целочисленное (время наступления события), целочисленное (клавиши модификаторы), целочисленное (координата курсора, абсцисса), целочисленное (координата курсора, ордината), целочисленное (количество щелчков), логическое значение (должно ли при событии всплывать меню), целочисленное (тип прокрутки), целочисленное (количество единиц прокрутки), целочисленное (количество единиц вращения); создает объект своего класса; возвращает ничтоже;

Методы класса MouseEvent

`getWheelRotation()`

используется для возврата количества единиц вращения колеса мыши; параметров не принимает; возвращает количество единиц вращений колеса мыши; возвращает целочисленное;

`getPreciseWheelRotation()`

используется для возврата количества единиц вращения колеса мыши; параметров не принимает;
возвращает количество единиц вращений колеса мыши; возвращает целочисленное;
этот метод предназначен для работы с мышью с высокой разрешающей способностью колеса;

`getScrollType()`

используется для возврата типа вращения; параметров не принимает; возвращает тип вращений колеса мыши (по строкам или экранам); возвращает целочисленное;

`getScrollAmount()`

используется для возврата количества единиц прокрутки;
параметров не принимает;
возвращает количество единиц прокрутки (этот метод доступен только при построчном типе прокрутки);
возвращает целочисленное;

Переменные класса MouseEvent

`WHEEL_BLOCK_SCROLL`

константа; целочисленное; обозначает прокрутку контента на страницу;

`WHEEL_UNIT_SCROLL`

константа; целочисленно; обозначает прокрутку контента на одну строку;

Класс TextEvent

Конструктор класса TextEvent

`TextEvent()`

конструктор класса `TextEvent`;
принимает параметры — объект (источник) и целочисленное (конкретный тип события);
создает объект своего класса;
возвращает ничтоже;

Методы класса TextEvent

их нет;

Переменные класса TextEvent

`TEXT_VALUE_CHANGED`

константа; целочисленное; обозначает изменение текста;

Класс WindowEvent

Конструктор класса WindowEvent

`WindowEvent()`

конструктор класса `WindowEvent`; принимает параметры — объект (источник), и целочисленное (конкретный тип события); создает объект своего класса; возвращает ничтоже;

Методы класса WindowEvent

`getWindow()`

используется для возврата источника события; параметров не принимает; возвращает объект- источник события; возвращает объект;

`getOppositeWindow()`

используется для возврата противоположного объекта; параметров не принимает; возвращает противоположный объект; возвращает объект;

`getOldState()`

используется для возврата прежнего состояния события; параметров не принимает; возвращает прежнее состояние события; возвращает целочисленное;

`getNewState()`

используется для возврата текущего состояния события; параметров не принимает; возвращает текущее состояние события; возвращает целочисленное;

Переменные класса WindowEvent

`WINDOW_ACTIVATED`

константа; целочисленное; обозначает одноименное действие;

`WINDOW_CLOSED`

константа; целочисленное; обозначает одноименное действие;

WINDOW_CLOSING

константа; целочисленное; обозначает одноименное действие;

WINDOW_DEACTIVATED

константа; целочисленное; обозначает одноименное действие;

WINDOW_DEICONIFIED

константа; целочисленное; обозначает одноименное действие;

WINDOW_GAINED_FOCUS

константа; целочисленное; обозначает одноименное действие;

WINDOW_LOST_FOCUS

константа; целочисленное; обозначает одноименное действие;

WINDOW_OPENED

константа; целочисленное; обозначает одноименное действие;

WINDOW_STATE_CHANGED

константа; целочисленное; обозначает одноименное действие;

Часть 3 (Ввод-вывод)

Ввод- вывод

Для организаци процессов ввода- вывода используются потоки ввода- вывода. Потоки ввода- вывода, как обычно, это абстракции, с помощью которых и организуются соответствующие процессы. Эти абстракции позволяют работать с устройствами ввода- вывода не вдаваясь в детали их организации.
Потоки ввода- вывода целесообразно подразделять на следующие их виды: потоки ввода- вывода байтов — используются для ввода- вывода данных двоичных файлов, и потоки ввода- вывода символов — используются для ввода-вывода данных в виде единичных символов в кодировке Unicode.
Как обычно, потоки ввода- вывода реализованы в классах. Эти классы лежат в `java.io`.

Потоки ввода- вывода байтов

Потоки ввода- вывода байтов реализованы в соответствующих классах. Вершины их иерархии — `InputStream` (потоки ввода), и `OutputStream` (потоки вывода). Эти классы — абстрактные, они инкапсуллируют конкретные классы для работы с конкретными устройствами. Абстрактные классы `InputStream` и `OutputStream` инкапсуллируют ряд абстрактных методов, конкретизируемых в их подклассах (в частности, методы `read()`, и `write()`).
Эти классы инкапсуллируют следующие потоки.

`BufferedInputStream`	Буферизованный поток ввода
`BufferedOutputStream`	Буферизованный поток вывода
`ByteArrayInputStream`	Поток ввода байтов из массива
`ByteArrayOutputStream`	Поток вывода байтов в массив
`DataInputStream`	Поток ввода данных (с методами чтения языковых типов)
`DataOutputStream`	Поток вывода данных (с методами записи языковых типов)
`FileInputStream`	Поток ввода из файла
`FileOutputStream`	Поток вывода в файл
`FilterInputStream`	Поток ввода (реал. `InputStream`)
`FilterOutputStream`	Поток вывола (реал. `InputStream`)
`ObjectInputStream`	Поток ввода объектов
`ObjectOutputStream`	Поток вывода объектов
`PipedInputStream`	Поток ввода (канал)
`PipedOutputStream`	Поток вывода (канал)
`PrintStream`	Поток вывода (методы print(), и println())
`PushbackInputStream`	Поток ввода (поддержка возврата байта в поток)
`SequenceInputStream`	Поток ввода (ряд потоков, используемых в порядке очереди)

Потоки ввода- вывода символов

Потоки ввода- вывода символов реализованы в соответствующих классах. Вершины их иерархии — `Reader` (потоки ввода), и `Writer` (потоки вывода). Эти классы — абстрактные, они инкапсуллируют конкретные классы для работы с конкретными устройствами ввода- вывода. Абстрактные классы `Reader` и `Writer` инкапсуллируют ряд абстрактных методов, конкретизируемых в их подклассах (в частности, методы `read()`, и `write()`).
Эти классы инкапсуллируют следующие потоки.

`BufferedReader`	Буферизованный поток ввода
`BufferedWriter`	Буферизованный поток вывода
`CharArrayReader`	Поток ввода символов из массива
`CharArrayWriter`	Поток вывода символов в массив
`FileReader`	Поток ввода символов из файла
`FileWriter`	Поток вывода символов в файл
`FilterReader`	Поток ввода фильтрующий
`FilterWriter`	Поток вывода фильтрующий
`InputStreamReader`	Поток ввода (преобразует байты в символы)
`OutputStreamWriter`	Поток вывода (преобразует символы в байты)
`LineNumberReader`	Поток ввода (подсчитывает строки)
`PipedReader`	Поток ввода (канал)
`PipedWriter`	Поток вывода (канал)
`PrintWriter`	Поток вывода (методы print(), и println())
`PushbackReader`	Поток ввода (поддержка возврата байта в поток)
`StringReader`	Поток ввода (читает символы из строки)
`StringWriter`	Поток вывода (записывает символы в строку)

Предопределенные потоки ввода- вывода

Предопределенные потоки ввода- вывода — `in` (стандартный поток ввода, по умолчанию — клавиатура), `out` (стандартный поток вывода, по умолчанию - консоль), `err` (стандартный поток вывода ошибок, по умолчанию — консоль), инкапсуллированы в классе `java.lang.System`. Они объявлены как `public static final`. Эти потоки могут быть перенаправлены. Они инкапсуллируют объекты классов `InputSream`, `PrintStream`, и опять- таки `PrintStream`, соответственно.

Класс File

Для работы с файлами файловой системы используется класс `File`. Класс `File` оперирует не с потоками ввода- вывода, а с самими файлами. Файлы и каталоги являются объектами этого класса (именно его, даже не подклассов).

Конструктор класса File

`File()`

конструктор класса `File`; принимает параметры — значение (строка — путь к файлу, или объект — ссылка на файл, этот параметр указывает каталог создаваемого файла), и значение (строка — имя файла, этот параметр можно опустить — если создается каталог); создает объект своего класса; возвращает ничтоже;

Методы класса File

`exists()`

используется для проверки файла на существоване; параметров не принимает; проверяет, существует ли файл; возвращает логическое значение (`true` – да, `false` – нет);

`getAbsolutePath()`

используется для возврата абсолютного пути файла; параметров не принимает; возвращает абсолютный путь своего файла (включая имя своего файла); возвращает строку;
абсолютный путь — это путь относительно корня диска (для сравнения, относительный путь — это путь относительно какого- либо каталога по умолчанию);

`getPath()`

используется для возврата пути файла; параметров не принимает; возвращает путь своего файла (включая имя своего файла); возвращает строку;

`getParent()`

используется для возврата родительского каталога; параметров не принимает; возвращает родительский (вмещающий) каталог своего файла; возвращает строку;

`getName()`

используется для возврата имени файла; параметров не принимает; возвращает имя (собственно имя) своего файла; возвращает строку;

`canRead()`

используется для проверки доступности по чтению; параметров не принимает; проверяет, доступен ли свой файл по чтению; возвращает логическое значение (`true` – да, `false` – нет);

`canWrite()`

используется для проверки доступности по записи; параметров не принимает; проверяет, доступен ли свой файл по записи; возвращает логическое значение (`true` – да, `false` – нет);

`isFile()`

используется для проверки является ли совй объект файлом; параметров не принимает; проверяет, является ли свой объект файлом (не каталогом); возвращает логическое значение (`true` – да, `false` – нет (либо это служебный файл JDK));

`isDirectory()`

используется для проверки является ли свой объект каталогом; параметров не принимает; проверяет, является ли свой объект каталогом; возвращает логическое значение (`true` – да, `false` – нет);

`list()`

используется для возврата списка файлов каталога; параметров не принимает; возвращает список файлов своего объекта- каталога (возвращает список файлов непосредственно в своем объекте- каталоге, и не производит рекурсивного обращения к содержимому вложенных каталогов); возвращает массив строк;
этот метод имеет специфический перегруженный вариант, он принимает объект пользовательского класса (реализующий интерфейс `FilenameFilter`), возвращает список файлов отфильтрованным (по пользовательскому условию), возвращает массив строк; объект пользовательского класса следует создавать определенным образом (он создается обычной для этого записью; но необходимо помнить — он должен реализовать интерфейс `FilenameFilter`; этот интерфейс имеет единственный метод — `accept()` - этот метод, будучи определен в классе, будет автоматически вызываться самим интерпретатором для каждого пункта списка метода `list()`; метод `accept()` должен определяться со следующими условиями — он должен возвращать логическое значение (истину для включаемого в результат фильтрации файла), принимать параметры — ссылку на объект- каталог и строку (в ней методу будет передаваться имя файла из списка), и оперировать с пользовательским

шаблоном соответствия; пользовательскому классу следует задать конструктор — чтобы объект создавался со значением (шаблон соответствия), и метод `accept()` в своем теле оперировал с ним как со значением, известным по факту создания объекта (так или иначе, применяя пользовательский шаблон соответствия к имени файла));

`listFiles()`

используется как альтернатива методу `list()`; параметров не принимает; возвращает список файлов своего объекта- каталога (возвращает список файлов непосредственно в своем объекте- каталоге, и не производит рекурсивного обращения к содержимому вложенных каталогов); возвращает массив объектов-файлов;
перегруженнная версия используется так же, как перегруженная версия метода `list()`, но возвращает массив файлов;

`isAbsolute()`

используется для проверки, есть ли у файла абсолютный путь; параметров не принимает; проверяет, есть ли у своего файла абсолютный путь;
возвращает логическое значение (`true` – да, `false` – нет);

`renameTo()`

используется для переименования файла; принимает параметр — строку (должна содержать новое имя файла, переименовать старым именем нельзя); выполняет переименование своего файла (если указанным именем можно переименовать);
возвращает логическое значение (`true` – переименован, `false` – нельзя);

`delete()`

используется для удаления файла;
параметров не принимает;
удаляет свой файл (если это — каталог, то он удаляется если только пуст);
возвращает логическое значение (`true` – удален, `false` – нет);

`getTotalSpace()`

используется для возврата емкости тома;
параметров не принимает;
возвращает емкость тома, хранящего свой файл этого метода;
возвращает `long`;

`getFreeSpace()`

используется для возврата емкости свободного пространства тома; параметров не принимает; возвращает емкость свободного простанства тома, хранящего свой файл этого метода;
возвращает `long`;

`getUsableSpace()`

используется для возврата полезной емкости свободного пространства тома; параметров не принимает; возвращает полезную емкость свободного простанства тома, хранящего свой файл этого метода;
возвращает `long`;

`deleteOnExit()`

используется для удаления файла; параметров не принимает; удаляет свой файл при завершении работы JVM;
возвращает ничтоже;

`isHidden()`

используется для проверки, является ли файл скрытым; параметров не принимает; проверяет, является ли свой файл скрытым;
возвращает логическое значение (`true` – да, `false` – нет);

`setLastModified()`

используется для установки временной метки; принимает параметр — `long` (временная метка, в миллисекундах); устанавливает указанную временную метку своему файлу;
возвращает логическое значение (`true` – `Ok`, `false` – нет);

`setReadOnly()`

используется для установки аттрибута файла «только для чтения»; параметров не принимает; устанавлвает одноименный аттрибут своего файла;
возвращает логическое значение (`true` – `Ok`, `false` – нет);

`Прочие методы`

прочие методы малоинтересны;

Закрытие потоков ввода- вывода

По завершении использования потока ввода- вывода, он должен быть закрыт. Если это не будет сделано, то используемые потоком ресурсы не будут высвобождены (произойдет фактическая утечка ресурсов).
Традиционный способ закрытия потока заключается в следующем — по завершении использования потока выбрасывается прерывание, это прерывание перехватывается, и в дальнейшем происходит закрытие потока (вызывается метод потока `close()`, обычно это производится в блоке `finally`).
Более современный подход закрытия потока заключается в следующем — по завершении использования потока (по факту завершения блока `try`) этот поток завершается автоматически. Для этого используется специфическая форма записи - «`try` с ресурсами». Эта форма записи такова:

```
try () {        // в () прописывается ресурс
 …              // код блока, ресурс в нем явно не закрывается
}               // конец блока
```

в этой записи в `()` прописывается `ClassName ObjectName = new ObjectName`, тем самым указывается ресурс (файл, или прочее), подлежащий автоматическому закрытию.
При использовании «`try` с ресурсами» необходимо помнить следующее. Ресурсы, допускающие автоматическое закрытие, это объекты классов, реализующих интерфейс `AutoCloseable`. Явное закрытие ресурса не применяется. Можно указать и несколько автоматически закрываемых ресурсов — тогда записи в `()` указываются просто через запятую.
О создании потоков ввода- вывода
Потоки ввода- вывода создаются как конкретные (не абстрактные) классы потоков. Эти потоки создаются специфическими конструкторами, и результирующие объекты потоков обладают соответствующими методами. Эти результирующие объекты- потоки наследуют и переопределяют методы своих абстрактных суперклассов (необходимо помнить, что эти методы переопределяются, и поэтому обладают своей спецификой), и вводят свои собственные специфические методы. Подробнее о всех этих методах — в справочнике. Подробнее о конструкторах конкретных классов потоков — также в справочнике. Все эти методы и конструкторы идентичны (так как потоки, по своей сути, решают сходные задачи), однако в деталях имеются различия — таким образом, чтобы уточнить детали, целесообразней пользоваться готовыми справочниками, нежели такми записями, как `JavaII`.

NIO

Система ввода- вывода NIO вводит свои новшества. Это буферы и каналы. Буферы предназначены для буферного хранения данных. Каналы предназначены для прямого взаимодействия с устройством. Система NIO используется таким образом — устанавливается канал, затем обмен данными ведется через буфер.

Эта система реализована в пакете `java.nio`.

Буферы

Буферы являются производными от класса `Buffer`.
Все буферы характеризуются следующими параметрами — текущая позиция, предел, емкость. Текущая позиция — это позиция (индекс) в буфере, с которой стартует (очередная) операция с данными в буфере; разумеется, эта позиция не константна в процессе использования буфера. Предел — это размер буфера (в его элементах); строго говоря, это индекс за последней доступной ячейкой буфера. Емкость — это размер буфера (в его элементах); обычно это то же число, что и предел.
Прочие сведения — в справочнике.

Каналы

Каналы являются классами, реализующими интерфейс `Channel` (наследует от интерфейса `AutoClosable`).
Прочие сведения — в справочнике.

Часть 4 (Коллекции)

Коллекции

Коллекции — это собственно коллекции элементов.
Элементы коллекции инкапсуллированы в объект. Различные коллекции предназначены для хранения элементов разных типов, и предоставляют различные методы для работы с ними.
Средство языка, внедряющее в него коллекции — Collection Framework.
Средства этого фреймворка располагаются в пакете `java.util` (этот пакет должен быть импортирован явно). Коллекции, с точки зрения реализации, представляют собой ряд классов (и реализуемых ими интерфейсов) вышеобозначенного пакета. Используя эти классы и интерфейсы, можно объявить и пользовательские классы с необходимой функциональностью.
Языковые классы коллекций будут рассмотрены далее.
Функциональность этих классов зависит от их базовых классов и реализуемых ими интерфейсов. Классы (абстрактные классы, находящиеся в верху иерархии языковых классов) реализуют следующие интерфейсы (один или несколько из них): `Collection`, `Queue`, `Deque`, `List`, `Set`, `SortedSet`, `NavigableSet`, `Map`, `SortedMap`, `NavigableMap`. Эти интерфейсы реализуются следующими абстрактными классами: `AbstractCollection`, `AbstractList`, `AbstractSet`, `AbstractQueue`, `AbstractSequentalList`, `AbstractMap`.

Интерфейс Collection

Это обобщенный интерфейс, наследующий от `Iterable`. Его реализуют все коллекции. В нем имеются следующие методы.

`add()`

используется для добавления элемента в коллекцию; принимает параметр — элемент коллекции; добавляет указанный элемент в свою коллекцию; возвращает логическое значение (`true` - добавлен, `false` - нет);

`addAll()`

используется для добавления элементов коллекции в свою коллекцию; принимает параметр — объект коллекции; добавляет элементы указанной коллекции в свою коллекцию; возвращает логическое значение (`true` - добавлен, `false` - нет);

`clear()`

используется для очистки своей коллекции; параметров не принимает; удаляет все элементы из своей коллекции; возвращает ничтоже;

`contains()`

используется для определения, присутствует ли в коллекции заданный элемент; принимает параметр — элемент коллекции; определяет, есть ли в своей коллекции указанный элемент; возвращает логическое значение (`true` – да, `false` - нет);

`isEmpty()`

используется для проверки, пуста ли своя коллекция; параметров не принимает; проверяет, пуста ли своя коллекция; возвращает логическое значение (`true` – да, `false` - нет);

`iterator()`

используется для возврата итератора; параметров не принимает; возвращает итератор для обхода своей коллекции;
возвращает объект- итератор;

`remove()`

используется для удаления элемента из коллекции; принимает параметр — элемент коллекции; удаляет указанный элемент из своей коллекции;
возвращает логическое значение (`true` – удален, `false` – нет);

`removaAll()`

используется для удаления элементов из коллекции; параметров не принимает; удаляет все элементы из своей коллекции;
возвращает логическое значение (`true` – удалены, `false` – нет);

`retainAll()`

используется для удаления элементов из коллекции; принимает параметр — объект коллекции;
удаляет все элементы из своей коллекции, за исключением элементов, присущих и указанной коллекции;
возвращает логическое значение (`true` – удалены, `false` – нет);

`size()`

используется для возврата размера коллекции; параметров не принимает;возвращает размер своей коллекции в элементах; возвращает `int`;

`toArray()`

используется для преобразования своей коллекции к массиву; параметров не принимает; преобразет свою коллекцию к массиву; возвращает массив;
Интерфейс Queue
Этот интерфейснаследует от интерфейса Collection.
Он вводит следуюшие методы.

`element()`

используется для возврата элемента очереди; параметров не принимает; возвращает (не выталкивая) элемент своей очереди; возвращает элемент;

`offer()`

используется для добавки элемента в конец очереди; принимает параметр — элемент очереди; добавляет указанный элементвконец своей очереди; возвращает логическое значение (`true` – добавлен, `false` – нет);

`peek()`

используется для возврата элемента из начала очереди; параметров не принимает; возвращает (без выталкивания) элемент из начала очереди; возвращает элемент;

`poll()`

используется для выталкивания элемента из начала очереди; параметров не принимает; выталкивает элемент из начала очереди; возвращает элемент;

`remove()`

используется для выталкивания элемента из начала очереди; параметров не принимает; выталкивает элемент из начала очереди; возвращает элемент;

Интерфейс Queue

Этот интерфейс расширяет наследует от интерфейса `Collection`, и вводит следующие методы.

`element()`

используется для возврата элемента очереди; параметров не принимает; возвращает (без выталкивания) элемент из головы своей очереди;
возвращает элемент;

`offer()`

используется для добавления элемента в очередь; принимает параметр — элемент очереди;
добавляет указанный элемент в свою очередь;
возвращает логическое значение (`true` – added, `false` – no);

`peek()`

используется для возврата элемента очереди; параметров не принимает; возвращает (без выталкивания) элемент из головы своей очереди;
возвращает элемент;

Интерфейс Deque

Он наследует от интерфейса `Queue`.
Он вводит следующие методы.

`addFirst()`

используется для добавления элемента в начало очереди; принимает параметр — элемент очереди;
добавляет указанный элемент в начало воей очереди;
возвращает ничтоже;

`addLast()`

используется для добавления элемента в конец очереди; принимает параметр — элемент очереди;
добавляет указанный элемент в конец воей очереди;
возвращает ничтоже;

`getFirst()`

используется для получения элемента из начала очереди; параметров не принимает; возвращает (без выталкивания) элемент из начала своей очереди; возвращает элемент;

`getLast()`

используется для получения элемента из конца очереди; параметров не принимает; возвращает (без выталкивания) элемент из конца своей очереди; возвращает элемент;

`offerFirst()`

используется для добавления элемента в начало очереди; принимает параметр — элемент очереди; добавляет указанный элемент в начало своей очереди; возвращает логическое значение (`true` – добавлен, `false` – нет);

`offerLast()`

используется для добавления элемента в конец очереди; принимает параметр — элемент очереди; добавляет указанный элемент в конец своей очереди; возвращает логическое значение (`true` – добавлен, `false` – нет);

`peekFirst()`

используется для получения элемента из начала очереди; параметров не принимает; возвращает (без выталкивания) элемент из начала своей очереди; возвращает элемент;

`peekLast()`

используется для получения элемента из конца очереди; параметров не принимает;
возвращает (без выталкивания) элемент из конца своей очереди;
возвращает элемент;

`pollFirst()`

используется для получения элемента из начала очереди; параметров не принимает;
возвращает (с выталкиванием) элемент из начала своей очереди;
возвращает элемент;

`pollLast()`

используется для получения элемента из конца очереди; параметров не принимает; возвращает (с выталкиванием) элемент из конца своей очереди; возвращает элемент;

`pop()`

используется для выталкивания элемента из начала очереди; параметров не принимает; выталкивает элемент из начала своей очереди; возвращает эл- т;

`push()`

используется для заталкивания элемента в начало очереди; принимает параметр — элемент очереди; заталкивает элемент вначало своей очереди; возвращает ничтоже;

`removeFirst()`

используется для выталкивания элемента из начала своей очереди; параметров не принимает; выталкивает элемент из начала своей очереди; возвращает элемент;

`removeLast()`

используется для выталкивания элемента из конца своей очереди; параметров не принимает; выталкивает элемент из конца своей очереди; возвращает элемент;

`removeFirstOccurence()`

используется для выталкивания первого вхождения указанного элемента из очереди; принимает параметр — элемент очереди; выталкивает первое вхождение указанного элемента из своей очереди; возвращает логическое значение (`true` – удален, `false` - нет);

`removeLastOccurence()`

используется для выталкивания последнего вхождения указанного элемента из очереди; принимает параметр — элемент очереди; выталкивает последнее вхождение указанного элемента из своей очереди; возвращает логическое значение (`true` – удален, `false` - нет);

Интерфейс List

Этот интерфейс содержит следующие методы.

`add()`

используется для добавки элемента в коллекцию; принимает параметры — индекс позиции вставки и объект — элемент коллекции; руководствуясь переданными параметрами, добавляет элемент в свою коллекцию; возвращает ничтоже;

`addAll()`

используется для добавки элементов в коллекцию; принимает параметры — индекс позиции вставки и объект коллекции; руководствуясь переданными параметрами, добавляет элементы в свою коллекцию; возвращает ничтоже;

`get()`

используется для получения элемента коллекции; принимает параметр — индекс элемента; возвращает элемент по указанному индексу; возвращает элемент коллекции;

`indexOf()`

используется для возврата индекса элемента; принимает параметр — элемент коллекции; возвращает индекс первого вхождения указанного элемента в свою коллекцию;
возвращает индекс элемента;

`lastIndexOf()`

используется для возврата индекса элемента; принимает параметр — элемент коллекции;
возвращает индекс последнего вхождения указанного элемента в свою коллекцию;
возвращает индекс элемента;

`listIterator()`

используется для возврата итератора; параметров не принимает;
возвращает объект- итератор для обхода своей коллекции;
возвращает объект;

`of()`

используется для создания списка из указанных элементов; принимает параметры — ряд объектов — элементов коллекции; создает список из указанных элементов; возвращает объект- список;

`remove()`

используется для удаленияэлемента коллекции; принимает параметр — индекс элемента; удаляет указанный элемент из своей коллекции; возвращает удаленный элемент;

`set()`

используется для инициализации элемента коллекции; принимает параметры — индекс элемента и объект- элемент (содержит значение инициализации); руководствуясь переданными параметрами, инициализирует элемент своей коллекции; возвращает ничтоже;

`sort()`

используется для сортировки коллекции; принимает параметр — объект компаратор; руководствуясь переданными параметрами, сортирует свою коллекцию; возвращает ничтоже;

`subList()`

используется для получения подсписка списка элементов; принимает параметры — начальная позиция элемента и конечная позиция элемента; руководствуясь переданными параметрами, создает подсписок списка элементов; возвращает объект-список;

Интерфейс Set

Этот интерфейс наследует от интерфейса `Collection`, и переопределяет его методы. Новых методов не вводит.

Интерфейс SortedSet()

Этот интерфейс наследует от интерфейса `Set`. Он вводит следующие методы.

`first()`

используется для возврата первого элемента множества; параметров не принимает; возвращает первый элемент множества; возвращает элемент;

`last()`

используется для возврата последнего элемента множества; параметров не принимает; возвращает последний элемент множества; возвращает элемент;

`headSet()`

используется для возврата подмножества своего множества; принимает параметр — элемент множества; возвращает подмножество своего множества, до указанного элемента; возвращает объект `SortedSet`;

`subSet()`

используется для возврата подмножества своего множества; принимает параметры — элемент множества (стартовый) и элемент множества (финальный); возвращает подмножество своего множества, от стартового до финального элемента; возвращает объект `SortedSet`;

`tailSet()`

используется для возврата подмножества своего множества; принимает параметр -элемент множества; возвращает подмножество своего множества, с указанного элемента; возвращает объект `SortedSet`;

Интерфейс NavigableSet

Этот интерфейс наследует от интерфейса `SortedSet`. Он вводит следующие методы.

`ceiling()`

используется для возврата элемента из множества; принимает параметр — элемент (указанный элемент); возвращает элемент множества, больший указанного элемента; возвращает элемент множества;

этот метод возвращает наименьший из удовлетворяющих условию элементов;
сравнение приозводит оператором >=;
если подходящий элемент не найден, то возвращает null;

`floor()`

используется для возврата элемента из множества; принимает параметр — элемент (указанный элемент); возвращает элемент множества, меньший указанного элемента; возвращает элемент множества;
этот метод возвращает наибольший из удовлетворяющих условию элементов;
сравнение приозводит оператором <=;
если подходящий элемент не найден, то возвращает null;

`higher()`

используется для возврата элемента из множества; принимает параметр — элемент (указанный элемент); возвращает элемент множества, больший указанного элемента; возвращает элемент множества;
этот метод возвращает наименьший из удовлетворяющих условию элементов;
сравнение приозводит оператором >;
если подходящий элемент не найден, то возвращает null;

`lower()`

используется для возврата элемента из множества; принимает параметр — элемент (указанный элемент);
возвращает элемент множества, меньший указанного элемента; возвращает элемент множества;
этот метод возвращает наибольший из удовлетворяющих условию элементов;
сравнение приозводит оператором <;
если подходящий элемент не найден, то возвращает null;

`pollFirst()`

используется для выталкивания элемента; параметров не принимает; выталкивает первый элемент из множества;
возвращает элемент;

`pollLast()`

используется для выталкивания элемента; параметров не принимает; выталкивает последний элемент из множества;
возвращает элемент;

`descendingSet()`

ипользуется для получения множества; параметров не принимает; возвращает множество, порядок элементов в нем — обратный; возвращает `NavigableSet`;

`headSet()`

используется для получения множества; принимает параметры — элемент множества (граница) и логическое значение (`true` – по границу включительно, `false` – не включительно); возвращает множество, по указанную границу; возвращает `NavigableSet`;

`subSet()`

используется для получения множества; принимает параметры — элемент (граница, нижняя), логическое значение (`true` – по нижнюю границу включительно, `false` – не включительно), элемент (граница, верхняя), и логическое значение (`true` – по верхнюю границу включительно, `false` – не включительно); руководствуясь переданными параметрами, возвращает подмножество своего множества; возвращает `NavigableSet`;

`tailSet()`

используется для получения множества; принимает параметры — элемент множества (граница) и логическое значение (`true` – с границы включительно, `false` – не включительно); возвращает множество, с указанной границы; возвращает `NavigableSet`;

Интерфейс Map

Этот интерфейс используется для работы с отображениями. Отображение — это слоарь, элементы словаря представляют собой пару ключ- значение. Словари используются для быстрого поиска по ним.
Этот интерфейс не наследует от интерфейса `Collection`.
Инкапсуллирует интерфейс `Map.Entry`.

Методы интерфейса Map

`clear()`

используется для очистки отображения; параметров не принимает; очищает свое отображение от всех элементов; возвращает ничтоже;

`containsKey()`

используется для проверки, содержит ли отображение ключ; принимает параметр — объект- ключ; проверяет, содержит ли свое отображение указанный ключ; возвращает логическое значение (`true` – yes, `false` – no);

`containsValue()`

используется для проверки, содержит ли отображение значение; принимает параметр — объект- значение; проверяет, содержит ли свое отображение указанное значение; возвращает логическое значение (`true` – yes, `false` – no);

`entrySet()`

используется для возврата элементов отображения; параметров не принимает; возвращает множество элементов отображения; возвращает `Set`;

`equals()`

используется для проверки, идентичности коллекций; принимает параметр — объект коллекции; проверяет, идентична ли своя коллекция заданной; возвращает логическое значение (`true` – yes, `false` – no);

`isEmpty()`

используется для проверки, пусто ли свое отображение; параметров не принимает; проверяет, пусто ли свое отображение; возвращает логическое значение (`true` – yes, `false` – no);

`get()`

используется для получения значения; принимает параметр — объект- ключ; возвращает значение по указанному ключу;
возвращает объект- значение;

`getOrDefault()`

используется для получения значения; принимает параметры — объект- ключ и объект- значение (по умолчанию);
возвращает значение по указанному ключу (если не найдено — то объект-значение по умолчанию);
возвращает объект- значение;

`put()`

используется для добавления значения в отображение; принимает параметры — объект- ключ и объект- значение; руководствуясь переданными параметрами, добавляет значение в отображение; возвращает переданный объект- ключ; объект- ключ возвращается, если была перезапись уже существующего значения, иначе — `null`;

`putIfAbsent()`

используется для добавления значения в отображение; принимает параметры — объект- ключ и объект- значение; руководствуясь переданными параметрами, добавляет значение в отображение, если это не приведет к перезаписи уже существующего значения; возвращает переданный объект- ключ;

`keySet()`

используется для возврата ключей отображения; параметров не принимает; возвращает ключи отображения; возвращает массив объектов- ключей;

`values()`

используется для возврата значений отображения; параметров не принимает; возвращает значения отображения; возвращает массив объектов- значений;

`putAll()`

используется для добавления элементов в отображение; принимает параметр — объект- отображение; добавляет элементы указанного отображения в свое отображение; возвращает ничтоже;

`remove()`

используется для удаления элемента отображения; принимает параметр — объект- ключ; удаляет указанный элемент из своего отображения; возвращает объект- значение (удаленного элемента);

`size()`

используется для возврата размера отображения; параметров не принимает; возвращает размер своего отображения; возвращает `int`;

Методы интерфейса Map.Entry

equals()

описан выше;

getKey()

используется для возврата ключа; параметров не принимает; возвращает ключ объекта- отображения; возвращает объект- ключ;

getValue()

используется для возврата значения; параметров не принимает; возвращает значения объекта- отображения; возвращает объект- значение;

setValue()

используется для установки значения; принимает параметр — объект- значение; устанавливает значение объекта- отображения; возвращает объект- значение;

hashCode()

используется для возврата хэш- кода отображения; параметров не принимает; возвращает хэш своего отображения; возвращает `int`;

Интерфейс SortedMap

Этот интерфейс наследует от интерфейса `Map`.
Он вводит следующие методы.

firstKey()

используется для возврата ключа отображения; параметров не принимает; возвращает первый ключ своего отображения;
возвращает объект- ключ;

lastKey()

используется для возврата ключа отображения; параметров не принимает; возвращает последний ключ своего отображения;
возвращает объект- ключ;

`headMap()`

используется для возврата части отображения; принимает параметр — объект-ключ; возвращает часть своего отображения, до указанного ключа (включительно); возвращает отображение;

`subMap()`

используется для возврата части отображения; принимает параметры — объект-ключ (стартовый) и объект- ключ (финальный); возвращает часть своего отображения, со стартового ключа (включительно) до финального ключа (включительно); возвращает отображение;

`tailMap()`

используется для возврата части отображения; принимает параметр — объект-ключ; возвращает часть своего отображения, с указанного ключа (включительно); возвращает отображение;

Интерфейс NavigableMap

Этот интерфейс наследует от интерфейса `SortedMap`.
Он вводит следующие методы.

`ceilingEntry()`

используется для получения элемента; принимает параметр — объект- ключ; возвращает элемент с наименьшим ключем до указанного ключа; возвращает элемент;
этот метод производит сравнение оператором `>=`;

`floorEntry()`

используется для получения элемента; принимает параметр — объект- ключ; возвращает элемент с наибольшим ключем от указанного ключа; возвращает элемент;
этот метод производит сравнение оператором `<=`;

`higherEnty()`

используется для получения элемента; принимает параметр — объект- ключ; возвращает элемент с наименьшим ключем от указанного ключа; возвращает элемент;
этот метод производит сравнение оператором `>`;

`lowerEntry()`

используется для получения элемента; принимает параметр — объект- ключ; возвращает элемент с наибольшим ключем от указанного ключа; возвращает элемент;
этот метод производит сравнение оператором `<=`;

`firstEntry()`

используется для возврата первого элемента отображения; параметров не принимает; возвращает первый элемент отображения; возвращает элемент;

`lastEntry()`

используется для возврата последнего элемента отображения; параметров не принимает; возвращает последний элемент отображения; возвращает элемент;

`pollFirstEntry()`

используется для выталкивания первого элемента отображения; параметров не принимает; выталкивает первый элемент отображения;
возвращает элемент;

`pollLastEntry()`

используется для выталкивания последнего элемента отображения; параметров не принимает;
выталкивает последний элемент отображения;
возвращает элемент;

`ceilingKey()`

используется для возврата элемента; принимает параметр — объект- ключ; возвращает наименьший ключ элемента отображения, с указанного ключа; возвращает объект- ключ;
этот метод производит сравнение оператором `>=`;

`floorKey()`

используется для возврата элемента; принимает параметр — объект- ключ; возвращает наибольший ключ элемента отображения, до указанного ключа; возвращает объект- ключ;
этот метод производит сравнение оператором `<=`;

`lowerKey()`

используется для возврата элемента; принимает параметр — объект- ключ; возвращает наибольший ключ элемента отображения, до указанного ключа; возвращает объект- ключ;
этот метод производит сравнение оператором `<`;

`higherKey()`

используется для возврата элемента; принимает параметр — объект- ключ; возвращает наименьший ключ элемента отображения, с указанного ключа; возвращает объект- ключ;
этот метод производит сравнение оператором `>`;

`descendingKeySet()`

используется для возврата отображения; параметров не принимает; возвращает свое отображение, с обратным порядком элементов; возвращает отображение (`NavigableSet`);

`descendingMap`

используется для возврата отображения; параметров не принимает; возвращает свое отображение, с обратным порядком элементов; возвращает отображение (`NavigableMap`);

`navigableKeySet()`

используется для возврата отображения; параметров не принимает; возвращает свое отображение, с прямым порядком элементов; возвращает отображение (`NavigableSet`);

`headMap()`

используется для получения части отображения; принимает параметры — объект- ключ (верхняя граница) и логическое значение (`true` – включать границу, `false` – нет); руководствуясь переданными параметрами, возвращает часть своего отображения; возвращает отображение (`NavigableMap`);

`subMap()`

используется для получения части отображения; принимает параметры — объект- ключ (верхняя граница) и логическое значение (`true` – включать границу, `false` – нет) и объект- ключ (нижняя граница) и логическое значение

(`true` – включать границу, `false` – нет); руководствуясь переданными параметрами, возвращает часть своего отображения; возвращает отображение (`NavigableMap`);

`tailMap()`

используется для получения части отображения; принимает параметры — объект- ключ (нижняя граница) и логическое значение (`true` – включать границу, `false` – нет); руководствуясь переданными параметрами, возвращает часть своего отображения; возвращает отображение (`NavigableMap`);

Класс ArrayList

Это обобщенный класс (`class ArrayList<T>`), служащий для организации списка массивов, который может изменять свою длину динамически (как в большую, так и в меньшую сторону, при необходимости соответствующего изменения). По сути, он представляет собой список (динамический массив) ссылок на его элементы.
Этот класс наследует от класса `ArrayList`, и реализует интерфейс `List`.

Конструктор класса ArrayList

`ArrayList()`

конструктор класса `ArrayList`; принимает параметры — либо ничего (тогда будет создан пустой результирующий объект), либо списочный массив (массив, из которого будет создан результирующий объект), либо `int` (стартовая емкость создаваемого объекта); руководствуясь переданными параметрами, создает объект своего класса; возвращает ничтоже;

Методы класса ArrayList

`toString()`

используется для преобразования к строке; параметров не принимает; преобразует свойобъект к строке; возвращает строку;

`add()`

используется для добавления элементов в список; принимает параметры – `int` (индекс, по которому будет вставлен объект) и объект (вставляемый объект, элемент коллекции); руководствуясь переданными параметрами, вставляет объект в список; возвращает ничтоже;

`addAll()`

используется для добавления элементов в список; принимает параметры – `int` (индекс, по которому будет вставлен объект) и объект (вставляемый объект коллекции); руководствуясь переданными параметрами, вставляет все элементы указанной коллекции в список; возвращает логическое значение (`true` – свой объект коллекции изменен, `false` - нет);

`get()`

используется для получения элемента коллекции; принимает параметр – `int` (индекс элемента); возвращает элемент коллекции по указанному индексу; возвращает объект;

`set()`

используется для установки значения элемента коллекции; принимает параметры – `int` (индекс целевого элемента коллекции) и объект (содержащий значение инициализации); руководствуясь переданными параметрами, устанавливает значение элемента коллекции; возвращает объект (значение инициализации);

`remove()`

используетсядляудаления элемента из коллекции; принимает параметр – `int` (индекс удаляемого элемента);
удаляет указанный элемент из коллекции;
возвращает объект (удаленный элемент);

`of()`

используется для создания объекта `List`; принимет параметры — объекты (элементы коллекции);
создает объект `List` из указанных элементов;
возвращает объект `List`;
этот метод статический;

`subList()`

используется для получения подсписка элементов коллекции; принимает параметры – `int` (стартовая позиция) и `int` (финальная позиция);
создает подсписок из указанных параметрами элементов своей коллекции;
возвращает объект `List`;

`listIterator()`

используется для возврата итератора для обхода списка; параметров не принимает; возвращает итератор для обхода своего объекта; возвращает объект `ListIterator`;

`indexOf()`

используется для возврата индекса элемента коллекции; принимает параметр — объект (указанный элемент коллекции); возвращает индекс указанного элемента коллекции; возвращает `int`;

`lastIndexOf()`

используется для возврата индекса элемента коллекции; принимает параметр — объект (указанный элемент коллекции); возвращает индекс последнего вхождения указанного элемента коллекции в коллекцию; возвращает `int`;

`sort()`

используется для сортировки своего объекта; принимает параметр — объект-компаратор; руководствуясь переданными параметрами, сортирует свой объект; возвращает ничтоже;

`toArray()`

используется для преобразования своего объекта к массиву; параметров не принимает;
преобразует свой объект к массиву;
возвращает объект `Array`;
этот метод может принимать параметр — массив, тогда результирующий массив будет из элементов того же типа, что у указанного массива;

`ensureCapacity()`

используется для явного увеличения емкости коллекции; принимает параметр – `int` (количество элементов, этим количеством определяется новый размер коллекции);
увеличивает размер своей коллекции (устанавливает его указанным значением);
возвращает ничтоже;
этот метод имеет смысл применять когда заведомо известно, что размер коллекции достигнет определенного значения, так как это позволит заведомо произвести выделение памяти (что затратно);

`trimToSize()`

используется для усечения размера коллекции; принимает параметр — `int` (количество элементов, этим количеством определяется новый размер коллекции); усекает размер своей коллекции (устанавливает его указанным значением); возвращает ничтоже;
этот метод имеет смысл применять когда заведомо известно, что размер коллекции достигнет определенного значения, так как это позволит заведомо произвести освобождение памяти (что даст возможность ее своевременного выделения,что невозможно принехватке памяти);

Класс LinkedList

Это обобщенный класс (`class LinkedList<T>`), служащий для создания связанных списков.
Он наследует от класса `AbstractSequentialList`, и реализует интерфейсы `List`, `Deque` (наследует от интерфейса `Queue`).
Своих методов не вводит.

Конструктор класса LinkedList

`LinkedList()`

используетсядлясоздания объектов класса `LinkedList`; принимает параметры — либо ничего (тогда будет создан пустой объект), либо объект (тогда будет создан объект из указанного); руководствуясь переданными параметрами, создает объект своего класса; возвращает ничтоже;

Класс HashSet

Этот класс позволяет создавать таблицы хэшей.
Это абстрактный класс, наследующий от класса `AbstractSet` и реализующий интерфейс `Set`.
Своих методов не вводит.

Конструктор класса HashSet

`HashSet()`

конструктор класса `HashSet`; принимает параметры — либо ничего (тогда будет создано пустое множество, по умолчанию его емкость `16`), либо объект коллекции (тогда из ее элементов будет создано множество), либо `int` (тогда

множество будет создано заданного размера), либо `int` (размер множества) и `float` (коэффициент заполнения, когда множество заполняется на указанную этим числом долю его текущей емкости, его размер увеличивается; допустимые значения [`0.0 … 1.0`]; если этот параметр не задан явно, то значение `0.75`); руководствуясь переданными параметрами, создает объект своего класса; возвращает ничтоже;

Класс LinkedHashSet

Этот обобщенный класс наследует от класса `HashSet`, и расширяет его функциональность (он хранит элементы упорядочено — в порядке их добавления). У этого класса те же самые конструкторы (разумеется, имя конструктора соответствующее – `LinkedHashSet`) и методы, что у класса `HashSet`.

Класс TreeSet

Это обобщенный класс (`class TreeSet<T>`), который служит для хранения множества в древовидной структуре в отсортированном порядке.
Он наследует от класса `AbstractSet`, и реализует интерфейс `NavigableSet`.
Своих методов не вводит.

Конструктор класса TreeSet

`TreeSet()`

конструктор класса TreeSet; принимает параметры — либо ничего (тогда — пустое множество), либо объект коллекции (тогда множество будет создано из ее элементов), либо объект коллекции (SortedSet – тогда множество будет создано из элементов этой отсортированной коллекции), либо компаратор (тогда будет создано пустое множество, которое будет хранить данные отсортировано согласно компаратору);
руководствуясь переданными параметрами, создает объект своего класса; возвращает ничтоже;

Класс TreeMap

Это обобщенный класс (`class TreeMap<T, U>`), он позволяет создавать отображения, хранящиеся в древовидной структуре.
Этот класс наследует от класса `AbstractMap`, и реализует интерфейс `NavigableMap` (наследующий от интерфейса `SortedMap`).

Конструктор класса TreeMap

```
TreeMap()
```

конструктор класса `TreeMap`; принимает параметры — либо ничего (тогда будет создано пустое отображение), либо объект отображение – `Map` (тогда конструируемое отображение будет создано из него), либо объект отображение – `SortedMap` (тогда конструируемое отображение будет создано из него), либо компаратор (тогда отображение будет хранить элементы отсортировано согласно компаратору);
создает объект своего класса;
возвращает ничтоже;

Класс PriorityQueue

Это обобщенный класс (`class PriorityQueue<T>`), служащий для создания очередей с приоритетами.
Он наследует от класса `AbstractQueue`, и реализует интерфейс `Queue`.
Свои методы не вводит.

Конструктор класса Priorityueue

```
PrioriyQueue()
```

конструктор класса `PriorityQueue`;
приниамает параметры — либо ничего (тогда — пусая очередь), либо `int` (размер очереди), либо `int` (размер очереди) и компаратор (отвечает за сортировку), либо объект коллекции (очередь будет создана из ее элементов), либо объект коллекции (`PriorityQueue`), либо объект коллекции (`SortedSet`);
руководствуясь переданными параметрами, создает объект своего класса;
возвращает ничтоже;

Класс ArrayDeque

Это обобщенный класс (`class ArrayDeque<T>`), позволяющий создавать динамические массивы. Он позволяет создавать очереди (в том числе, двунаправленные).
Этот класс наследует от класса `AbstractCollection`, реализует интерфейс `Deque` (наследует от интерфейса `Queue`).
Собственных методов не вводит.

Конструктор класса ArrayDeque

`ArrayDeque()`

конструктор класса `ArrayDeque`; принимает параметры — либо ничего (тогда будет создана пустая очередь, ее размер по умолчанию `16`), либо `int` (размер очереди, в элементах), либо объект (тогда очередь будет создана из заданной коллекции); создает объект своего класса; возвращает ничтоже;

Класс EnumSet

Это обобщенный класс (`class EnumSet<E extends Enum<E>>`), используется для создания множества с ключами типа перечисления. Объявление класса выглядит указанным образом потому, что необходимо проследить, чтобы он конкретизировался как перечисление.
Он наследует от класса `AbstractSet`, и реализует интерфейс `Set`.
Класс `EnumSet` не имеет конструкторов, так как он — перечисление.
Он вводит следующие методы.

`allOf()`

используется для создания перечисления; принимает параметр — перечисление (из него и будет создано результирующее перечисление); руководствуясь переданными параметрами, создает перечисление; возвращает перечисление;
этот метод статический;

`complementOf()`

используется для создания перечисления; принимает параметр — перечисление (из него и будет создано результирующее перечисление); руководствуясь переданными параметрами, создает перечисление (притом, дополняя отсутствующие в перечислении элементы); возвращает перечисление;
этот метод статический;

`copyOf()`

используется для создания перечисления; принимает параметр — перечисление или коллекцию, конкретизированную в типе перечисления (из него и будет создано результирующее перечисление);
руководствуясь переданными параметрами, создает перечисление;
возвращает перечисление;
этот метод статический;

`noneOf()`

используется для создания перечисления; принимает параметр — перечисление (будет создано результирующее перечисление, исключающее его типы — пустое перечисление); руководствуясь переданными параметрами, создает перечисление; возвращает перечисление;
этот метод статический;

`of()`
используется для создания перечисления; принимает параметры — ряд перечислений (из него и будет создано результирующее перечисление); руководствуясь переданными параметрами, создает перечисление; возвращает перечисление;
этот метод статический;

`range()`

используется для создания перечисления; принимает параметр — перечисление и перечисление (в пределах заданных перечислений и будет создано результирующее перечисление); руководствуясь переданными параметрами, создает перечисление; возвращает перечисление;
этот метод статический;

Компараторы

Компаратор — это объект, используемый для разрешения задачи сравнения с элементами коллекций. Они необходимы, поскольку сам язык не располагает логикой сравнения пользовательских объектов, используемых как элементы коллекций. Все объекты, безусловно, могут сравниваться как объекты; но не все объекты могут сравниваться как элементы коллекций.
Для того, чтобы пользовательские объекты можно было сравнивать как элементы коллекций, они должны реализовать интерфейс `Comparable` (`java.lang.Comparable`). Объект, реализующий этот интерфейс, является объектом- компаратором. Этот интерфейс оперирует обобщенным типом.
Этот интерфейс содержит метод `compareTo()`; он используется для сравнения объектов, принимает параметр — объект, и сравнивает его со своим объектом, возвращает `int` (отрицательное — свой объект меньше, ноль — равен, положительное — больше).
Этот интерфейс также содержит метод `compare()` - он идентичен вышеописанному методу, за тем исключением, что сравнивает два объекта (возвращает отрицательное — если первый объект меньше).
Кроме того, если конструктор класса- коллекции поддерживает создание объекта- коллекции, принимая объект- компаратор — то сами элементы коллекции могут и не реализовать интерфейс `Comparable`, переданный в параметрах компаратор все равно будет использоваться для их сравнения.

И, кроме того, язык поддерживает применение цепочек компараторов. Для этого применяется метод `thenComparing()`, определенный в интерфейсе `Comparable`. Этот метод принимает параметр — объект- компаратор. В результате применения этого метода, сравнение будет происходить сначала согласно первому компаратору (из которого вызван этот метод), затем согласно второму (переданному в параметрах). Этот метод применяется при создании объекта- компаратора. В записи это выглядит так:

```
C0ClassName ObjectName
= new C0ClassName().thenComparing(new C1ClassName());
```

в языке `Java` часто ипользуется такой способ вызова методов. Он не часто обсуждается в учебных пособиях, но он вполне корректен.

Итераторы

Итератор — это объект, реализующий интерфейс `Iterator`. Этот интерфейс вводит следующие методы: `next()`, `hasNext()`, `remove()`. Все эти методы описаны выше (это методы интерфейсов и классов коллекций).

Итератор также может реализовать интерфейс `ListIterator`. Он вводит следующие методы: `add()`, `next()`, `hasNext()`, `previous()`, `hasPrevious()`, `previousIndex()`, `set()`, `remove()`. Языковые классы (и их методы), реализующие этот интерфейс, были рассмотрены выше.

Использование итераторов производится применением их методов (что логично в рамках ООП).

Часть 5 (Stream API)

Stream API

Stream API – это средства, предназаначенные для поиска, сортировки, и прочего манипулирования данными. Эти средства используются с применением коллекций, многопоточного исполнения кода, и лямбда- выражений.
Stream API работает с потоками данных. Речь идет о потоках данных, обрабатываемых в процессе вычислений. Таким образом, поток данных — это не поток исполнения программы, и не устройство ввода- вывода. Источниками данных в потоках данных являются потоки ввода- вывода, коллекции, и все, что может выступать источником секвенции тех или иных данных.
Средства StreamAPI располагаются в пакете `java.util.stream`. В этом пакете располагается ряд интерфейсов. Вершина их иерархии — интерфейс `BaseStream`. Это обобщенный интерфейс, наследующий от `AutoClosable`. Интерфейс `BaseStream` вводит следующие методы.

Методы интерфейса BaseStream

`close()`

используется для закрытия потока данных; параметров не принимает; закрывает вызывающий поток данных; возвращает ничтоже;

`isParallel()`

используется для определения, является ли поток данных параллельным; параметров не принимает; проверяет, является ли вызывающий поток параллельным; возвращает логическое значение (`true` – да, `false` - нет);

`iterator()`

используется для возврата итератора; параметров не принимает; возвращает итератор для своего потока данных; возвращает объект- итератор;

`onClose()`

используется для возврата потока данных; принимает параметры — обработчик событий (закрытия потока); возвращает (новый) поток данных, с указанным обработчиком; возвращает поток;

`parallel()`

используется для возврата параллельного потока данных; параметров не принимает; возвращает параллельный поток данных, созданный на базе своего (если свой поток и так параллельный — то его и возвращает); возвращает поток;

`sequental()`

используется для возврата последовательного потока данных; параметров не принимает; возвращает последовательный поток данных, созданный на базе своего (если свой поток и так последовательный — то его и возвращает); возвращает поток;

`spliterator()`

используется для возврата итератора- разделителя; параметров не принимает; возвращает итератор- разделитель для своего потока данных; возвращает объект — итератор- разделитель;

`unordererd()`

используется для возврата неупорядоченного потока данных; параметров не принимает;
возвращает неупорядоченный поток данных, созданный на базе своего (если свой поток и так неупорядоенный — то его и возвращает);
возвращает поток;

Интерфейс `Stream` наследует от `BaseStream`.
Он вводит следующие методы.

Методы интерфейса Stream

`collect()`

используется для аккумулирования элементов; принимает параметр — пользовательскую функцию;
руководствуясь переданными параметрами, аккумулирует элементы в объекте-контейнере и возвращает контейнер; возвращает объект- контейнер;
это конечная операция;
пользовательская функция должна регулировать процесс накопления (она используется для накопления в определенном порядке);
эта функция должна возвращать обобщение (оперирующее типами самого объекта- контейнера и элементов в контейнере);
этафункциядолжна оперировать следующими типами: элементами потока данных, элементами контейнера, и контейнером;

`count()`

используется для подсчета элементов потока; параметров не принимает; возвращает количество элементов своего потока данных; возвращает `long`;
это конечная операция;

`filter()`

используется для создания потока данных; принимает параметр — пользовательскую функцию; создает новый поток данных, используя для этого свой поток данных, и отфильтровываяиз него элементы (руководствуясь переданными параметрами); возвращает поток данных;
об этой функции — ее тело исполняет операции над отдельным элементом, ее вызов осуществляется автоматически для каждого из перебираемых элементов;
это промежуточная операция;

`forEach()`

используется для перебора элементов потока; принимает параметр — пользовательскую функцию; руководствуясь переданными параметрами, выполняет перебор элементов своего потока; возвращает ничтоже;
об этой функции — ее тело исполняет операции над отдельным элементом, ее вызов осуществляется автоматически для каждого из перебираемых элементов;
это конечная операция;

`map()`

используется для отображения потока данных; принимает параметры — значение (используется пользовательской функцией как первый аргумент при первом ее вызове, этот параметр можно опустить) и пользовательскую функцию; руководствуясь переданными параметрами, выполняет отображение элементов своего потока; возвращает поток;
об этой функции — ее тело исполняет операции над отдельным элементом, ее вызов осуществляется автоматически для каждого из перебираемых элементов;
это промежуточная операция;

`mapToInt()`

используется для отображения потока данных; принимает параметр — пользовательскую функцию; руководствуясь переданными параметрами, выполняет отображение элементов (с соответствующим типом элементов результирующего потока) своего потока; возвращает поток;
об этой функции — ее тело исполняет операции над отдельным элементом, ее вызов осуществляется автоматически для каждого из перебираемых элементов;

это промежуточная операция;

`mapToLong()`

используется для отображения потока данных; принимает параметр — пользовательскую функцию; руководствуясь переданными параметрами, выполняет отображение элементов (с соответствующим типом элементов результирующего потока) своего потока; возвращает поток;
об этой функции — ее тело исполняет операции над отдельным элементом, ее вызов осуществляется автоматически для каждого из перебираемых элементов;
это промежуточная операция;

`mapToDouble()`

используется для отображения потока данных; принимает параметр — пользовательскую функцию; руководствуясь переданными параметрами, выполняет отображение элементов (с соответствующим типом элементов результирующего потока) своего потока; возвращает поток;
об этой функции — ее тело исполняет операции над отдельным элементом, ее вызов осуществляется автоматически для каждого из перебираемых элементов;
это промежуточная операция;

`min()`

используется для возврата минимального элемента из потока; принимает параметр — компаратор; руководствуясь переданными параметрами, возвращает минимальный элемент своего потока; возвращает элемент;
это конечная операция;

`max()`

используется для возврата максимального элемента из потока; принимает параметр — компаратор; руководствуясь переданными параметрами, возвращает максимальный элемент своего потока; возвращает элемент;
это конечная операция;

`reduce()`

используется для редуцирования потока; принимает параметры — пользовательскую функцию; руководствуясь переданными параметрами, выполняет редуцирование своего потока; возвращает элемент;
об этой функции — ее тело исполняет операции над отдельным элементом, ее вызов осуществляется автоматически для каждого из перебираемых элементов (конкретнее, она вызывается следующим образом: она вызывается со своими аргументами, затем в последующих вызовах ее результат выступает ее первым

аргументом, а последующими аргументами — перебираемые элементы, и по их перебору возвращается результат — единственное значение);
это конечная операция;

`sorted()`

используется для сортировки потока; параметров не принимает; возвращает поток, состоящий из элементов своего потока, идущих в отсортированном порядке (порядок сортировки естественный (по возрастанию)); возвращает поток;
это промежуточная операция;

`toArray()`

используется для преобразования к массиву; параметров не принимает; преобразует свой поток к массиву; возвращает массив;
это конечная операция;

Об операциях с потоками данных

Все операции с потоками целесообразно подразделять на промежуточные (возвращают поток данных), и конечные (возвращают результат процессирования данных). Исходя из этого, полезная работа производится именно конечными операциями.
Жизненный цикл потока данных выглядит так: поток данных создается, опционально выполняются промежуточные операции над ним, и затем совершается конечная операция. После выполнения конечной операции прочие операции (промежуточные или конечные) над потоком не возможны.
Об операциях над потоками также следует заметить следующее. Промежуточные операции не исполняются немедленно. Они исполняются отложено — когда какая- либо конечная операция исполняется над потоком промежуточной операции.
Также следует заметить — при использовании методов потоков данных, принимающих функцию/ объект, инкапсуллирующий функцию, этим методам можно просто передавать лямбда- выражение.

Создание потоков данных

Потоки данных можно создать из коллекций. Для этого применяется метод `stream()`, доступный всем коллекциям:

```
Stream<VarType> StreamName
= CollectionVarType.stream(CollectionVarName);
```

этот статический метод вызывается как метод определенного класса коллекций, получает в параметрах объект коллекции, и возвращает поток данных.

Аналогичный метод `parallelStream()` создает параллельный поток данных. Если создание параллельного потока не возможно, то создается последовательный поток.
Параллельные потоки поддерживают параллельное исполнение операций. В целях корректного исполнения параллельных операций с потоками данных, при распараллеленной работе потоками данных следует избегать операций с сохранением данных (таких, как метод сортировки потока данных).
Потоки данных также создаются и промежуточными операциями над ними.