Часть 1 (AWT)

AWT

AWT – Advanced WindowingToolkit. Это самая первая графическая библиотека. И, при этом, единственная графическая библиотека, не объявленная устаревшей (завершение срока ее поддержки не объявлено). Эта библиотека располагается в пакете `java.awt`.

Класс Component

Этот класс является вершиной иерархии классов отображаемых объектов (кроме объектов меню).
Это абстрактный класс.

Класс Container

Это подкласс `Component`, он отвечает за компоновку отображаемых объектов (он используется для инкапсуллирования отображаемых объектов).

Класс Panel

Это подкласс `Container`, является суперклассом для `Applet`.

Класс Window

Это подкласс `Container`, используется для создания собственно окна.

Класс Frame

Это подкласс `Window`, используется для создания контента окна.
Окно создается либо без заголовка, либо с заголовком (тогда конструктор окна принимает параметр — строку (заголовок окна), и его запись принимает вид `Frame( … );`).
Заголовок окна можно задать/ изменить и после его создания — методом `setTitle()` - он принимает строку (заголовок окна) и возвращает ничтоже.

Размеры окна задаются после его создания — методом `setSize()`, он принимает в параметрах объект `Dimension` (содержит габариты окна — `width`, `height`), либо два целочисленных (ширина и высота — габариты окна в пикселах). Возврат габаритов окна производится методом `getSize()` - он возвращает объект `Dimension`.
Созданное окно изначально является не отображаемым. Отображение созданного окна регулируется методом `setVisible()` - принимает логическое значение (`true` – visible, `false` – no).
На практике окно создается не как объект `Frame`, а как объект его подкласса (это потому, что окно класса `Frame` не позволяет динамически формировать контент в нем; чтобы избежать этих ограничений, в пользовательском подклассе `Frame` следует переопределить соответствующие методы).

Класс Canvas

Это подкласс `Component`, используется для создания окна, инкапсуллирующего его содержимое.

О графическом выводе

Весь графический вывод происходит в графическом контексте. Графический контекст инкапсуллиован объектом `Graphics`.
Этот объект может быть возвращен методом `getGaphics()` класса `Component`. Объект `Graphics` содержит ряд методов, в том числе методы для рисования графических примитивов (о них — вдругих источниках).

Цвет

Средства для работы с цветом (и собственно цвет) содержатся в классе `Color`. Пользовательский цвет можно создать так — `Color( … )`, передав в параметрах секвенцию целочисленных (ряд значений R, G, B, в десятичке), либо целочисленное (собственно значение целевого цвета, 32-битное, младший байт — не значащий).
Методы `getRed()`, `getGreen()`, `getBlue()` возвращают значения одноименных составляющих цвета; параметров непринимают; возвращают целочисленное.
Метод `getRGB()` возвращает собственно значение цвета; параметров непринимает; возвращает целочисленное.
Метод `setColor()` позволяет изменить значение цвета, принимает параметр — значение цвета (объект `Color`), возвращает ничтоже.

Шрифт

Средства для работы со шрифтом (и собственно шрифт) содержаться в классе `Font`. Шрифт характеризуется именем семейства, логическим именем — важно во время выполнения, и именем гарнитуры.
Статический метод `equals()` возвращает результат проверки, является ли указанный объект шрифта содержащим тот же шрифт, что и свой объект.
Статический метод `hashCode()` возвращает `int` – свой хэш.
Метод `toString()` возвращает строку — представление своего шрифта.
Статический метод `decode()` возвращает объект шрифта; принимает строку — имя шрифта.
Статический метод `getFamily()` возвращает строку — имя семейства своего шрифта.
Статический метод `getFont()` возвращает объект шрифта; принимает строку — системное свойство.
Статический метод `getFontName()` возвращает строку — имя своего шрифта.
Статический метод `getName()` возвращает строку — лог. имя своего шрифта.
Статический метод `getSize()` возвращает `int` – размер своего шрифта, пт.
Статический метод `getStyle()` возвращает `int` – стиль своего шрифта.
Метод `isBold()` возвращает `boolean` – является ли начертание таковым.
Метод `isItalic()` возвращает `boolean` – является ли начертание таковым.
Метод `isPlain()` возвращает `boolean` – является ли начертание таковым.
При работе с шрифтами целесообразно располагать списком доступных шрифтов. Метод `GraphicsEnvironment.getAvailableFontFamilyNames()` возвращает массив строк — имен семейств шрифтов; метод `GraphicsEnvironment.getAllFonts()` возвращает массив объектов шрифтов. Эти методы не статические, и вызываются из экземпляра указанного класса; ссылку на него можно получить статическим методом `GraphicsEnironment.getLocalGraphicsEnvironment()`.
Чтобы использовать шрифт, требуется создать объект `Font`: конструктор принимает параметры — строку (имя шрифта), `int` (начертание), `int` (размер,пт.). Реализации языка должны поддерживать шрифты `Dialog`, `DialogInput`, `SansSerif`, `Serif`, `Monospaced`. Начертание может быть задано константами: `PlAIN`, `BOLD`, `ITALIC`. Эти константы статические. Чтобы начать работать со шрифтом, его надо выбрать — методом `setFont()` - принимает в параметрах объект шрифта; возвращает ничтоже.

Элементы управления

Речь идет о стандартных элементах управления, которыми располагают графические программы.
Чтобы оснастить приложение элементом управления, необходимо создать экземпляр элемента управления, и добавить его в окно приложения (методом

`add()` – принимает ссылку на экземпляр элемента управления; возвращает ссылку на него). После введения элемент будет отображаться в окне. Чтобы удалить из приложение элемент управления, необходимо воспользоваться соответствующим методом (методом `remove()` – принимает ссылку на экземпляр элемента управления; возвращает ничтоже). Метод `removeAll()` удаляет все элементы управления.
Логика каждого элемента управления запускается событием этого элемента.
Далее будут рассмотрены конкретные элементы управления.

Метки

Метки это просто текстовые строки, отображаемые в приложении.
Они создаются так: `Label()` – параметров либо не принимает (тогда метка будет пустой), либо принимает строку (контент метки) и целочисленное (способ выравнивания текста метки — статические константы `Label.LEFT`, `Label.RIGHT`, `Label.CENTER`); возвращает ничтоже.
Метод метки `getText()` возвращает строку (контент метки). Метод `setText()` устанавливает контент метки; принимает параметр — строку. Метод `getAlignment()` возвращает выравнивание текста метки (целочисленное, вышеупомянутые константы). Метод `setAlignment()` устанавливает выравнивание текста метки; принимает целочисленное (вышеупомянутые константы).

Кнопки

Это собственно кнопки, отображаемые в приложении.
Они создаются так: `Button()` – параметров либо не принимает, либо принимает строку (выводится на кнопке); возвращает ничтоже.
После создания кнопки можно получить ее строку — методом `getLabel()`. Установить эту строку можно методом `setLabel()` – принимает строку; возвращает ничтоже.

Флажки

Это собственно флажки, отображаемые в приложении.
Они создаются так: `Checkbox()` – параметров либо не принимает, либо принимает строку (выводится при флажке); возвращает ничтоже. Может принимать также еще логическое значение (исходное состояние флажка, `true` – on), объект `CheckboxGroup` (объект группы флажков, если указан — то флажок будет в этой группе); причем, эти два параметра могут следовать друг за другом в разном порядке.
Метод флажка `getState()` возвращает логическое значение (состояние флажка). Метод `setState()` устанавливает состояние флажка. Метод

`getLabel()` возвращает строку (выводимую у флажка). Метод `setLabel()` устанавливает строку (выводимую у флажка).

Переключатели

Это собственно переключатели, отображаемые в приложении.
Они создаются так: создается группа переключателей (`CheckboxGroup()`), затем она указывается при создании самих переключатедей (сами переключатели создаются так — просто создаются объекты- переключатели — `Checkbox()` – конструктор принимает строку (отображаемую при переключателе), имя объекта группы переключателей (в которую включается переключатель), и логическое значение (состояние переключателя, `true` – on)).
Метод `getSelectedCheckbox()` группы переключателей возвращает ссылку на установленный переключатель. Метод `setSelectedChecbox()` группы переключателей устанавливает указанный переключатель.

Раскрывающиеся списки

Это собственно раскрывающиеся списки, отображаемые в приложении.
Они создаются так: создается объект списка (`Choice()`), и формируются элементы списка (элементы списка представляют собой просто строки, они добавляются методом `add()` объекта списка, этот метод принимает строку).
Метод `getSelectedItem()` возвращает строку — выбранный элемент.
Метод `getSelectedIndex()` возвращает `int` (индекс выбранного элемента, индексация начинается с нуля). Метод `getItemCount()` возвращает `int` (количество элементов). Метод `select()` устанавливает указанный элемент списка (он указывается его строкой или его индексом). Метод `getItem()` возвращает указанный элемент (он указывается его строкой или индексом).
Метод `getItemCount()` возвращает `int` – количество элементов.

Списки

Это собственно прокручивающиеся списки, отображаемые в приложении.
Они создаются так: создается объект списка (`List()` – принимает параметры — либо ничего, либо `int` (количество строк, отображаемых в окне прокрутки), и логическое значение (указывает, можно ли выбрать более одного элемента из списка, этот параметр можно опустить)); затем методом `add()` объекта списка добавляются его элементы (метод принимает строку — имя элемента, и `int` – его позицию в списке (с нуля) — этот параметр можно опустить).
Метод `getSelectedItem()` возвращает строку — выбранный элемент.
Метод `getSelectedIndex()` возвращает `int` (индекс выбранного элемента, индексация начинается с нуля). Метод `getItemCount()` возвращает `int`

(количество элементов). Метод `select()` устанавливает указанный элемент списка (он указывается его строкой или его индексом).
Метод `getSelectedItems()` возвращает массив строк — выбранных элементов. Метод `getSelectedIndexes()` возвращает массив `int` (индекс выбранного элемента, индексация начинается с нуля).
Метод `getItemCount()` возвращает `int` – количество элементов.

Полосы прокрутки

Это собственно полосы прокрутки, отображаемые приложением.
Они могут располагаться как горизонтально, так и вертикально. И имеют элементы управления — стрелки прокрутки и ползунок. Один клик по стрелке считается за одну единицу прокрутки.
Полосы прокрутки — это объекты класса `Scrollbar`. Конструктор принимает параметры — либо ничего, либо `int` (одна из констант `Scrollbar.VERTICAL`, `Scrollbar.HORIZONTAL`), и ряд `int` который можно опустить (это исходное положение ползунка, размер ползунка, минимум прокрутки, и максимум прокрутки). Если этот ряд параметров опущен, то соответствующие значения следует определить после создания объекта полосы прокрутки. Это делается методом `setValues()` - он принимает вышеуказанный ряд параметров.
Метод `getValue()` возвращает текущее значение прокрутке в единицах прокрутки (кликах по стрелке прокрутки). Метод `setValue()` устанавливает значение полосы прокрутки, принимает параметр – `int`. Методы `getMinimum()`, `getMaximum()` возвращают экстремальные значения прокрутки. Метод `setUnitIncrement()` изменяет значение шага прокрутки. Метод `setBlockIncrement()` изменяет значение шага постраничной прокрутки.

Текстовые поля

Это собственно текстовые поля однострочные, отображаемые в приложении.
Текстовые поля создаются так: `TextField()` - принимает параметры — либо ничего, либо `int` или `String` (заданная ширина поля в символах или строка отображаемая в поле по умолчанию соответственно), либо `String` и `int` (строка отображаемая по умолчанию и ширина в символах).
Метод `getText()` возвращает строку, отображаемую в поле; параметров не принимает. Метод `setText()` устанавливает строку, отображаемую в поле; возвращает ничтоже. Метод `select()` выделет указанный участок отображаемой строки; принимает параметры `int` (стартовая позиция выделения), `int` (финальная позиция выделения); возвращает ничтоже. Метод `getSelectedText()` возвращает выделенный участок строки; параметров не принимает; возвращает строку. Метод `isEditable()` проверяет, можно ли изменять текст своего поля; параметров не принимает; возвращает логическое значение (`true` – yes). Метод `setEditable()` позволяет задать, редактируем

ли текст поля; принимает параметр - логическое значение (`true` – yes); возвращает ничтоже. Метод `setEchoChar()` позволяет задать символ, который заменяет вводимые пользователемв поле символы; принимает параметр — `char`. Метод `getEchoChar()` возвращает этот символ. Метод `echoCharIsSet()` проверяет, использован ли метод `setEchoChar()`; возвращает логическое значение.

Текстовые области

Это собственно текстовые поля многостроные, отображаемые в приложении.
Они создаются так: `TextArea()` – конструктор принимает параметры — либо ничего, либо строку (выводится по умолчнию), либо строку (выводится по умолчанию), `int` (количество строк) и `int` (количество символов), либо строку (выводится по умолчанию), `int` (количество строк), `int` (количество символов), и int (полосы прокрутки — `SCROLLBARS_NONE`, `SCROLLBARS_VERTICAL_ONLY`, `SCROLLBARS_HORIZONTAL_ONLY`, `SCROLLBARS_BOTH`), либо `int` (количество строк) и `int` (количество символов в строке).
Метод `getText()` возвращает строку, отображаемую в поле; параметров не принимает. Метод `setText()` устанавливает строку, отображаемую в поле; возвращает ничтоже. Метод `select()` выделет указанный участок отображаемой строки; принимает параметры `int` (стартовая позиция выделения), `int` (финальная позиция выделения); возвращает ничтоже. Метод `getSelectedText()` возвращает выделенный участок строки; параметров не принимает; возвращает строку. Метод `isEditable()` проверяет, можно ли изменять текст своего поля; параметров не принимает; возвращает логическое значение (`true` – yes). Метод `setEditable()` позволяет задать, редактируем ли текст поля; принимает параметр - логическое значение (`true` – yes); возвращает ничтоже. Метод `append()` дописывает указанную строку в поле; принимает параметр – строку; возвращает ничтоже. Метод `insert()` вставляет указнную строку по указанной позиции; принимает параметры — строку и `int` (указанная позиция); возвращает ничтоже. Метод replaceRange`()` пр заменяет указанной строкой текст в поле по указанным позициям; принимает параметры — строку, `int` (стартовая позиция), и `int` (финальная позиция); возвращает ничтоже.

Диспетчеры компоновки

Компоновка отображаемых компонентов в окне приложения осуществляется либо вручную, либо диспетчером компоновки. Применение диспетчера компоновки позволяет осуществлять компоновку во время исполнения, и использовать полученные во время исполнения величины, значимые для компоновки.

Экземпляры класса `Container` имеют сопоставленные им диспетчеры компоновки. Диспетчер компоновки — это объект какого- либо класса, реализующего интерфейс `LayoutManager`. Диспетчер компоновки устанавливается методом `SetLayout()`; этот метод принимает объект — диспетчер компоновки; возвращает ничтоже. Если этот метод не использован, то применяется диспетчер компоновки по умолчанию. Можно отменить диспетчер компоновки — для этого методу `SetLayout()` передается ссылка на ничто, и тогда компоновка осуществляется вручную.
Есть различные языковые классы диспетчеров компоновки.

Класс FlowLayout

Этот класс используется по умолчанию для поточной компоновки.
Компоненты располагаются последовательно, построчно, и принеобходимости переносятся на следующую строку.
Эти диспетчеры компоновки создаются так — `FlowLayout()`; принимает параметры — либо ничего, либо `int` (способ компоновки — `FlowLayout.LEFT`, `FlowLayout.CENTER`, `FlowLayout.RIGHT`, `FlowLayout.LEADING`, `FlowLayout.TRAILING`), `int` (зазор по горизонтали) и `int` (зазор по вертикали).

Класс BorderLayout

Этот класс служит для компоновки отображаемых компонентов приложения, отображаемого на переднем плане.
Эти диспетчеры компоновки создаются так — `BorderLayout()`; принимает параметры — либо ничего, либо `int` (зазор по горизонтали), и `int` (зазор по вертикали). Они позволяют явно задать компоновку компонента, вводимого в окно — методом `add()`, принимает параметры — вводимый компонент, и объект (способ компоновки, объект класса `Object`, одна из констант `BorderLayout.CENTER`, `BorderLayout.EAST`, `BorderLayout.NORTH`, `BorderLayout.SOUTH`, `BorderLayout.WEST`).

Класс GridLayout

Этот класс служит для компоновки табличным способом.
Эти диспетчеры компоновки содаются так — `GridLayout()`; принимает параметры — либо ничего (и компоновка будет в одном столбце), либо `int` (количество строк) и `int` (количество столбцов), либо то же `int`, `int`, и `int` (зазор по горизонтали), `int` (зазор по вертикали).

Класс CardLayout

Этот класс используется для компоновки по различным картам компоновки.

Эти диспетчеры компоновки создаются так — `CardLayout()`; принимает параметры — либо ничего, либо `int` (зазор по горизонтали), и `int` (зазор по вертикали). Разумеется, если в параметрах указать зазоры, то это скажется на компоновке, и онаможет принять вид, сильно отличающийся от компоновки этим способом по умолчанию.
Карты компоновк хранятся в колоде карт компоновки. Колода карт компоновки хранится в объекте `Panel`. Карты этой колоды — тоже в объектах `Panel`. На картах колоды размещаются отображаемые компоненты приложения. Карты колоды размещаютсяв колоде. Колода карт размещается в окне приложения. Колоде задаетсядиспетчер компоновки — `CardLayout`.
Для работы с картами колоды служат соответствующие методы диспетчера.
Метод `first()` принимает параметр — объект колоды, активирует указанную карту, возвращает ничтоже.
Метод `last()` принимает параметр — объект колоды, активирует указанную карту, возвращает ничтоже.
Метод `next()` принимает параметр — объект колоды, активирует указанную карту, возвращает ничтоже.
Метод `previous()` принимает параметр — объект колоды, активирует указанную карту, возвращает ничтоже.
Метод `show()` принимает параметры — объект колоды и строку (имя карты), отображает указанную карту, возвращает ничтоже.

Класс GridBagLayout

Этот класс альтернативной табличной компоновки. Он позволяет выполнять компоновку в таблице, имеющей иррегулярную структуру (разбитие по столбцам в строках).
Эти диспетчеры создаются так — `GridBagLayout()` - параметров не принимает.
Метод `setConstraints()` диспетчера устанавливает ограничения на компонуемые элементы графического интерфейса. Он принимает параметры — ссылку на компонент GUI, и объект `GridBagConstraints` – объект ограничений. Это следующие ограничения: `int anchor` (выравнивание в ячейке, `GridBagConstraints.CENTER` по умолчанию)), `int fill` (полосы прокрутки, `GridBagConstraints.NONE` по умолчанию), `int gridheight` (высота ячейке, `1` по умолчанию), `int gridwidth` (ширина в ячейке, `1` по умолчанию), `int gridx` (координата, абсцисса ячейки, `GridBagConstraints.RELATIVE` по умолчанию), `int greedy` (координата, ордината ячейки, `GridBagConstraints.RELATIVE` по умолчанию), `int ipadx` (отступ по горизонтали, дополнительный, `0` по умолчанию), `int ipady` (отступ по вертикали, дополнительный, `0` по умолчанию), `Insets insets` (отступы, основные, все `0` по умолчанию), `double weightx` (отступ внешний по горизонтали, `0.0` по умолчанию), `double weighty` (отступ внешний по вертикали, `0.0` по умолчанию). Подробнее об этих значениях — в справочнике.

Меню

Это собственно строка меню и ее контент.
Класс `MenuBar` – строка меню. Класс `Menu` – пункт строки меню. Класс `MenuItem` – пункт его списка, класс `CheckboxMenuItem` – пункт его списка, который можно выбрать наряду с другими такими же пунктами.
Строка меню создается его конструктором, без параметров.
Ее пункт создается его конструктором, без параметров, либо с параметром — строкой (имя).
Пункт его списка создается его конструктором – `MenuItem()`, без параметров, либо с параметром — строкой (имя). Его метод `isEnabled()` позволяет определить его активность, параметров не принимает, возвращает логическое значение. Метод `setEnabled()` позволяет регулировать его активность, принимает логическое значение (`true` – yes), возвращает ничтоже. Метод `getLabel()` возвращает метку этого пункта. Метод `setLabel()` позволяет задать эту метку.
Пункт этого списка можно также создать конструктором `CheckboxMenuItem()`, без параметров, либо с параметрами — строка (метка) и логическое значение (`true` – on, этот параметр можно опустить). Состояние этого пункта можно проверить методом `getState()`, возвращает логическое значение. Метод `setState()` позволяет задать его состояние, принимает логическое значение.
Пункт меню вводится в меню методом `add()` – принимает параметр (вводимый пункт меню), возвращает принятый в параметрах пункт меню. Этим же методом в них вводятся их подпункты. Каждый раз используется метод того объекта, в который вводится пункт.

Диалоговые окна

Это собственно диалоговые окна, формируемые прложением.
В этих окнах отсутствует строка меню, в остальном же они идентичны окнам приложений. Диалоговые окна бывают модальными (безусловно удерживающими на себе фокус внимания) и немодальными (позволяющими переключить фокус внимания на основное окно).
Диалоговые окна создаются конструктором `Dialog()` – принимает параметры — ссылку (на основное окно), строку (заголовок, этот параметр можно опустить), и логическое значение (`true` – modal).
Диалоговые окна для загрузки файлов создаются конструктором `FileDialog()` – принимает параметры — ссылку (на основное окно), строку (заголовок, этот параметр можно опустить), `int` (способ – `FileDialog.LOAD`, `File.Dialog.SAVE`).
Методы `getFile()`, `getDirectory()` позволяют запросить имя и каталог указанного пользователем файла, они возвращают эти сведения в виде строки. Метод `isMultipleMode()` проверяет, можно ли в текущем режиме выбора файлов выбрать несколько файлов, возвращает логическое значение (`true` –

yes). Метод `setMultipleMode()` устанавливает режим, позволяющий выбрать несколько файлов. Метод `getFiles()` позволяет выбрать ряд файлов.

Изображения

Средства для работы с изображениями лежат в пакете `java.awt.image`.
Изображения представляют собой собственно изображения типов GIF, PNG, JPG, инкапсуллированные в объекте класса `Image`.
Изображения создаются методом `createImage()` - принимает параметры — пару чисел (габариты изображения) или (ссылку на) объект (интерфейсного) класса `ImageProducer` (о нем — далее). Этот метод доступен из объектов класса `Container`. Устоявшаяся практика — вызывать его из объекта класса `Canvas`.
Изображение также может быть получено загрузкой — методом `getImage()` - принимает параметры — `URL` (указывает изображение) и строку (имя изображения, этот параметр можно опустить).
Воспроизведение изображения производится методом `drawImage()` класса `Graphics`. Этот метод принимает параметры — объект изображения (класса `Image`), пару чисел (координаты изображения), и объект класса `ImageObserver` (устоявшаяся практика — просто указывать `this`). Класс `ImageObserver` определяет метод `ImageUpdate()` - он возвращает логическое значение (`true` – загрузка изображений завершена).

Интерфейс ImageProducer

Этот (интерфейсный) класс реализован в классах `MemoryImageSource`, `FilteredImageSource`. Этот интерфейс и эти реализующие его классы доступны для использования программистом. Однако, они предназначены для низкоуроневой работы с отображаемым изображением, и, таким образом, они являются служебными.

Прочие классы

Имеются и прочие классы для работы с отображаемым изображением. Однако, они предназначены для низкоуроневой работы с отображаемым изображением, и, таким образом, они являются служебными.

Часть 2 (Сеть)

Сеть

Средства для работы в сети располагаются в пакете `java.net`.

Класс InetAddress

Этот класс используется для работы с адресами в Интернете.
Речь идет как об IP- адресах (IPv4, IPv6), так и DNS- адресах — эти объекты инкапсуллируют все эти адреса.
Объекты этого класса создаются не конструктором, а фабричными методами этого класса:
`InetAddress.getLocalHost()` - возвращает объект (адреса локального хоста),
`InetAdress.getByName()` - возвращает объект (адреса указанного именем хоста),
`InetAddress.getAllByNames()` - возвращает массив объектов (адреса указанных именем хостов; этот метод полезен, когда одному имени сопоставлен ряд IP- адресов — что обыкновенно для Интернета),
`InetAddress.getDyAdress()` - возвращает объект (адреса указанного IP-адресом хоста).

Методы класса InetAddress

`toString()`

используется для возврата строкового представления своего объекта; параметров не принимает;
возвращает строковое предсталение своего объекта (содержит DNS- адрес и IP-адрес);
возвращает строку;

`equals()`

используется для сравнения объектов;
принимает параметр — объект `InetAddress` (указанный объект);
сравнивает свой объект с указанным, и возвращает результат сравнения (сравнение производится по DNS- адресам);
возвращает логическое значение (`true` – equals, `false` – no);

`getAddress()`

используется для возврата IP- адреса; параметров не принимает; возвращает IP-адрес из своего объекта; возвращает массив байтов;

`getHostAddress()`

используется для возврата адреса; параметров не принимает; возвращает адрес (IP, DNS) из своего объекта; возвращает строку;

`getHostName()`

используется для возврата DNS- адреса; параметров не принимает; возвращает DNS- адрес из своего объекта; возвращает массив байтов;

`isMulticastAddress()`

используется для проверки, является ли адрес групповым; параметров не принимает; проверяет, является ли адрес из своего объекта групповым; возвращает логичекое значение (`true` – yes, `false` – no);

Этот класс имеет подклассы — `Inet4Address`, `Inet6Address`; они предназначены для использования только при использовании IPv4, IPv6 соответственно. Они поддерживаются языком сугубо для поддержки мультипротокольности на едином хосте. Если от программы на хосте не требуется мультипротокольности, то использоать эти классы не обязательно, и достаточно класса `InetAddress`.

Сокеты

Это собственно сокеты (также известные как «сокеты Беркли»).
Сокеты используются приложениями/ протоколами (прикладного уровня) для двунаправленных соединений.
ЯПВУ `Java` поддерживает сокеты для серверной и клиентской сторон. Серверная сторона ожидает запроса клиента, клиентская сторона инициирует запрос к серверу.

Констуктор класса Socket

`Socket()`

конструктор класса `Socket`; принимает параметры — строку (DNS- адрес, на месте этойстроки можно указать и объект `InetAdress`), `int` (номер порта); создает объект своего класса; возвращает ничтоже;
после создания сокета происходит соединение клиента с сервером (оно выполняется неявно, самим языком);

Методы класса Socket

`getInetAddress()`

используется для возврата объекта `InetAddress`; параметов не принимает; возвращает объект `InetAddress`, сопоставленный со своим объектом- сокетом; возвращает объект `InetAddress`;
если сокет не подключен, то возвращает `null`;

`getPort()`

используется для возврата порта; параметров не принимает; возвращает удаленный порт своего объекта- сокета; возвращает `int`;
если сокет не привязан, то возвращает `0`;

`getLocalPort()`

используется для возврата порта; параметров не принимает; возвращает локальный порт своего объекта- сокета; возвращает `int`;
если сокет не привязан, то возвращает `-1`;

`getInputStream()`

используется для возврата потока ввода; параметров не принимает; возвращает поток ввода, сопоставленный со своим объектом- сокетом; возвращает `InputStream`;

`getOutputStream()`

используется для возврата потока вывода; параметров не принимает; возвращает поток вывода, сопоставленный со своим объектом- сокетом; возвращает `OutputStream`;

`isConnected()`

используется для проверки соединения; параметров не принимает; проверяет, произошло ли соединение клиента с сервером; возвращает логическое значение (`true` – yes, `false` – no);

`connect()`

используется для установки соединения;

`isBound()`

используется для проверки привязан ли сокет к адресу; параметров не принимает; проверяет, привязан ли свой объект- сокет к адресу; возвращает логическое значение (`true` – yes, `false` – no);

`isClosed()`

используется для проверки закрыт ли сокет; параметров не принимает; проверяет, закрыт ли свой сокет; возвращает логическое значение (`true` – yes, `false` – no);

`close()`

используется для закрытия сокета; параметров не принимает; закрывает свой объект- сокет; возвращает ничтоже;

Класс URL

Этот класс используется для инкапсуллирования URL.

Конструктор класса URL

`URL()`

конструктор класса `URL`; принимает параметры — строку (содержит URL), либо строку (протокол), строку (имя домена), `int/ String` (номер порта);
создает объект своего класса;
возвращает ничтоже;
альтернативная форма конструктора принимает объект URL, и строку (содержит URL);

Класс URLConnection

Этот класс служит для доступа к атрибутам ресурса в сети.
Объекты этого класса создаются не конструктором, а методом `openConnection()` объекта `URL`: метод параметров не прнимает, создает объект `URLConnection`.
У класса `URLConnection` имеется ряд методов (нижеописанные методы работают с заголокими HTTP).
Метод `getContentLenght()` возвращает `int` (количество байт контента, `-1` – если длина не доступна), парамтров не принимает.
Метод `getContentLenghtLong()` возвращает `long` (количество байт контента, `-1` – если длина не доступна), парамтров не принимает.
Метод `getContentType()` возвращает заголовок `content-type` (если тип содержимого недоступен — то `null`).
Метод `getDate()` возвращает время ответа (в мс с 01.01.1970).
Метод `getExpiration()` возвращает время срока действия ресурса (в мс с 01.01.1970).
Метод `getHeaderField()` возвращает указанное параметром поле заголовка (параметр – `int`, указывает индекс поля, или строка — имя поля заголовка), возвращает строку (если заголовка по указанному индексу не обнаружено — то `null`).
Метод `getHeaderFieldKey()` возвращает указанный параметром ключ поля заголовка (параметр – `int`, указывает индекс поля), возвращает строку (если заголовка по указанному индексу не обнаружено — то `null`).
Метод `getHeaderFiels()` возвращает поля заголовка (ключи и значения), возвращает `Map`.
Метод `getLastMoified()` возвращает время последнего изменения ресурса (в мс с 01.01.1970).
Метод `getInputStream()` возвращает поток ввода.
Этот класс имеет подкласс `HttpURLConnection()` - объект этого класса создается тем же методом `openConnection()` - затем результат явно приводится к типу `HttpURLConnection`. Этот подкласс вводит вои методы.
Метод `getFollowRedirects()` определяет, последует ли автоматическая переадресация, возвращает логическое значение (`tue` – yes, `false` – no).
Метод `getRequestMethod()` возвращает метод запроса (`GET`, `POST`, …).
Метод `getResponseCode()` возвращает код ответа (`-1` – если не определен).
Метод `getResponseCode()` возвращает строку — сообщение ответа (`null` – если нет).
Метод `setFollowRedirects()` устанавливает автоматическую переадресацию (принимает логическое значение), возвращет ничтоже; этот метод статический.
Метод `setRequestMathod()` устанавливает метод запроса, возр- ет ничтоже.

Использование сквозного транспорта

Надежный транспорт не столь быстр как сквозной. Использование сквозного транспорта происходит с применением классов `DatagramSocket` (сокет, используемый для сквозного транспорта), `DatagramPacket` (передаваемый пакет).

Класс DatagramSocket

Конструктор класса DatagramSocket

`DatagramSocket()`

конструктор класса `DatagramSocket`; принимает параметры — либо ничего, либо int (номер порта), либо `int` (номер порта) и `InetAdress` (содержащий IP- адрес), либо `InetAddress` (содержащий IP- адрес и порт); создает объект своего класса; возвращает ничтоже;

Методы класса DatagramSocket

Это методы, общие для различных сокетов (`getInetAdress()`, `getLocalPort()`, `getPort()`, `isBound()`, `isConnected()`, `setSoTimeout()`, `close()`), и следующие методы.

`send()`

используется для отпраки пакета; принимает параметр — `DatagramPacket`; пересылает указанный объект; возвращает ничтоже;

`receive()`

используется для получения пакета; принимает параметр — `DatagramPacket`; ожидает получения указанного объекта; возвращает ничтоже;

Класс DatagramPacket

Конструктор класса DatagramPacket

`DatagramPacket()`

конструктор класса `DatagramPacket`; принимает параметры — массив байтов (полезная нагрузка), `int` (смещение в буфере, этот параметр можно опустить), `int` (размер буфера); создает объект своего класса; возвращает ничтоже;
альтернативная форма конструктора принимает параметры — массив байтов (полезная нагрузка), `int` (смещение в буфере, этот параметр можно опустить), `int` (размер буфера), `InetAddress` (содержащий IP- адрес), `int` (номер порта);

Методы класса DatagramPacket

`getAddress()`

используется для возврата адреса; параметров не принимает; возвращает адрес источника/приемника принимаемого/отправляемого пакета;
возвращает `InetAddress`;

`getData()`

используется для возврата полезной нагрузки; параметров не принимает; возвращает полезную нагрузку;
возвращает массив байтов;

`getLenght()`

используется для возврата длины полезной нагрузки; параметров не принимает; возвращает длину полезной нагрузки; возвращает `int`;

`getOffset()`

используется для возврата смещения; параметров не принимает; возвращает смещение; возвращает `int`;

`getPort()`

используется для возврата номера порта; параметровне принимает; возвращает номер порта; возвращает `int`;

`setAddress()`

используется для установки адреса; принимает параметр – `InetAddress` (содержащий IP- адрес); задает указанный адрес отправки; возвращает ничтоже;

`setData()`

используется для установки буфера; принимает параметры — массив байтов (на базе него формируется буфер), `int` (смещение), int (размер); руководствуясь переданными параметрами, создает буфер; возвращает ничтоже;

`setLenght()`

используется для установки длины пакета; принимает параметр – `int` (длина пакета); устанавлиает указанную длину пакета; возвращает ничтоже;

`setPort()`

используется для установки порта; принимает параметр – `int` (номер порта); устанавливает указанный порт; возвращает ничтоже;

Часть 3 (Сервлеты)

Сервлеты

Сервлеты — это небольшие программы, работающие на стороне сервера, и расширяющие его возможности.
Ранее, функциональность сервлетов реализовалась приложениями CGI (эти приложения могли быть написаны на разных языках, в том числе компиллируемых). Эти приложения обнаруживали недостаточную производительность; разумеется, они (будучи предназначенными для исполнения на стороне сервера) в интересах их производительности поддерживали многозадачность — но они поддерживали многозадачность на уровне процессов (что требовало соответствующих ресурсов).
Чтобы не сталкиваться с недостатками таких приложений и были разработаны средства разработки сервлетов.
Сервлеты исполняются на стороне сервера, и вызываются им по его логике. Жизненный цикл сервлета стартует по вызову его метода `init()` - этот метод выполняет инициализацию сервлета (и поэтому вызывается только на старте его жизненного цикла). Будучи инициализирован, сервлет готов к использованию; он вызывается сервером (по его логике) вызовом метода `service()` сервлета. Жизненный цикл сервлета завершается вызовом его метода `destroy()`.
Для разработки сервлетов употребляются соответствующие средства — средства сервера приложений `Glassfish` (входит в комплект поставки `Java EE`), либо контейнер сервлетов `Tomcat` (поддерживается Apache Software Foundation). Оба эти средства позволяют разарабатывать свои сервлеты на ЯПВУ `Java`. Подробнее об этих средствах — в соответствующей документации.