Графічний інтерфейс мови Java

В настоящее время практически все прикладные программы используют для взаимодействия с пользователем **графический интерфейс GUI** (Graphical User Interface – графический интерфейс пользователя).

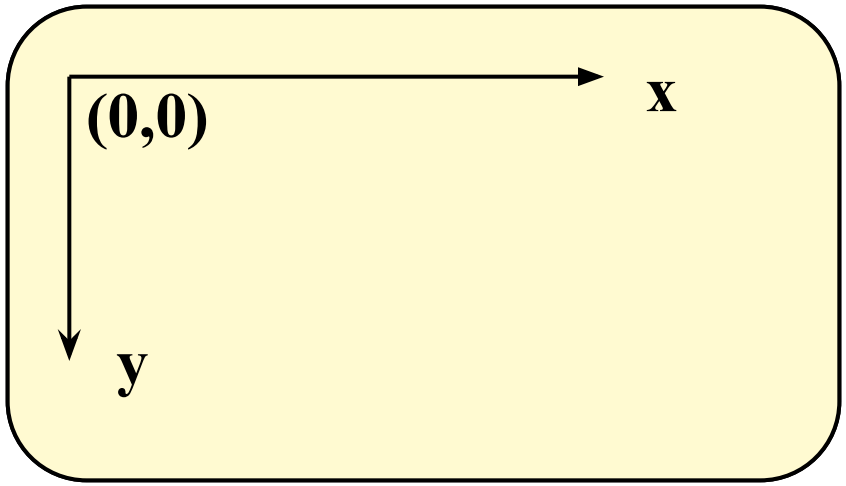
Одна из основных целей Java – создать независимую от платформы среду разработки GUI с **единым интерфейсом прикладных программ** – API (Application Program Interface).

Фирма Sun назвала этот единый API **инструментальными средствами управления абстрактными окнами** – AWT (Abstract Windowing Toolkit).

Засоби AWT в Java надають такі можливості:

- рисование линий и геометрических фигур различных цветов;
- выбор различных шрифтов и начертаний для вывода текста.
- создание обработчиков сообщений для взаимодействия с мышью и клавиатурой;
- создание анимации;
- загрузка и рисование изображений;
- создание и использование компонентов – элементов интерфейса: кнопок, списков, полос прокрутки, меню и т. п.;
- создание и использование контейнеров для обеспечения основных функций окон и диалогов;
- создание и использование менеджеров компоновки;
- работа с изображениями.

В Java для работы с изображениями используется обычная декартова система координат (**x, y**), где **x** – количество экранных точек от левой границы экрана, **y** – количество точек от верхней границы экрана. Левому верхнему углу соответствуют координаты **(0, 0)**



Каждый компонент AWT использует платформенно-зависимый (native) код для вывода себя на экран.

Начиная с версии JDK 1.1 была разработана группа «легких» (lightweight) компонент, которые являются платформенно-независимыми и используют объекты специального пакета графических элементов – библиотеки Swing.

Структура компонент AWT

Пакет AWT работает с графикой на двух различных уровнях. На нижнем уровне происходит работа с базовыми графическими функциями, в частности рисование фигур и вывод текста и различными устройствами ввода, например, мышью и клавиатурой. С помощью этих функций можно реализовать любые элементы GUI.

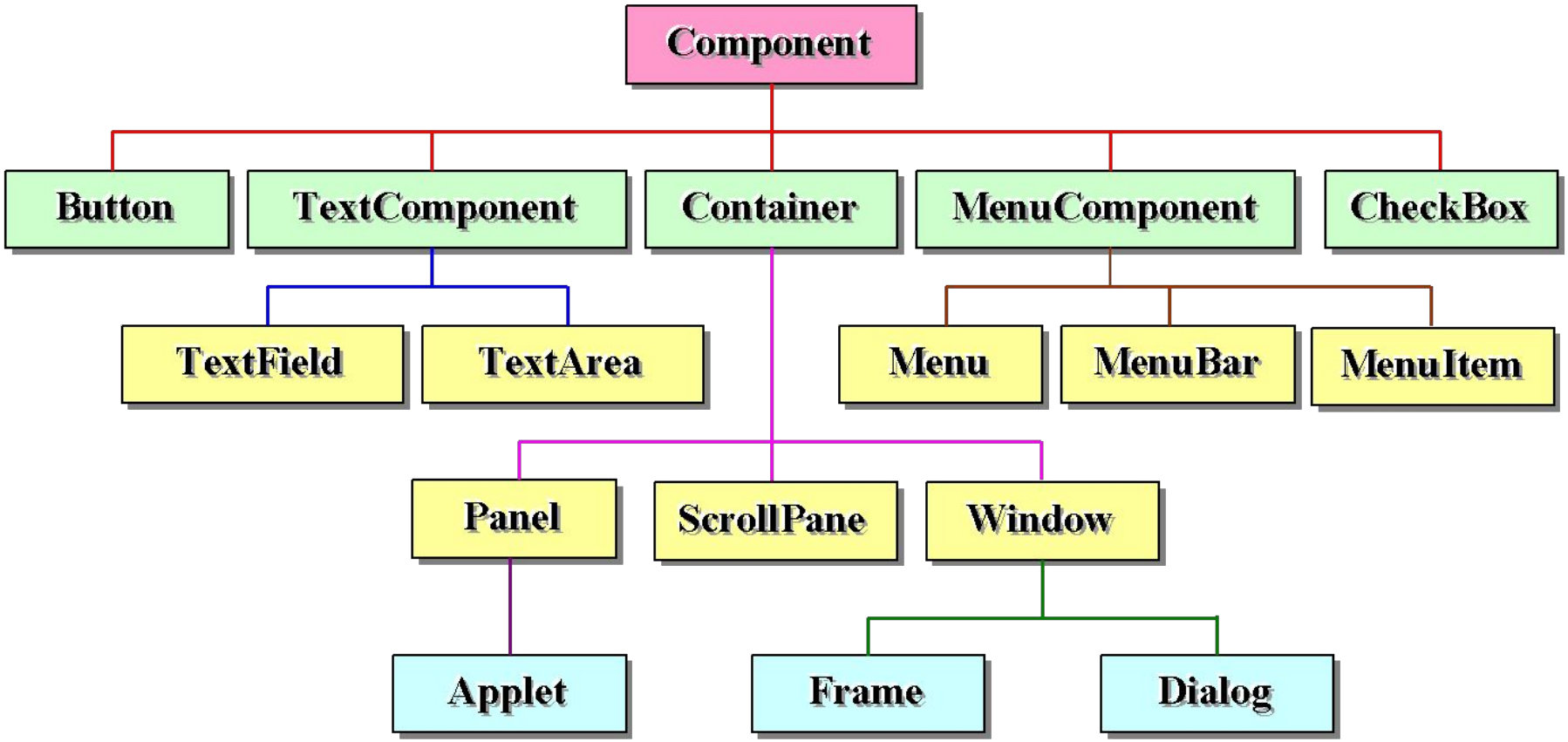
Однако в пакете **java.awt** определены также функции верхнего уровня, позволяющие определять стандартные компоненты GUI (например, кнопки или переключатели).

Базовым классом иерархии AWT Java является класс **Component**.

По функциональному назначению подклассы класса **Component** можно разделить на две группы. Первая группа – класс **Container** и производные от него классы обеспечивают работу пользователей с окнами, т.е. оконный интерфейс пользователя.

Остальные подклассы обеспечивают реализацию конкретных элементов пользовательского интерфейса, называемых элементами управления.

Основні класи ієрархії AWT в Java



Пакет AWT включает также несколько вспомогательных классов, которые представляют различные геометрические фигуры или их элементы. Это классы **Point**, **Dimension**, **Rectangle** и **Polygon**

Объект **Point** представляет точку (**x**, **y**) в координатной системе Java.

Этот объект можно создать с помощью одного из конструкторов:

```
public Point()
public Point(int x, int y)
public Point(Point p)
```

Методы класса **Point**:

```
public Point getLocation()
public double getX()
public double getY()
```

Поля класса **Point**:

```
public int x
public int y
```

Установка положения точки

```
public void setLocation(Point p)
public void setLocation(int x, int y)
public void setLocation(double x, double y)
```

Изменение значений координат **x** и **y** точки

```
public void move(int x, int y)
public void translate(int dx, int dy)
```

Класс **Dimension** представляет геометрические размеры: ширину и высоту.

Три конструктора класса **Dimension**:

```
public Dimension()
public Dimension(int width, int height)
public Dimension(Dimension d)
```

Методы класса **Dimension**:

```
public Dimension getSize()
public double getWidth()
public double getHeight()
```

Поля класса **Dimension**:

```
public int width
public int
height
```

Установка размеров объекта **Dimension**:

```
public void setSize(int width, int height)
public void setSize(double width, double height)
public void setSize(Dimension d)
```

Объект **Rectangle** представляет собой совокупность объектов **Point** и **Dimension**. Объект **Point** задает положение левого верхнего угла прямоугольника, а **Dimension** – его размеры.

Конструкторы:

```
public Rectangle(Point p, Dimension d)
public Rectangle(int x, int y, int width, int height)
public Rectangle()
public Rectangle(Rectangle r)
```

Установка новых параметров для прямоугольника

```
public void setRect(double x, double y, double width, double height)
public void setBounds(int x, int y, int width, int height)
```

Методы класса **Rectangle**:

```
public void grow(int h, int v)
public boolean intersects(Rectangle r)
public Rectangle intersection(Rectangle r)
public boolean contains(int x, int y)
public Rectangle union(Rectangle r)
public Rectangle add(Point pt)
public Rectangle add(int newx, int newy)
```

Класс **Polygon** определяет многоугольник

Конструкторы:

```
public Polygon()
public Polygon(int[] xpoints, int[] ypoints, int npoints)
```

Поля класса **Polygon**:

```
public int[] xpoints
public int[] ypoints
public int npoints
```

Методы класса **Polygon**:

```
public Rectangle getBounds()
public void translate(int deltaX, int deltaY)

public boolean contains(double x, double y)
public boolean contains(int x, int y)
public boolean contains(Point p)

public boolean contains(double x, double y, double w, double h)
public boolean intersects(double x, double y, double w, double h)
public void addPoint(int x, int y)
```

Клас Component

Класс **Component** – это абстрактный класс, в котором определено большое количество реализованных методов, общих для всех компонентов и контейнеров из AWT. Практически все они связаны с отображением компонентов и обработкой событий ввода.

Компоненты позволяют задавать множество различных параметров, например, основной и фоновый цвет, шрифт, а также флаг, указывающий, надо ли отображать компонент на экране.

Методы класса **Component**:

```
public void setVisible(boolean b)
```

в зависимости от значения параметра **b** включает (**true**) или выключает (**false**) отображение компонента на экране. При этом невидимый компонент по-прежнему существует. Метод **setVisible()** важен для окон, поскольку по умолчанию они создаются невидимыми.

public void setEnabled(boolean b)

отключает компонент, если значение параметра **b** равно **false** или включает компонент, если значение **b** равно **true**. Включенный компонент может реагировать на действия пользователя и генерировать события.

public boolean isEnabled()

проверяет состояние компонента (включен он или выключен).

public Dimension getMinimumSize()
public Dimension getMaximumSize()

возвращают соответственно минимально и максимально допустимые ширину и высоту компонента

public Dimension getPreferredSize()

возвращает оптимальные (предпочтительные) для компонента ширину и высоту

public Dimension getSize() — выдает текущий размер компонента

public int getWidth()
public int getHeight() — выдает ширину/высоту компонента

public Point getLocation() — получает позицию компонента относительно к содержащему компоненту в виде объекта **Point**

public Point getLocationOnScreen() — выдает позицию компонента относительно левого верхнего угла экрана компьютера

public Rectangle getBounds() — выдает обрамляющий объект **Rectangle** компонента

public void setLocation(int x, int y)
public void setLocation(Point p)

перемещают компонент в новое положение (координаты **x** и **y** указывают позицию левого верхнего угла компонента относительно родительского компонента).

public void setSize(int width, int height)
public void setSize(Dimension d)

устанавливают новые размеры компонента

public void setBounds(int x, int y, int width, int height)
public void setBounds(Rectangle r)

перемещают компонент и изменяют его размеры

Клас Container

- **панель (Panel)** – реализация класса **Container**. Панель сама по себе не является окном. Ее единственное назначение – расположение компонентов в пределах окна;
- **апплет (Applet)** – окно для вывода апплета в Web-браузере;
- **панель прокрутки (ScrollPane)** – панель с автоматической прокруткой большого компонента;
- **окно (Window)** – родительский контейнер для классов-контейнеров Frame и Dialog;
- **фрейм (Frame)** – Полнофункциональное окно со своим собственным заголовком и значком. Окна могут иметь меню и использовать курсоры нескольких различных форм;
- **диалоговое окно (Dialog)** – раскрывающееся окно, не настолько универсальное, как обычное окно.

От остальных компонентов контейнеры отличаются **возможностью содержать в себе другие (дочерние) компоненты**. Компонент помещается в контейнер путем обращения к одному из методов **add()** контейнера**:**

public Component add(Component comp)

добавление компонента **comp** в конец контейнера;

public Component add(Component comp, int index)

добавление компонента **comp** в контейнер в позиции **index**;

public void add(Component comp, Object constraints)

добавление объекта **constraints** в конец контейнера;

public void add(Component comp, Object constraints, int index)

добавление объекта **constraints** в контейнер в позиции **index**.

Методы класса **Container**:

public void remove(Component comp)
public void remove(int index)

удаляет компонент из контейнера по ссылке или по индексу

public void removeAll()

удаляет из контейнера все находившиеся в нем компоненты

public Container getParent()

возвращает ссылку на родительский компонент

public Component getComponent(int index)

позволяет получить ссылку на компонент по заданному индексу

public Component getComponentAt(int x, int y)

возвращает компонент, содержащий точку с заданными координатами **x** и **y**

public Component[] getComponents()

выдает все содержащиеся в контейнере компоненты

public int getComponentsCount()

возвращает количество компонентов

Клас Panel

Панель (Panel) применяется для объединения компонентов в группы. Класс **Panel** является суперклассом для класса **Applet**. Когда экранный вывод направляется апплету, он рисуется с помощью объекта **Panel**.

Конструктор:

```
Panel myPanel = new Panel();
```

Панель не содержит своих методов. Все ее методы унаследованы от классов **Component** и **Container**.

Панель может быть включена в состав другого контейнера с помощью метода **add()**, унаследованного от класса **Container**.

После добавления панели, можно задать ее позицию и размеры с помощью методов **setLocation()**, **setSize()** или **setBounds()** класса **Component**.

Панель может содержать одну или несколько других панелей, так что панели можно вкладывать друг в друга:

```
Panel mainPanel, subPanel1, subPanel2;
SubPanel1 = new Panel();  // Создать первую вложенную панель
subPanel2 = new Panel();  // Создать вторую вложенную панель
mainPanel = new Panel();  // Создать родительскую панель
mainPanel.add(subPanel1);     // Сделать subPanel1 дочерней
                              // по отношению к mainPanel
mainPanel.add(subPanel2);     // Сделать subPanel2 дочерней
                              // по отношению к mainPanel
```

Количество уровней вложенности панелей не ограничивается.

Вывод графического контекста

При создании компонента графического приложения автоматически формируется его **графический контекст** (graphics context). В контексте размещается область рисования и вывода текста и изображений. Все классы, связанные с рисованием находятся в пакете **java.awt**.

Управляет контекстом класс **Graphics** или класс **Graphics2D**. Поскольку графический контекст сильно зависит от конкретной графической платформы, эти классы сделаны абстрактными. Поэтому нельзя непосредственно создать экземпляры класса **Graphics** или **Graphics2D**. Однако каждая виртуальная машина Java реализует методы этих классов, создает их экземпляры для компонента и предоставляет объект класса **Graphics** с помощью метода класса **Component**:

```
public Graphics getGraphics()
```

Методы класса **Component**: **paint()**, **update()**, **repaint()** используются для отображения (рисования на экране) компонента.

public void paint(Graphics g)

рисует компонент на экране. В качестве параметра используется графический контекст **g** – объект класса **Graphics.** Этот метод должен переопределяться апплетом или графическим приложением.

public void repaint()

указывает, что компонент при первой возможности должен быть перерисован. Это рано или поздно (но не сразу) приведет к вызову метода **update().**

public void repaint(int x, int y, int width, int height)

заказывает перерисовку части компонента, находящейся в пределах заданного прямоугольника.

public void repaint(long tm)

требует произвести перерисовку компонента не позже, чем через **tm** миллисекунд

public void repaint(long tm, int x, int y, int width, int height)

требует перерисовать указанную часть компонента не позже, чем через **tm** миллисекунд.

public void update(Graphics g)

производит перерисовку компонента с использованием графического кон текста **g.**

В классе **Graphics** определены методы рисования для работы с контурными фигурами:

- **void drawLine (int x1, int y1, int x2, int y2)** – выводит линию от позиции, заданной первыми двумя целыми числами (координаты **x1 и y1),** до позиции, обозначенной координатами **x2** и **y2;**
- **void drawRect (int x, int y, int width, int height) –** выводит прямоугольник. Координаты **x** и **y** указывают верхний левый угол прямоугольника**, width** и **height** – соответственно ширину и высоту;
- **void draw3DRect (int x, int y, int width, int height, boolean raised) –** выводит подсвеченный трехмерный прямоугольник. Сам прямоугольник задается как и в методе **drawRect,** а булевская переменная **raised** указывает, должен ли прямоугольник быть поднят над фоном;
- **void drawRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)** – выводит прямоугольник с закругленными углами, вписанный в нормальный прямоугольник, заданный первыми четырьмя параметрами. Параметры **arcWidth** и **arcHeight** указывают ширину и высоту дуги для закругленных углов;

- **void drawOval (int x, int y, int width, int height)** – выводит овал, вписанный в прямоугольник, определенный как в методе **drawRect;**
- **void drawArc (int x, int y, int width, int height, int startAngle, int arcAngle)** – выводит дугу, вписанную в прямоугольник, определенный как в методе **drawRect.** Параметры **startAngle** и **arcAngle** указывают начальные и конечные углы, измеряемые в градусах. Отсчет углов начинается от центра правой стороны графической области. Положительные значения указывают направление вращения против часовой стрелки, а отрицательные – по часовой стрелке;
- **void drawPolygon (int []xPoints, int []yPoints, int nPoints) –** выводит многоугольник. Целочисленные массивы содержат координаты **x** и **y** для точек, составляющих многоугольник, а параметр **nPoints** указывает общее количество точек;
- **void drawPolygon (Polygon p) –** выводит многоугольник. Многоугольник задан объектом класса **Polygon;**
- **void drawPolyline(int[] xPoints, int[] yPoints, int nPoints) –** выводит последовательность соединенных между собой линий, концы которых задаются координатами **xPoints** и **yPoints, a** параметр **nPoints** указывает общее количество точек (количество линий на 1 меньше).

Використання кольорів

При излучении используется модель **RGB**, в которой в качестве основных цветов приняты **красный (red), зеленый (green)** и **синий (blue).** Черный цвет создается отсутствием излучения, а белый – смесью всех цветов.
Эта модель является основной при представлении цвета в Java.

Кроме того, может быть задана **прозрачность цвета (компонента alpha)** – от полностью непрозрачного до полностью прозрачного (невидимого) цвета. Эта составляющая проявляет себя при наложении одного цвета на другой.
Если **alpha** имеет максимальное значение, то цвет совершенно непрозрачен, предыдущий цвет не просвечивает сквозь него. Если alpha равна своему минимальному значению, то цвет абсолютно прозрачен, для каждого пикселя виден только предыдущий цвет.

Цвет задается как объект класса **Color** с помощью конструкторов этого класса:

```
public Color(int r, int g, int b, int a)
public Color(int r, int g, int b)
```

компоненты красного (**r**), зеленого **(g)**, синего **(b) цвета** и **alpha** (**a**) в виде целых чисел в диапазоне от **0** до **255.**

```
public Color(float r, float g, float b, float a)
public Color(float r, float g, float b)
```

задают компоненты цвета и **alpha** в виде вещественных чисел в диапазоне от **0.0f** до **1.0f**

```
public Color(int rgba, boolean hasalpha)
public Color(int rgb)
```

задает компоненты цвета и alpha в виде одного шестнадцатеричного числа. В битах 0-7 записывается синяя составляющая цвета, в битах 8-15 — зеленая, в битах 16-23 — красная. Для первого конструктора значение alpha задается в битах 24-31, если параметр **hasalpha** равен **true.**

Color(ColorSpace cspace, float[] components, float alpha)

позволяет создавать цвет не только в цветовой модели RGB, но и в других моделях, в частности в модели CMYK

public static getHSBColor(float h, float s, float b)

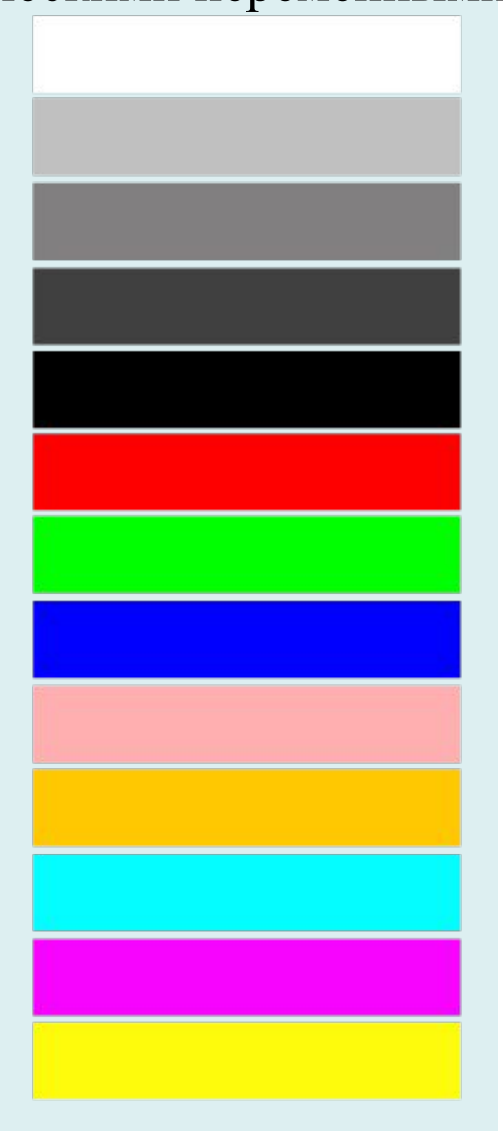
Метод класса **Color**, использующийся для создания цвета в модели **HSB** (Hue – оттенок, Saturation – насыщенность и Brightness – яркость) можно воспользоваться статическим

public static float[] RGBtoHSB(int r, int g, int b, float[] hsbvals)
public static int HSBtoRGB(int hue, int saturation, int brightness)

Методы класса **Color** возвращают цвет, преобразованный из цветовой модели RGB в HSB и обратно. Первый метод возвращает массив из трех значений: оттенка, насыщенности и яркости как в самой функции, так и в параметре **hsbvals**. Второй метод в битах 0-23 возвращаемого целого значения содержит компоненты цвета в модели **RGB**.

Цвета не всегда необходимо создавать, поскольку в классе **Color** есть набор предопределенных цветов, задаваемых статическими переменными типа **Color**:

- **Color.white** (белый цвет),
- **Color.lightGray** (светло-серый цвет)
- **Color.gray** (серый цвет),
- **Color.darkGray** (темно-серый цвет),
- **Color.black** (черный цвет),
- **Color.red** (красный цвет),
- **Color.green** (зеленый цвет),
- **Color.blue** (синий цвет),
- **Color.pink** (розовый цвет),
- **Color.orange** (оранжевый цвет),
- **Color.cyan** (сине-зеленый, или циановый, цвет).
- **Color.magenta** (пурпурный цвет),
- **Color.yellow** (желтый цвет),



Методы класса **Color** позволяют получить составляющие текущего цвета**:**

public int getRed() – значение красной компоненты **(от 0 до 255);**

public int getGreen() – значение зеленой компоненты (от **0 до 255);**

public int getBlue() – значение синей компоненты (от **0 до 255);**

public int getAlpha() – значение компоненты alpha (от **0 до 255);**

public int getRGB() – значение компонент в битах **31-0;**

public float[] getComponents(float[] compArray) – массив, содержащий значения красной, зеленой, синей компонент и компоненты alpha (значения в диапазоне от **0.0f** до **1.0f).**

public Color brighter()
public Color darker()

создают более яркий и более темный цвета по сравнению с текущим цветом

Создав цвет, можно задать его в качестве текущего цвета для рисования при помощи метода

public abstract void setColor (Color c)

```
public void setForeground(Color c)
public void setBackground(Color c)
```

Установка (обычно в методе **init()**) цвет переднего плана (тот цвет, которым будут рисоваться линии и выводится текст) и цвет фона

Если не задан цвет переднего плана, то в качестве текущего цвета используется черный цвет. Если не задан цвет фона, то используется цвет фона Web-браузера по умолчанию.

```
public Color getForeground()
public Color getBackground()
```

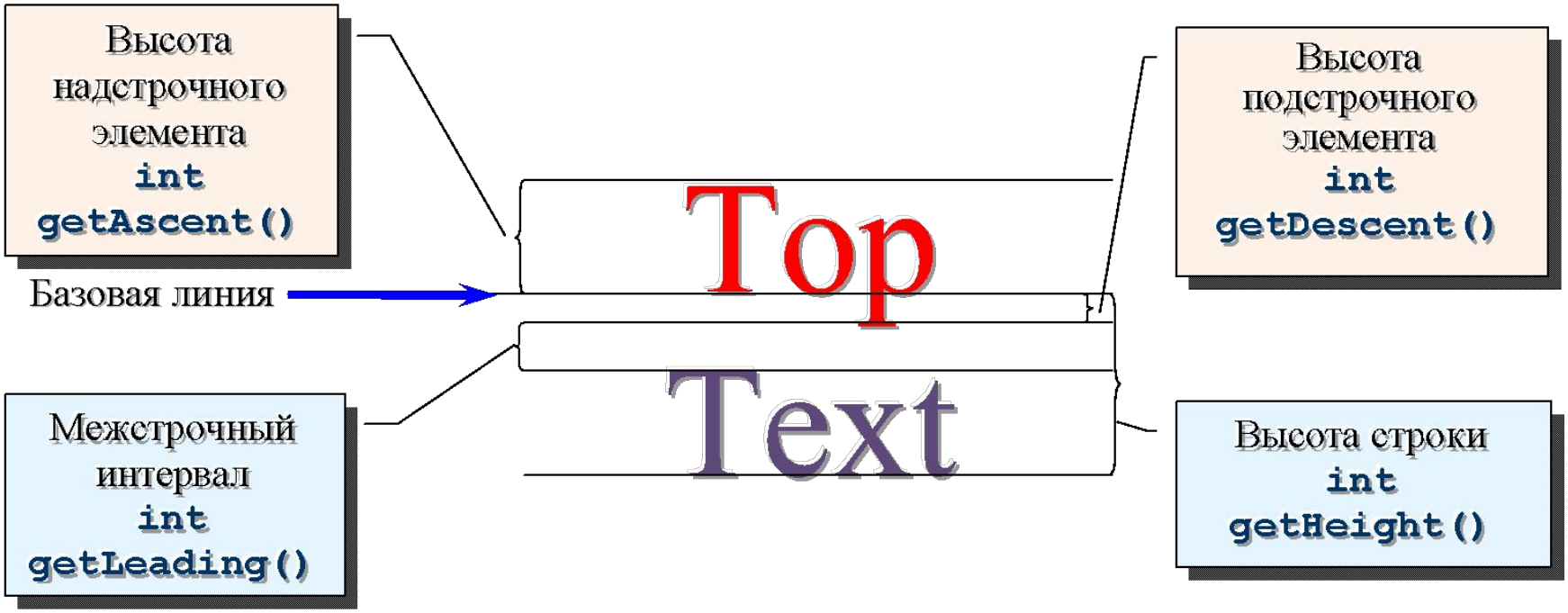
Получение текущих значений цветов переднего плана и фона

Все методы рисования фигур, кроме **drawArc(), drawLine()** и **drawPolyline()**, имеют соответствующие методы заполнения выводимой замкнутой фигуры в классе **Graphics**: **fillRect(), fill3Drect(), fillRoundRect(), fillOval(), fillPolygon().** Эти методы имеют те же параметры, что и соответствующие им методы рисования. Цвет заполнения фигуры предварительно задается с помощью метода **setColor()**

Виведення тексту

Поддержка вывода текста реализуется в Java с помощью методов классов **Graphics**, **Font** и **FontMetrics**.

Основные параметры, характеризующие расположение текста



Для вывода строки в окно апплета используется метод

```
public abstract void drawString(String str, int x, int y)
```

класса **Graphics**, которому в качестве параметров передается строка **str**, **а** также координаты **x** и **y** начала базовой линии. Следует отметить, что в отличие от вывода фигур, в которых значения **x** и **y** задаются для левого верхнего угла, для надписи в качестве начальной точки задается левый нижний угол, что следует иметь в виду при размещении надписи на экране.

Можно также вывести символы из массива символов или из массива байт с помощью методов класса **Graphics**:

```
public void drawChars(char charArray[], int offset, int length, int x, int y)
public void drawBytes(byte byteArray[], int offset, int length, int x, int y)
```

offset определяет позицию первого выводимого символа или байта относительно начала массива, а **length** определяет число выводимых символов или байт

Для вывода текста в программе на Java используется один из шрифтов, называемый стандартным шрифтом или шрифтом по умолчанию.
Для того, чтобы изменить шрифт выводимого текста задается в Java, его сначала надо определить как объект класса **Font** с помощью конструктора**:**

Font(String name, int style, int size)

При выводе текста логическим именам шрифтов и стилям сопоставляются физические имена шрифтов или имена семейств шрифтов. Это имена реальных шрифтов, имеющихся в графической подсистеме операционной системы.
Например, логическому имени **"serif"** может быть сопоставлено имя семейства **(**family) шрифтов **Times New Roman**, а в сочетании со стилями – конкретные физические имена **Times New Roman Bold**, **Times New Roman Italic**. Эти шрифты должны находиться в составе шрифтов графической системы той машины, на которой выполняется приложение.

Установка нового шрифта при выводе текста в методе **paint()** выполняется с помощью метода класса **Graphics**

```
public abstract void setFont(Font font)
```

С помощью методов класса **Font** можно получить характеристики текущего шрифта:

```
public String getName()
public int getStyle()
public int getSize()
```

Модель делегування подій в Java

Операционная система отслеживает все события, происходящие в ней. Система направляет эти события в соответствующие целевые объекты. Если, к примеру, пользователь щелкает по окну, система создает событие **mouse-down** и отсылает его данному окну для обработки. Затем это окно может выполнить некоторую операцию или просто передать событие обратно системе для обработки по умолчанию.

Начиная с JDK 1.1, в Java была создана новая модель обмена информацией о событиях, названная **моделью делегирования событий.**

Каждый элемент взаимодействия между интерфейсом пользователя и программой определяется как **событие.** Классы приложений выясняют свою заинтересованность в некотором ожидаемом компонентом событии путем опроса компонента и выдачи ему предложения поместить в список сведения о **блоке прослушивания** данного компонента (listener). Когда происходит некоторое событие, источник события уведомляет о нем зарегистрированные блоки прослушивания.

События образуют иерархию классов. Корнем этой иерархии является класс **EventObject**, расширяющий класс **Object**.

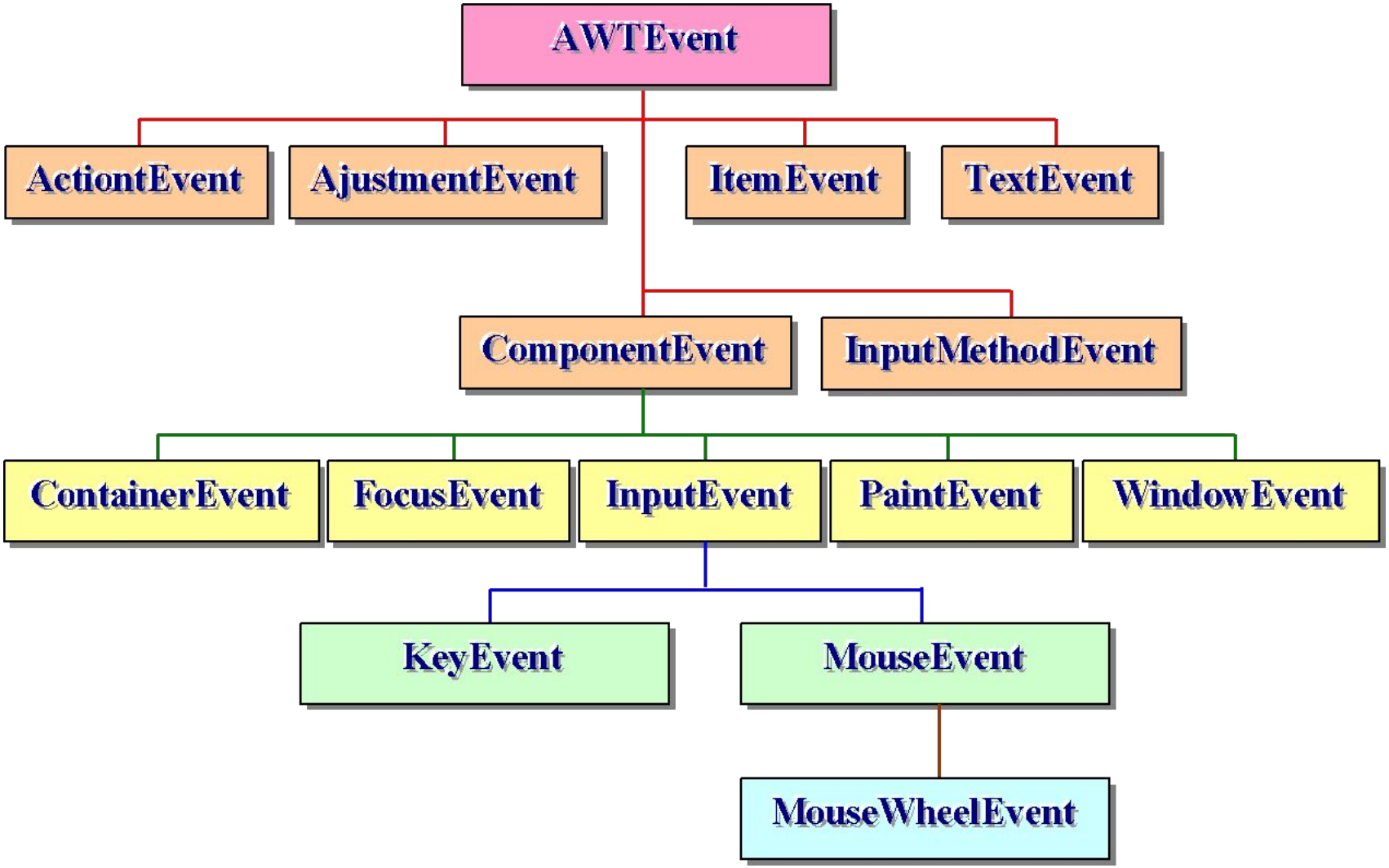
Для класса определены два метода, определяющие объект-источник события и строковое представление объекта-события:

```
public Object getSource()
```

```
public String toString()
```

События, связанные с графическим интерфейсом пользователя, в свою очередь, образуют ветвь в иерархии классов событий, корнем которой является класс **AWTEvent**, расширяющий класс **EventObject**.

Ієрархія подій для AWT



Типи подій

Существуют два различных типа событий – низкоуровневые и семантические.

Низкоуровневые события связаны с физическими аспектами интерфейса пользователя – щелчками мыши, нажатиями клавиш клавиатуры и т.д. Низкоуровневые события обрабатывает класс **ComponentEvent** и его потомки.

Семантические события строятся на базе низкоуровневых событий. Например, для выбора команды меню может потребоваться раскрыть список команд первым щелчком мыши, а затем вторым щелчком выбрать в нем требуемую команду. Эта последовательность низкоуровневых событий может быть объединена в одно семантическое событие. Все остальные события, кроме событий ветви **ComponentEvent**, являются семантическими.

Класи і інтерфейси обробки низькорівневих подій

В каждом классе событий определены соответствующие свойства и методы для обработки событий.

Почти каждый тип события, за исключением **PaintEvent** и **InputEvent**, имеет связанный с ним интерфейс или интерфейсы блоков прослушивания, в которых объявлены методы обработки данного типа события. Эти интерфейсы являются расширениями интерфейса **EventListener**, не определяющего никаких переменных и методов.

Класс ComponentEvent и интерфейс ComponentListener

Класс **ComponentEvent** обрабатывает события, связанные с изменением компонента. Причину изменения можно определить с помощью статических **final** переменных класса типа **int**.

public Component getComponent() возвращает компонент – источник события.

Класс ContainerEvent и компонент ContainerListener

Класс **ContainerEvent** обрабатывает события добавления или удаления компонента.

Методы класса возвращают компонент, на который действует событие и контейнер – источник события:

```
public Component getChild()
public Container getContainer()
```

Класс FocusEvent и интерфейс FocusListener

Класс **FocusEvent** обрабатывает события приобретения или потери фокуса.

```
public Component getOppositeComponent()
```

возвращает другой компонент, связанный со сменой фокуса

```
public boolean isTemporary()
```

возвращает **true,** если событие, связанное с изменением фокуса, является временным и **false** – в противном случае.

Класс WindowEvent и интерфейсы WindowListener, WindowStateListener, WindowFocusListener

Класс **WindowEvent** обрабатывает события, связанные с изменением состояния окна

Методы класса для события **WINDOW_STATE_CHANGED** возвращает новое или предыдущее состояние окна:

```
public int getNewState()
public int getOldState()
```

Возвращаемое значение можно сравнивать на равенство со следующими статическими целыми **final** переменными класса **Frame**:

- **NORMAL** – нормальное состояние окна;
- **ICONIFIED** – свернутое окно;
- **MAXIMIZED_HORIZ** – увеличение до максимального размера по горизонтали;
- **MAXIMIZED_VERT** – увеличение до максимального размера по вертикали;
- **MAXIMIZED_BOTH** – увеличение до максимального размера по горизонтали и вертикали.

public Window getOppositeWindow()

возвращает соответственно другое окно, которое затронуло данное событие

public Window getWindow()

возвращает окно – источник события

Интерфейс **WindowListener** содержит объявления методов обработки событий окна

Класс InputEvent

Класс **InputEvent** является корневым классом для всех событий ввода уровня компонента. События ввода направляются блокам прослушивания до их обычной обработки источником, который вызвал данное событие. Это позволяет блокам прослушивания и подклассам компонент «потреблять» событие так, чтобы источник не обрабатывал их обычным образом.

Класс содержит следующие статические **final** переменные типа **int,** которые используются при обработки событий мыши и клавиатуры.

public void consume()

«потребляет» событие так, чтобы оно не обрабатывалось обычным образом

public boolean isConsumed()

проверяет, не использовалось ли потребление события

public int getModifiers()

возвращает модификатор для данного события

public static String getModifiersExText(int modifiers)

получает текстовое значение для нажатых клавиш модификаторов

public int getModifiersEx()

позволяет получить расширенные значения модификаторов, которые представляют состояние всех клавиш-модификаторов и кнопок мыши

public long getWhen()

возвращает отметку о времени возникновения события

public boolean isAltDown()
public boolean isControlDown()
public boolean isShiftDown()

позволяют определить были ли нажаты соответственно клавиши **Alt, Ctrl** или **Shift** во время возникновения события

Класс KeyEvent и интерфейс KeyListener

Класс **KeyEvent** обрабатывает события клавиатуры

В классе **KeyEvent** определены коды всех клавиш в виде констант, называемых виртуальными кодами клавиш (virtual key codes), например, **VK_F1, VK_SHIFT, VK_A, VK_B, VK_PLUS.** Они перечислены в документации к классу **KeyEvent.** Фактическое значение виртуального кода зависит от языка и раскладки клавиатуры.

public char getKeyChar() — выводит значение символа

public int getKeyCode() — выводит значение кода символа

getKeyLocation() — выводит положение клавиши на клавиатуре

public static String getKeyText(int keyCode) — возвращает строку, описывающую клавишу

public static String getKeyModifiersText(int modifiers)

возвращает текст, описывающий клавишу-модификатор

public boolean isActionKey()

проверяет, не является ли клавиша, вызвавшая событие клавишей «действия»

public void setKeyChar(char keyChar)
public void setKeyCode(int keyCode)
public void setModifiers(int modifiers)

позволяют установить соответственно значение символа, кода и клавиши-модификатора

Класс **KeyEvent** наследует также все свойства и методы класса **InputEvent.**

Класс MouseEvent и интерфейсы MouseListener, MouseMotionListener

Класс **MouseEvent** обрабатывает события мыши

public int getButton() позволяет определить, какая кнопка была нажата

public int getClickCount() возвращает количество щелчков кнопки

public Point getPoint()
public int getX()
public int getY()

определяет точку экрана, в которой произошло событие мыши или отдельно координаты **x** и **y** точки

public boolean isPopupTrigger()

позволяет определить, является ли событие мыши событием вызова контекстного («всплывающего») меню

public static String getMouseModifiersText(int modifiers)

возвращает строку, идентифицирующую клавишу-модификатор (например, **"Shift"**), если она была нажата одновременно с кнопкой мыши

public void translatePoint(int x, int y)

переводит точку координаты точки на экране, в которой произошло событие мыши, в новую позицию с горизонтальным смещением **x** и вертикальным смещением **y**.

Класс MouseWheelEvent и интерфейс MouseWheelListener

Класс **MouseWheelEvent** обрабатывает события, связанные с вращением колесика мыши.

public int getScrollType()	позволяет получить тип вращения колесика мыши
public int getScrollAmount()	возвращает количество «единиц», которое необходимо прокрутить в ответ на это событие
public int getWheelRotation()	возвращает количество «щелчков», прокрученных колесиком мыши
public int getUnitsToScroll()	возвращает и количество «единиц» прокрутки

Обробка подій в Java

Чтобы блок прослушивания мог фиксировать и обрабатывать события он должен быть включен с помощью методов добавления блоков прослушивания в класс (эти методы определены в классе **Component**).

Методы включения блоков прослушивания имеют следующий общий формат:

```
public void add**** (**** l)
```

**** – имя соответствующего блока прослушивания.

Например включение в класс блока прослушивания для событий клавиатуры реализуется с помощью метода

```
public void addKeyListener (KeyListener l).
```

Для обработки события в каком либо классе необходимо:

1. Сделать доступным методы, определенные в соответствующем интерфейсе или интерфейсах, объявив класс как реализацию (**implementation**) соответствующего интерфейса или интерфейсов.

2. Включить необходимые блоки прослушивания с помощью соответствующего метода **add******.

3. Описать в программе все методы всех реализуемых интерфейсов (если какой-либо метод не используется в классе, он объявляется с пустым телом – {}). В методах интерфейсов обычно используются методы и свойства соответствующего класса обработки событий.

Скелет программы для обработки событий клавиатуры:

```
class KeyTest implements KeyListener
{
...
addKeyListener(имя-объекта-KeyListener);
...
public void keyPressed(KeyEvent e)
{
      // Обработка события нажатия клавиши или пусто
}
public void keyReleased(KeyEvent e)
{
      // Обработка события отпускания клавиши или пусто
}
public void keyTyped(KeyEvent e)
{
// Обработка ввода символа или пусто
}
```

Для того, чтобы в классе можно было описывать только те методы, которые необходимо, в JDK были введены специальные абстрактные классы – **адаптеры**, содержащие объявления тех же методов, что и соответствующие блоки прослушивания. Имена этих классов формируются так же, как и блоков прослушивания, только вместо суффикса **Listener** в них используется суффикс **Adapter**, например, **KeyAdapter** или **MouseAdapter**.

Если создать внутри данного класса подкласс, расширяющий класс соответствующего адаптера, то внутри него можно переопределить только те методы адаптера, которые необходимо использовать. В этом случае в соответствующем методе **add****** в качестве параметра можно задать экземпляр созданного подкласса, например

```
addKeyListener(new myKeyAdapter());
```

Более компактной записью, которая часто используется программистами на Java, является непосредственное определение безымянного внутреннего класса в параметре соответствующего метода **add******