# import java.awt.\*;

# Вступ

Програми являють собою послідовність дій з умовного і безумовними переходами. Програміст мислить у термінах дій і вибудовує послідовності дій в складніші макро-дії (процедури), коли він пише програми, тримає в голові наявний функціонал - безліч дій, які на поточний момент реалізовано в програмі і бібліотеках. Якщо програміст використовуватиме тільки мову програмування, йому доведеться відволікатися на безліч дрібних деталей. Тому, щоб оптимізувати роботу і не писати код з нуля, слід використовувати набори готових модулів, які дозволяють розв’язувати найпоширеніші завдання.

У програмуванні **стандартна бібліотека** ([англ.](https://www.wiki.uk-ua.nina.az/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0.html) *standard library*) — це бібліотека, що доступна в усіх реалізаціях даної мови програмування, тобто – це збірки модулів та різних компонентів, які полегшують процес створення нової програми. Залежно від поставленого завдання, застосовуються різні комплекси та елементи коду. Сучасні розробники повинні використовувати бібліотеки, тому вони скорочують кількість помилок. Бібліотеки використовують програмісти для полегшеного процесу розробки софту. Зміст такої бібліотеки зазвичай описано в специфікації мови, однак також він може частково або повністю визначатися більш неформальними практиками програмістів, що користуються нею. Зазвичай користувачі мови програмування вважають стандартну бібліотеку її частиною, хоча можливо, що розробники ставилися до неї як до окремого об'єкту. В багатьох мовах специфікацією визначається базовий функціонал, який повинен бути присутній в усіх реалізаціях, в доповненні до інших можливостей, реалізація яких необов'язкова. Таким чином, межа між мовою програмування та її бібліотеками індивідуальна для кожної мови. Дійсно — деякі мови розроблено таким чином, що значення певних синтаксичних конструкцій навіть не може бути описано без посилань до стандартної бібліотеки. При написанні софту існує безліч стандартних елементів: кнопки, поле з пошуком, програвачі. Розробнику не потрібно окремо розписувати кожен елемент з нуля. При підключенні модулів частина коду переноситься в нову програму. Залежно від мови програмування існують багато бібліотек, які спеціалізуються на :

* використання математичних операцій;
* графічні модулі;
* компоненти для роботи з файлами;
* модулі для підключення та використання мережі;
* елементи шифрування та захисту даних.

Чим їх більше, тим популярніша мова. Незважаючи на велику кількість різних компонентів, залежно від популярності мови, розробники постійно випускають нові модулі з кодом. Чим їх більше, тим мова вважається розвиненішою. Умовно можна поділити на відкриті та комерційні проекти. Відкриті — у вільному доступі. Будь-який розробник може запропонувати виправлену версію коду. Комерційні — створюються у студіях для продажу. Основними творцями є штатний програміст та тестувальник. Штатні працівники можуть написати власні модулі під проект. Далі їх можна використовувати як шаблон до стороннього софту.

Також компоненти діляться способом підключення до мови. Розрізняють 2 типи: динамічні та статичні. Динамічні — виконані у вигляді файлу, в якому знаходиться код. Починає роботу під час запуску або експлуатації програми. Розробник може будь-якої миті коригувати дані та оновити софт. Статичні — представлені у вигляді вихідного коду, розташованого всередині програми. На відміну від динамічних, провести зміни досить складно. Для цього доведеться перейти в дерево софту, провести зміни, заново запустити вихідний код. Вибір типу підключення залежить від цілей розробника. Якщо потрібно часто змінювати дані та вносити правки, рекомендується використовувати динамічну модель. Статичні надійніші. Сучасний софт написаний великою кількістю коду. Використання стандартних елементів з нуля збільшує час створення програми. Також, якщо розробник не використовує окремі компоненти, код програми стає об'ємним і масивним. До переваг використання бібліотек відносять:

* економія часу;
* структурування інформації;
* полегшення розробки;
* зменшення кількості помилок у коді.

Щоб подзвонити, людина не створює кожного разу телефон, а бере готовий виріб і дзвонить. Такий самий принцип закладено у програмуванні. Розробнику не потрібно думати про те, із чого складається кожен елемент. Йому необхідно реалізувати бізнес-ідею, зробити якісний додаток, який пройде тести. Також використання самостійного написання кожного елемента збільшує ймовірність помилки. Використання бібліотек полегшує роботу з кодом розробки власного софта. Розробнику не потрібно думати про те, як написати кнопку або інші елементи, що часто зустрічаються. Достатньо прописати та підключити модуль, який відкриє цілий масив із потрібними даними.

## 1. Структура бібліотеки

Часто більшість систем розробки додатків на мові програмування опираються на набір стандартних бібліотечних функцій або бібліотеки класів. У комплекті з усіма засобами розробки Java поставляються досить розвинені бібліотеки класів, які суттєво полегшуютьпроцес програмування. У бібліотеці Java інформацію розбито по ("packages"), у кожному з яких протестований код по певному напрямку. Наприклад, є такі пакети

* java.lang (основний пакет мови Java);
* java.util;
* java.io;
* java.net;
* java.math;
* java.awt;
* java.swing;
* java.sql;
* java.time тощо;

Програма на Java є набір *пакетів* (packages). Кожен *пакет* може містити вкладені *пакети* , тобто утворюють ієрархічну систему. Крім того, *пакети* можуть містити класи та інтерфейси і таким чином групують типи. Це необхідно відразу для кількох цілей. По-перше, суто фізично неможливо працювати з великою кількістю класів, якщо вони "скидані в купу". По-друге, модульна декомпозиція полегшує проектування системи. До того ж, як буде показано нижче, існує спеціальний рівень доступу, що дозволяє типу з одного пакета тісно взаємодіяти один з одним, ніж з класами з інших пакетів. Таким чином, за допомогою пакетів здійснюється логічне угрупування типів. Відомо, що велика зв'язність системи, тобто середня кількість класів, з якими взаємодіє кожен клас, помітно ускладнює розвиток та підтримку такої системи. Використовуючи пакети, набагато простіше організувати ефективну взаємодію підсистем один з одним. Нарешті, кожен пакет має власний простір імен, що дозволяє створювати однойменні класи у різних пакетах. Отже, розробникам витрачати час на вирішення конфлікту імен не потрібно.

*Елементами пакета* є *вкладені пакети* та *типи* (класи та інтерфейси). Однойменні *елементи* заборонені, тобто не може бути *однойменних класу* та *інтерфейсу* , або *вкладеного пакета* та *типу*. Інакше виникне помилка компіляції. Наприклад, у пакет **java.awt** входив вкладений пакет **image** та 46 класів та інтерфейсів. Складене ім'я будь-якого елемента пакета – це складове ім'я пакета плюс просте ім'я елемента. Наприклад, для класу **Object** у пакеті **java.lang** складним ім'ям буде **java.lang.Object**, а для пакету **image** у пакеті **java.awt** — **java.awt.image.** Ієрархічна структура пакетів була введена для зручності організації пов'язаних пакетів, проте вкладені пакети, або сусідні, тобто вкладені в той самий пакет, не мають жодних додаткових зв'язків між собою, крім обмеження на розбіжність імен. Найпростішим способом організації пакетів та типів є звичайна файлова структура. Розглянемо виразний приклад, коли всі пакети, вихідний та бінарний код розташовуються в одному каталозі та його підкаталогах. У цьому кореневому каталозі має бути папка **java**, відповідна основному пакету мови, а ній, своєю чергою, вкладені папки **applet, awt, io, lang, net, util**.

У мові Java всі класи відбуваються від класу Object, і, відповідно, успадковують методи цього класу. Деякі бібліотеки класів підключаються автоматично і їх називають вбудованими. До таких належить, зокрема, бібліотека з назвою java.lang. Інші бібліотеки класів необхідно підключати в початковому тексті програми Java явно за допомогою оператора import. Серед вбудованих класів слід виділити клас Math, призначений для виконання математичних операцій, таких як обчислення синуса, косинуса і тангенса. Передбачені також класи для виконання запуску процесів і потоків, управління системою безпеки, а також для вирішення інших системних задач. Бібліотека вбудованих класів містить дуже важливі класи для роботи з винятками. Ці класи потрібні для обробки помилкових ситуацій, які можуть виникнути при роботі додатків Java.

## 2. Бібліотеки класів

Перерахуємо спільні бібліотеки:

* Бібліотека класів **JAVA.UTIL**

Бібліотека класів java.util дуже корисна при складанні додатків, тому що в ній є класи для створення таких структур, як динамічні масиви, стеки і словники. Є класи для роботи з генератором псевдовипадкових чисел, для розбору рядків на складові елементи (токени), для роботи з календарною датою і часом.

* Бібліотека класів **JAVA.IO**

У бібліотеці класів java.io зібрані класи, що мають відношення до вводу та виводу даних через потоки. Використовуючи ці класи, можна працювати не тільки з потоками байт, але також і з потоками даних інших типів, наприклад числами int або текстовими рядками.

* Бібліотека класів **JAVA.NET**

Як приклад застосування, складеного на мові програмування Java і орієнтованого на роботу в мережі Internet, можна привести гру Java Color Lines — це мережева версія відомої гри Lines, яка виконана у вигляді декількох аплетів, взаємодіючих між собою і між сервером Web, на якому вони розташовані. Так як список імен гравців і досягнутих ними результатів зберігається на сервері, ви можете взяти участь в світовому турнірі, зігравши з гравцями з різних країн.

* Бібліотека класів **JAVA.AWT.IMAGE**

У середовищі будь-якої операційної системи робота з графічними зображеннями є досить складним завданням. В операційній системі Windows для цього застосовується графічний інтерфейс GDI. Якщо ви будете малювати графічні зображення в середовищі OS / 2 або X-Windows, вам, очевидно, доведеться використовувати інший програмний інтерфейс. Велику складність також викликає розбір заголовків графічних файлів, так як вони можуть мати різний формат і іноді містять неправильну або суперечливу інформацію.

Програмуючи на Java, малювання і обробка графічних зображень виконується набагато простіше, оскільки є можливість скористатись спеціально бібліотекою класів java.awt.image. Крім широкої різноманітності і зручності визначених у ній класів і методів, слід зазаначити здатність цієї бібліотеки працювати з графічними зображеннями в форматі GIF. Цей формат широко використовується в Internet, так як він дозволяє стискати файли графічних зображень у багато разів без втрати якості за рахунок усунення надмірності.

* Бібліотека класів **JAVA.AWT.PEER**

Бібліотека класів java.awt.peer служить для підключення компонент AWT (наприклад, кнопок, списків, полів редагування текстової інформації, перемикачів і так далі) до реалізацій, що залежать від платформи, в процесі створення цих компонент.

* Бібліотека класів **JAVA.APPLET**

Із назви бібліотеки можна зрозуміти, що бібліотека класів java.applet інкапсулює поведінку аплетів Java. Створювати свої аплети, необхідно використовувати клас Applet, розташований в цій бібліотеці класів. Додатково в бібліотеці класів java.applet визначені класи для програвання звукових фрагментів.

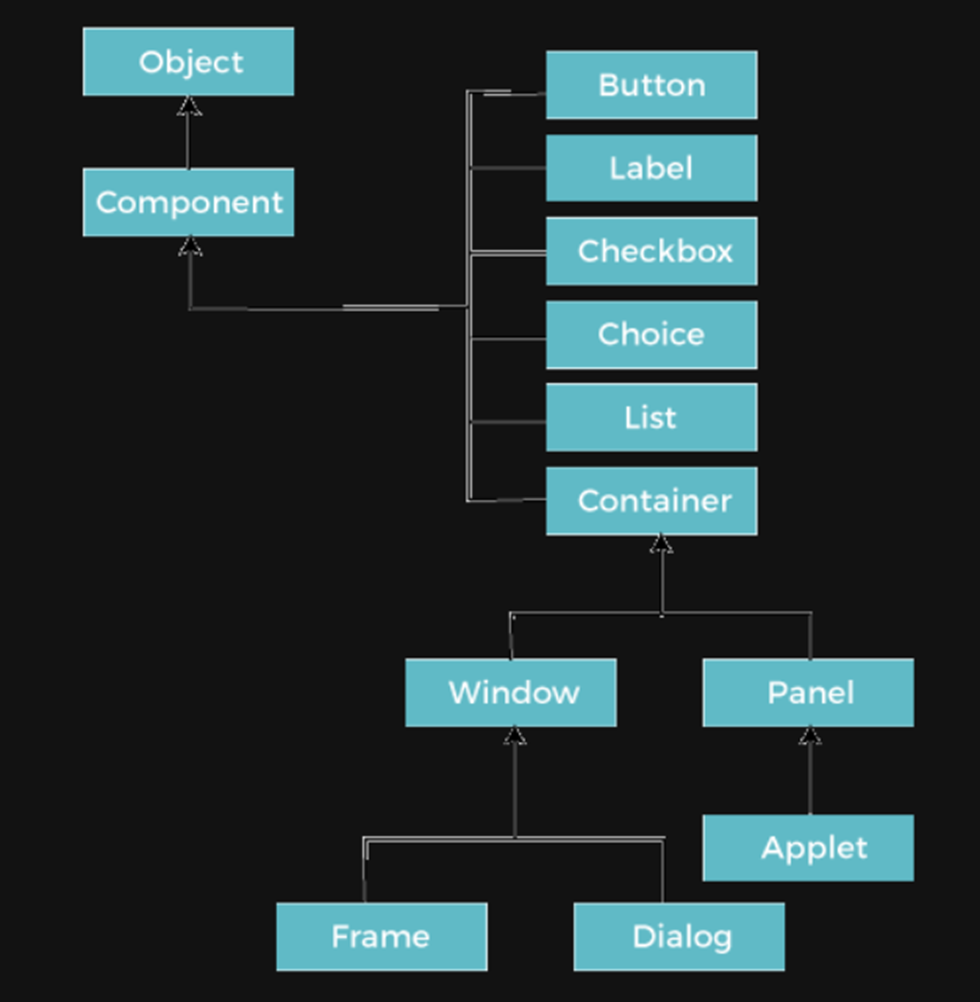
* Бібліотека класів **JAVA.AWT**

Для створення призначеного для користувача інтерфейсу аплети Java можуть і повинні використовувати бібліотеку класів java.awt. AWT — це скорочення від Abstract Window Toolkit (інструментарій для роботи з абстрактними вікнами).

Зупинюсь детальніше на основних функціях бібліотеки **JAVA.AWT.**

## 3. Бібліотека класів JAVA.AWT

Графічний інтерфейс користувача (GUI) пропонує взаємодію користувача через деякі графічні компоненти. Наприклад, наша базова операційна система також пропонує графічний інтерфейс користувача через вікно, кадр, панель, кнопку, текстове поле, TextArea, список, комбінований список, мітку, прапорець тощо. Усі вони відомі як компоненти. Використовуючи ці компоненти, ми можемо створити інтерактивний інтерфейс користувача для програми. GUI надає результат кінцевому користувачеві у відповідь викликані події. GUI повністю базується на подіях. Наприклад, натискання кнопки, закриття вікна, відкриття вікна, введення чогось у текстової області тощо. Ці дії відомі як події. Інтерфейс користувача полегшує використання програми кінцевим користувачем. Це також робить їх цікавими.



*Рис. 1* Схема класів компонент.

Як уже зазначалось усі елементи, такі як кнопка, текстові поля, смуги прокручування тощо, називаються компонентами. У Java AWT існують класи для кожного компонента, як показано на схемі вище (рис. 1). Щоб розмістити кожен компонент у певному місці на екрані, нам потрібно додати його до контейнера. Контейнер — це компонент в AWT, який може містити інші компоненти, такі як кнопки, текстові поля, мітки тощо. Класи, які розширюють клас Container, відомі як контейнери, такі як Frame, Dialog і Panel. По суті, це екран, на якому компоненти розміщені в певних місцях. Таким чином, він містить і контролює розташування компонентів.

У Java AWT є чотири типи контейнерів:

* **Window** (вікно);
* **Panel** (панель);
* **Frame** (рамка);
* **Dialog** (діалог).

Вікно — це контейнер, який не має рамок і рядків меню. Слід використовувати фрейм, діалог або інше вікно для створення вікна. Нам потрібно створити екземпляр класу Window, щоб створити цей контейнер. Панель — це контейнер, який не містить рядка заголовка, межі або рядка меню. Це універсальний контейнер для зберігання компонентів. Він може мати інші компоненти, такі як кнопка, текстове поле тощо. Екземпляр класу Panel створює контейнер, до якого ми можемо додавати компоненти. Рамка — це контейнер, який містить рядок заголовка та межу, а також може мати рядки меню. Він може містити інші компоненти, такі як кнопка, текстове поле, смуга прокручування тощо. Frame є найбільш широко використовуваним контейнером під час розробки програми AWT.

## Популярні методи класу Window

* **void addWindowListener(WindowListener l)**

Adds the specified window listener to receive window events from this window.

* **List<Image> getIconImages()**

Returns the sequence of images to be displayed as the icon for this window.

* **List<Image> getIconImages()**

Returns the sequence of images to be displayed as the icon for this window.

* **WindowListener[] getWindowListeners()**

Returns an array of all the window listeners registered on this window.

* **static Window[] getWindows()**

Returns an array of all Windows, both owned and ownerless, created by this application.

* **void paint(Graphics g)**

Paints the container.

* **void reshape(int x, int y, int width, int height)**

Deprecated. As of JDK version 1.1, replaced by setBounds(int, int, int, int).

* **void setCursor(Cursor cursor)**

Set the cursor image to a specified cursor.

* **void toBack()**

If this Window is visible, sends this Window to the back and may cause it to lose focus or activation if it is the focused or active Window.

## Panel

* ***Конструктор для класу Panel()***

Creates a new panel using the default layout manager.

* **Panel(LayoutManager layout)**

Creates a new panel with the specified layout manager.

## Методи класу

* **void addNotify()**

Creates the Panel's peer.

* **AccessibleContext getAccessibleContext()**

Gets the AccessibleContext associated with this Panel.

# Frame

## Конструктор класу Frame

* **AccessibleContext getAccessibleContext()**

Gets the AccessibleContext associated with this Panel.

* **Frame(GraphicsConfiguration gc)**

Constructs a new, initially invisible Frame with the specified GraphicsConfiguration.

* **Frame(String title)**

Constructs a new, initially invisible Frame object with the specified title.

* **Frame(String title, GraphicsConfiguration gc)**

Constructs a new, initially invisible Frame object with the specified title and a GraphicsConfiguration.

## Популярні методи класу

* **void addNotify()**

Makes this Frame displayable by connecting it to a native screen resource.

* **int getCursorType()**

Deprecated. As of JDK version 1.1, replaced by Component.getCursor().

* **static Frame[] getFrames()**

Returns an array of all Frames created by this application.

* **MenuBar getMenuBar()**

Gets the menu bar for this frame.

* **boolean isResizable()**

Indicates whether this frame is resizable by the user.

* **boolean isUndecorated()**

Indicates whether this frame is undecorated.

* **void setMaximizedBounds(Rectangle bounds)**

Sets the maximized bounds for this frame.

* **void setMenuBar(MenuBar mb)**

Sets the menu bar for this frame to the specified menu bar.

## Dialog Основні методи конструктора Dialog

* **Dialog(Dialog owner)**

Constructs an initially invisible, modeless Dialog with the specified owner Dialog and an empty title.

* **Dialog(Dialog owner, String title)**

Constructs an initially invisible, modeless Dialog with the specified owner Dialog and title.

* **Dialog(Dialog owner, String title, boolean modal)**

Constructs an initially invisible Dialog with the specified owner Dialog, title, and modality.

* **Dialog(Dialog owner, String title, boolean modal, GraphicsConfiguration gc)**

Constructs an initially invisible Dialog with the specified owner Dialog, title, modality and GraphicsConfiguration.

* **Dialog(Frame owner)**

Constructs an initially invisible, modeless Dialog with the specified owner Frame and an empty title.

* **Dialog(Frame owner, boolean modal)**

Constructs an initially invisible Dialog with the specified owner Frame and modality and an empty title.

* **Dialog(Frame owner, String title, boolean modal, GraphicsConfiguration gc)**

Constructs an initially invisible Dialog with the specified owner Frame, title, modality, and GraphicsConfiguration.

## Основні методи класу

* **void addNotify()**

Makes this Dialog displayable by connecting it to a native screen resource.

* **AccessibleContext getAccessibleContext()**

Gets the AccessibleContext associated with this Dialog.

* **Dialog.ModalityType getModalityType()**

Returns the modality type of this dialog.

* **String getTitle()**

Gets the title of the dialog.

* **void hide()**

Deprecated. As of JDK version 1.5, replaced by setVisible(boolean).

* **boolean isModal()**

Indicates whether the dialog is modal.

* **boolean isResizable()**

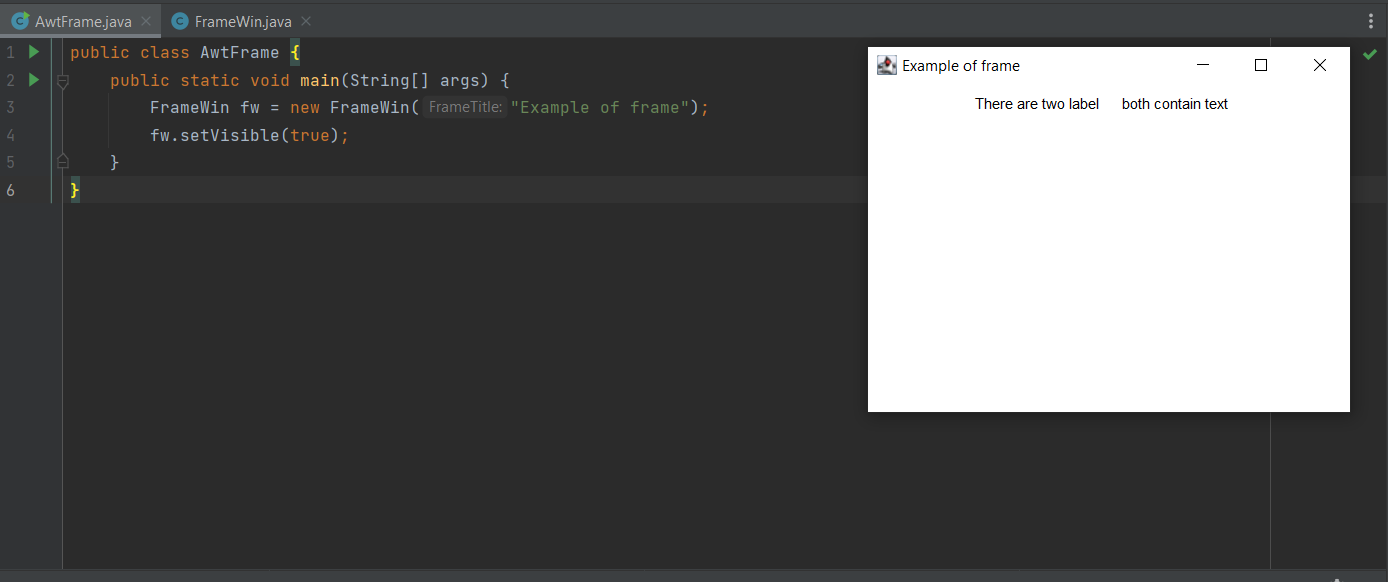
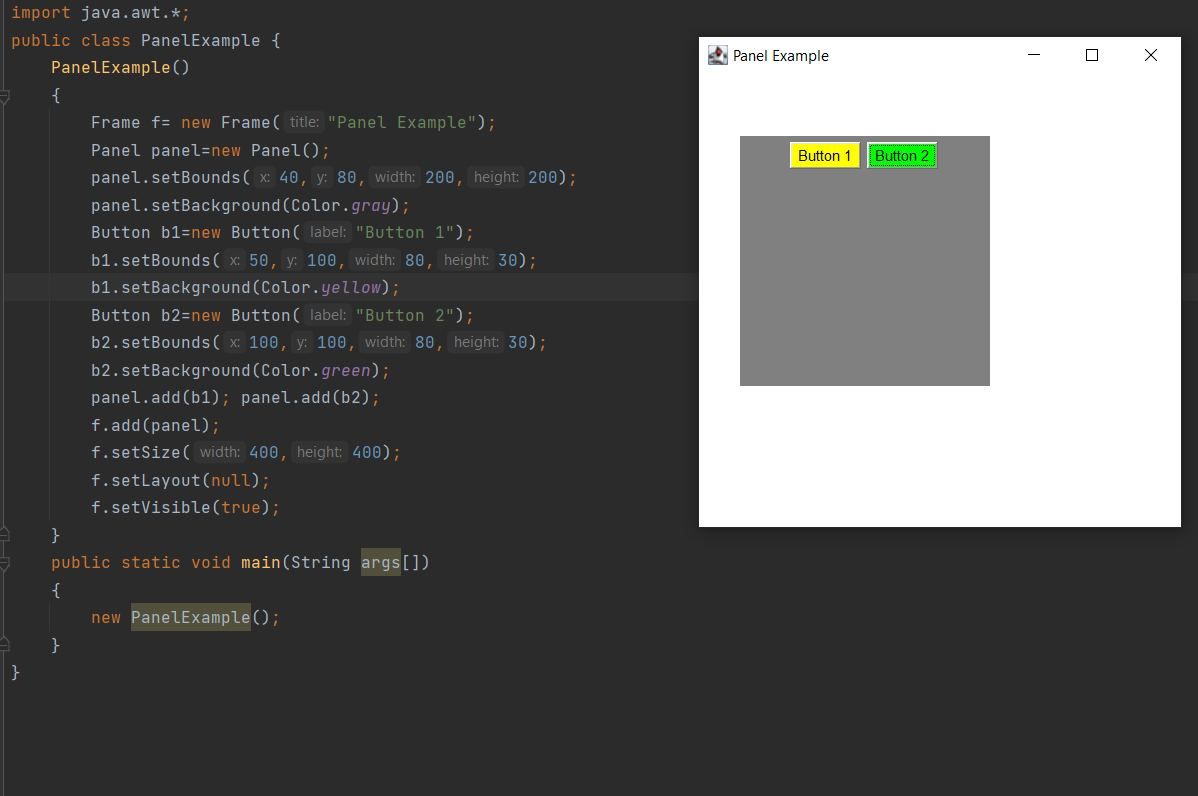
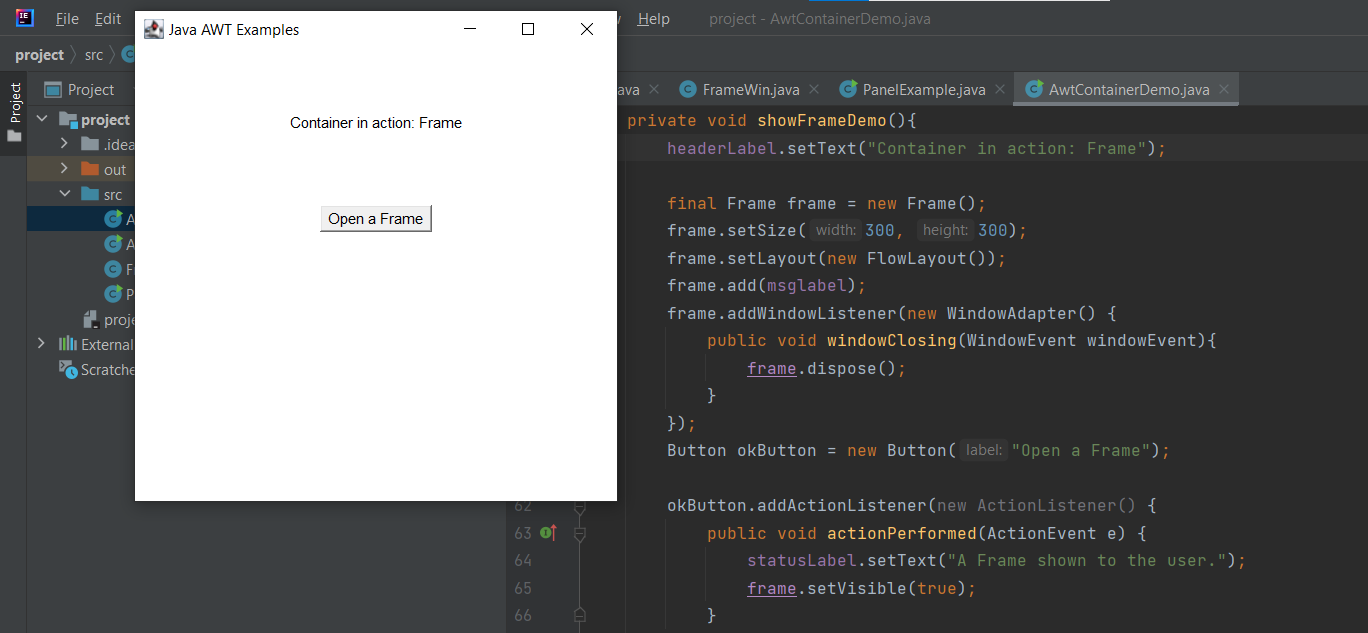
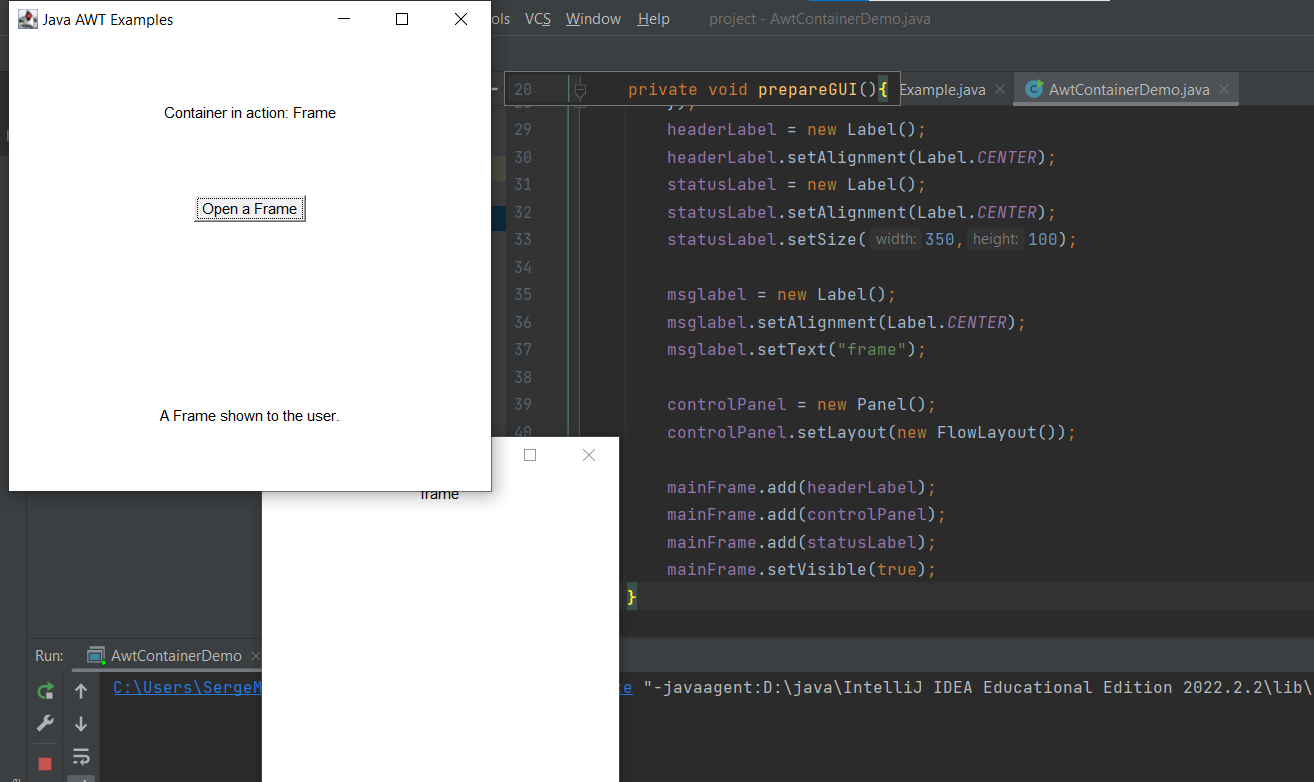
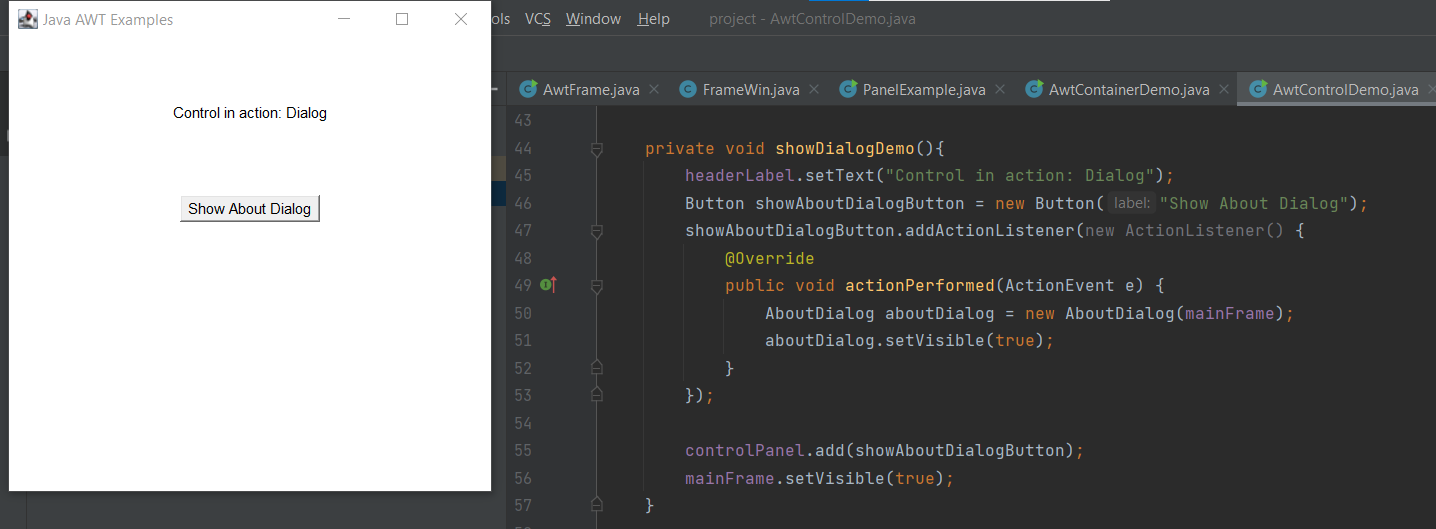
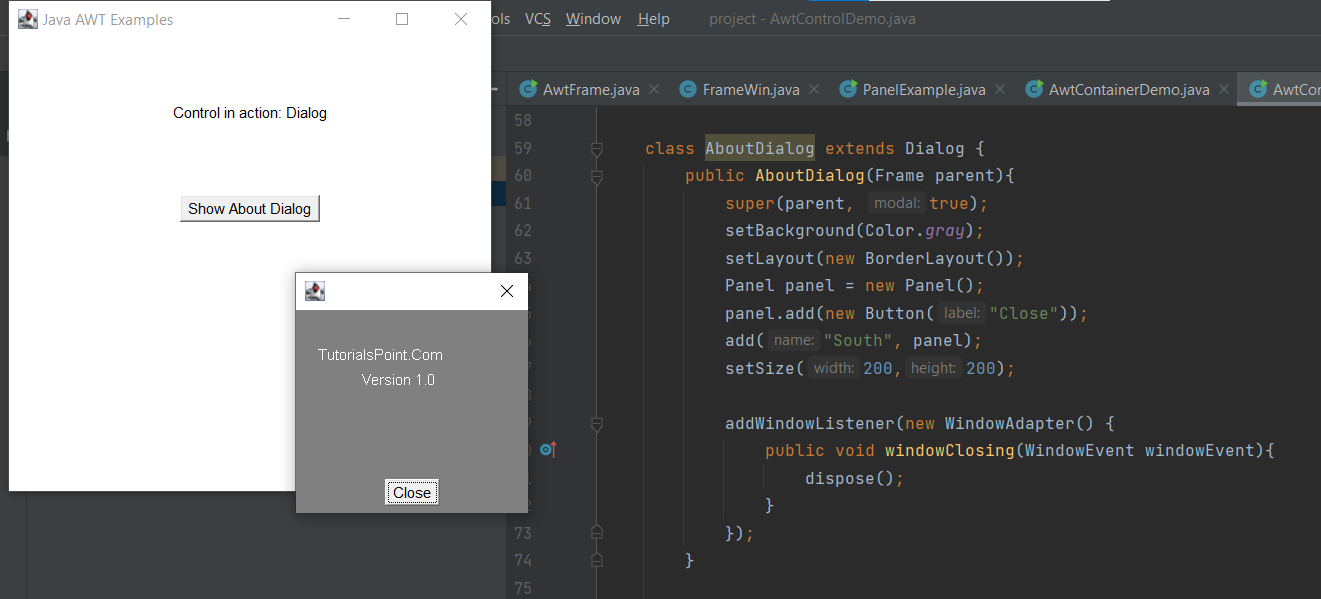
Indicates whether this dialog is resizable by the user.

* **boolean isUndecorated()**

Indicates whether this dialog is undecorated.

* **protected String paramString()**

Returns a string representing the state of this dialog.



import java.awt.FlowLayout;  
import java.awt.Frame;  
import java.awt.Label;  
import java.awt.event.WindowEvent;  
import java.awt.event.WindowListener;  
public class FrameWin extends Frame implements WindowListener {  
 public FrameWin(String FrameTitle){  
 super(FrameTitle);  
 setLayout(new FlowLayout());  
  
 //Create two label  
 Label L1 = new Label("There are two label");  
 Label L2 = new Label("both contain text");  
  
 //Add Label to the  
 add(L1);  
 add(L2);  
  
 //Set Size of the Frame  
 setSize(400, 300);  
 setLocation(100,100);  
  
 //Register with the Listener  
 addWindowListener(this);  
  
 }  
 //Implement the Listeners  
 public void windowOpened(WindowEvent e) {}  
 public void windowClosing(WindowEvent e) {  
 this.dispose();  
 }  
 public void windowClosed(WindowEvent e) {}  
 public void windowIconified(WindowEvent e) {}  
 public void windowDeiconified(WindowEvent e) {}  
 public void windowActivated(WindowEvent e) {}  
 public void windowDeactivated(WindowEvent e) {}  
}

Наведу приклад роботи класу, який мені доводилось використовувати, — Сlass Color AWT (клас кольорів AWT)

Інструментарій абстрактних вікон надає клас кольору, який використовується для створення та встановлення кольору для компонентів. Ми також можемо встановлювати кольори для компонентів, використовуючи фреймворк через властивості компонентівКолір, як і все в Java, — обєкт певного класу, а саме, класу Сolor. Основу класу складають сім конструкторів кольору. Найпростіший конструктор **Color(int red, int green, int blue) с**творює колір, одержаний змішуванням червоного (red), зеленого (green) і синього (blue). Ця кольорова модель називається RGB. Кожна складова змінюється від 0 (відсутність складової) до 255 (повна інтенсивність цієї складової). Наприклад, щоб визначити чистий яскраво-червоний pureRed і чистий яскраво-зелений pureGreen кольори, слід:

Color pureRed = new Color(255, 0, 0);

Color pureGreen = new Color(0, 255, 0);

У другому конструкторі **Color(float red, float green, float blue)** інтенсивність складової можна змінювати більш гладко дійсними числами від 0.0 (відсутність складової) до 1.0 (повна інтенсивність складової). Наприклад,

Color someColor = new Color(0.05f, 0.4f, 0.95f);

Третій конструктор — **Color(int rgb)**, який задає всі три складові в одному цілому числі. В бітах 16–23 записується червона складова, в бітах 8–15 — зелена, а в бітах 0–7 — синя складова кольору. Наприклад:

Color с = new Color(0хFF8F48FF);

Тут червона складова задана з інтенсивністю 0x8F, зелена — 0x48, синя — 0xFF.

Наступні три конструктори Color(int red, int green, int blue, int alpha), Color(float red, float green, float blue, float alpha), Color(int rgb, boolean hasAlpha), вводять четверту складову кольору, так звану "альфу", що визначає прозорість кольору. Ця складова проявляє себе при накладанні одного кольору на другий. Якщо альфа рівна 255 або 1,0, то колір повністю непрозорий, попередній колір не просвічується крізь нього. Якщо альфа рівна 0 або 0,0, то колір абсолютно прозорий, для кожного пікселя видно тільки попередній колір. Останній із цих конструкторів враховує складову альфа, що знаходиться в бітах 24—31, якщо параметр hasAІpha рівний true. Якщо ж hasAІpha рівне false, то складова альфа вважається рівною 255, незалежно від того, що записано в старших бітах параметра rgb. Перші три конструктори створюють непрозорий колір з альфою, рівною 255 або 1,0.

Сьомий конструктор **Color(ColorSpace cspace, float[] components, float alpha)** дозволяє створювати колір не тільки в кольоровій моделі (color model) RGB, але і в інших моделях: CMYK, HSB, CIEXYZ, визначених обєктом класу ColorSpace. Для створення кольору в моделі HSB можна скористатися статичним методом:

* getHSBColor(float hue, float saturation, float brightness).

Якщо немає необхідності ретельно підбирати кольори, то можна просто скористатися однією із тринадцати статичних констант типу color класу Сolor. Наперекір стандарту"Code Conventions" вони записуються рядковими літерами: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow.

Методи класу Color дозволяють одержати складові поточного кольору: getRed(), getGreen(), getBlue(), getAlpha(), getRGB(), getColorSpace (), getComponents (). Два методи створюють більш яскравий brighter() і більш темний darker() кольори в порівнянні з поточним кольором. Вони корисні, якщо треба виділити активний компонент або, навпаки, показати неактивний компонент блідніше решти компонентів. Два статичних методи повертають колір, перетворений із кольорової моделі RGB в HSB і навпаки:

* float[] RGBtoHSB(int red, int green, int blue, float[] hsb)
* int HSBtoRGB(int hue, int saturation, int brightness)

Створивши колір, можна рисувати ним в графічному контексті.

У таблиці перераховані різні методи, передбачені класом Color.

|  |  |
| --- | --- |
| **Конструктор / Методи** | **Опис** |
| getBlue () | Повертає компонент синього кольору в діапазоні 0-255. |
| brighter() | Створює яскравішу версію поточного кольору. |
| createContext (ColorModel cm, Rectangle r, Rectangle2D r2d, AffineTransform x, RenderingHints h) | Повертає новий PaintContext. |
| darker() /td> | Створює темнішу версію поточного кольору. |
| decode(String nm) | Повертає вказаний непрозорий колір, перетворюючи рядок у ціле число. |
| equals(Object obj) | Перевіряє, чи заданий кольоровий об’єкт дорівнює поточному об’єкту. |
| getAlpha () | Повертає значення альфа-кольору в діапазоні від 0 до 255. |
| getColor (рядок нм) | Повертає колір із властивостей системи. |
| getColor (рядок nm, колір v) |
| getColor (рядок nm, int v) |
| getColorComponents (ColorSpace cspace, float [] compArray) | Повертає масив типу float, що містить кольорові компоненти із зазначеного ColorSpace. |
| getColorComponents (float [] compArray) | Повертає масив типу float, що містить кольорові компоненти, із ColorSpace Color. |
| getColorSpace () | повертає ColorSpace поточного кольору. |
| getGreen () | Повертає компонент зеленого кольору в діапазоні 0-255 у просторі sRGB за замовчуванням. |
| getRed () | Повертає компонент червоного кольору в діапазоні 0-255 у просторі sRGB за замовчуванням. |
| getRGB () | Повертає значення RGB поточного кольору в sRGB ColorModel за замовчуванням. |
| getHSBColor (float h, float s, float b) | Створює об’єкт Колір, використовуючи кольорову модель HSB із заданими значеннями. |
| getTransparency () | повертає значення прозорості для цього кольору. |
| hashCode () | Повертає хеш-код для цього кольору. |
| HSBtoRGB (float h, float s, float b) | Перетворіть заданий HSB у значення RGB |
| RGBtoHSB (int r, int g, int b, float [] hsbvals) | перетворює задані значення RGB у значення HSB. |

## Програма для встановлення кольору фону панелі за допомогою кольору, зазначеного в константах класу

//Java program to set the background color of panel

// using the color specified in the constants

// of the class.

import java.awt.\*;

import javax.swing.\*;

class color extends JFrame {

    // constructor

    color()

    {

        super("color");

        // create a new Color

        Color c = Color.yellow;

        // create a panel

        JPanel p = new JPanel();

        // set the background of the frame

        // to the specified Color

        p.setBackground(c);

        setSize(200, 200);

        add(p);

        show();

    }

    // Main Method

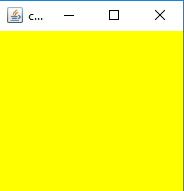
    public static void main(String args[])

    {

        color c = new color();

    }

}



## Програма для створення нового кольору, вказавши значення RGB і встановивши його як фон панелі

// Java program to create a new Color by using

// Color(int rgb) method, set it as background

// of panel .

import java.awt.\*;

import javax.swing.\*;

class color extends JFrame {

    // constructor

    color()

    {

        super("color");

        // create a new Color

        // RGB value of blue is 255

        // and set its alpha value as

        // 200 out of 255

        Color c = new Color(255);

        // create a panel

        JPanel p = new JPanel();

        // set the background of the

       // frame to the specified Color

        p.setBackground(c);

        setSize(200, 200);

        add(p);

        show();

    }

    // Main Method

    public static void main(String args[])

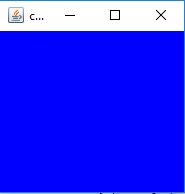
    {

        color c = new color();

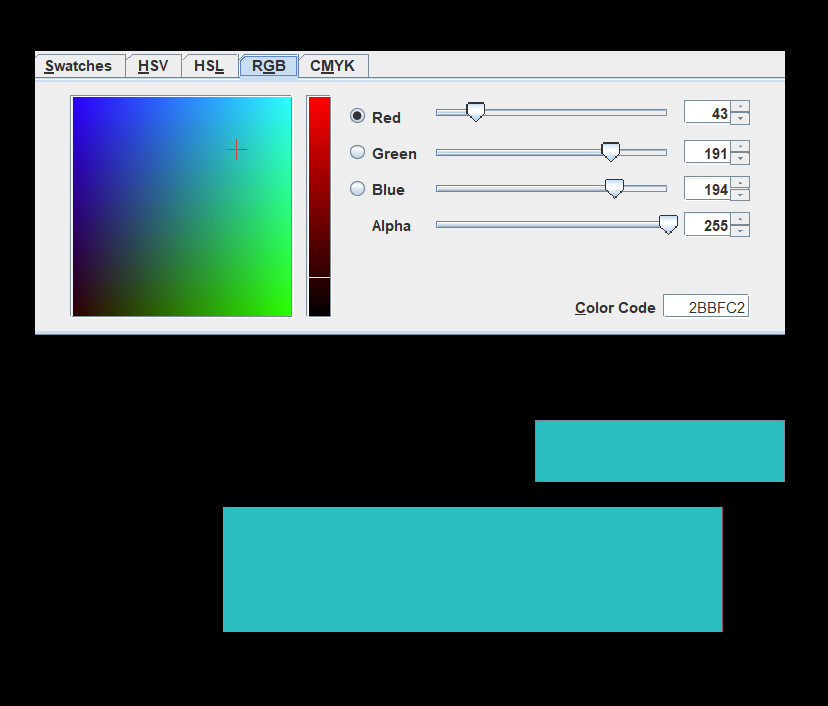
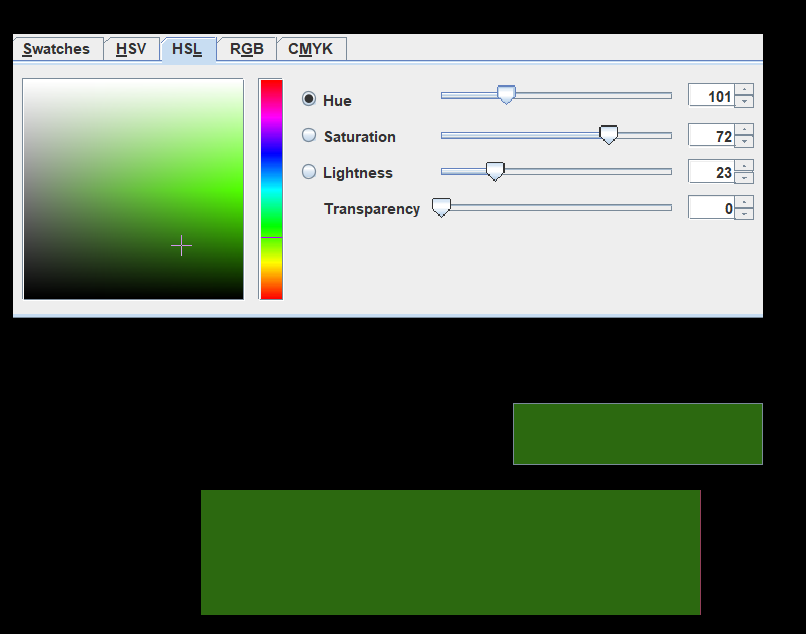
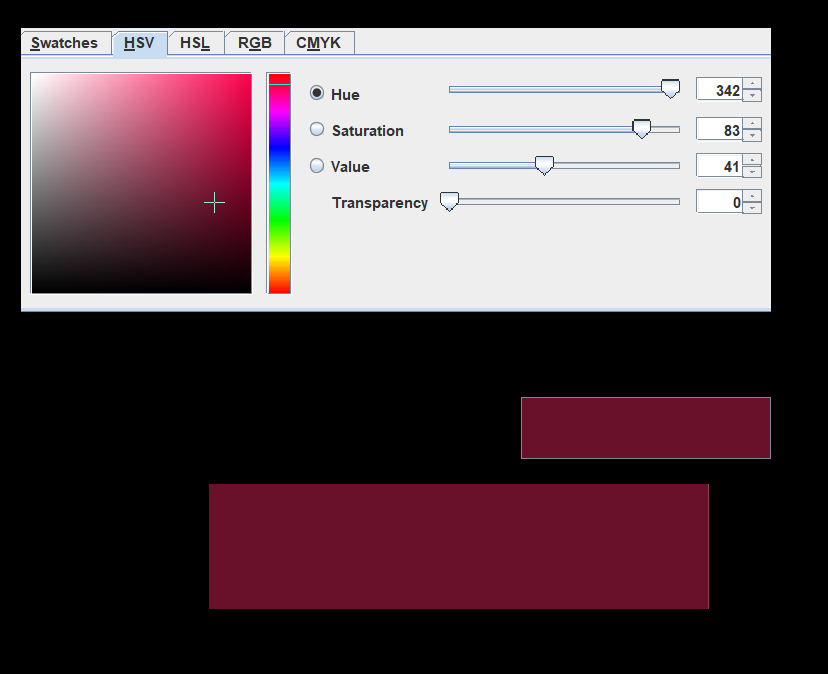
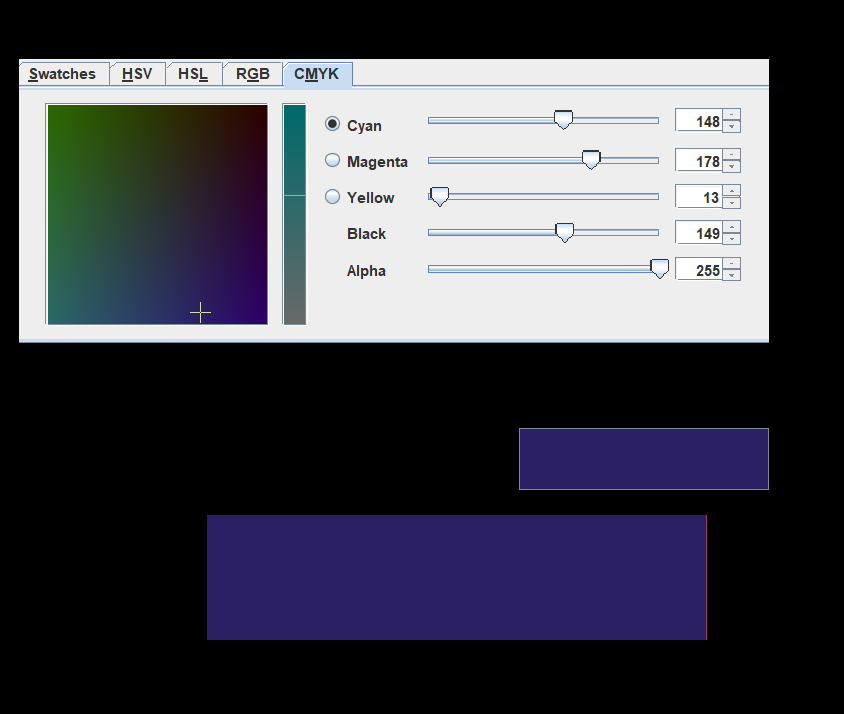
    }

}

**Output :**



Є можливість вибрати будь-який колір на кожній кольоровій гамі, щоб передати його на button і frame, що демонструється на рисунках:

 **** ****

## Основні класи бібліотеки Java.AWT

**AWT Controls**

Кожен інтерфейс користувача розглядає наступні три основні аспекти:

* **Елементи інтерфейсу користувача** . Це основні візуальні елементи, з якими користувач зрештою бачить та взаємодіє. GWT надає величезний список широко використовуваних і загальних елементів, що варіюються від базових до складних, які ми розглянемо у цьому посібнику.
* **Макети:** вони визначають, як елементи інтерфейсу користувача повинні бути організовані на екрані, і забезпечують остаточний вигляд графічного інтерфейсу користувача. Ця частина буде розглянута у розділі «Макет».
* **Поведінка:** це події, які відбуваються, коли користувач взаємодіє з елементами інтерфейсу користувача. Ця частина буде розглянута у розділі «Обробка подій».

**Елементи інтерфейсу користувача** . Це основні візуальні елементи, з якими користувач зрештою бачить та взаємодіє. GWT надає величезний список широко використовуваних і загальних елементів, що варіюються від базових до складних.

**Макети:** вони визначають, як елементи інтерфейсу користувача повинні бути організовані на екрані, і забезпечують остаточний вигляд графічного інтерфейсу користувача.

**Поведінка:** це події, які відбуваються, коли користувач взаємодіє з елементами інтерфейсу користувача.

Кожен елемент управління AWT успадковує властивості класу Component. Слід зазначити, що кожний клас має певний конструктор, поля,методи, методи наслідування,з якими можна ознайомитись в описі бібліотеки.

***Основні класи бібліотеки JAVA.AWT***

* **Клас AWT Component**

Клас **Component** є абстрактним базовим класом для елементів керування інтерфейсу користувача без меню AWT. Компонент представляє об'єкт із графічним поданням.

***Оголошення класу* java.awt.Component** :

Public abstract class Component extends Object implements ImageObserver , MenuContainer , Serializable

**Поля**

* **static float BOTTOM\_ALIGNMENT** – Константа простоти використання для getAlignmentY.
* **static float CENTER\_ALIGNMENT** – Константа простоти використання для getAlignmentY та getAlignmentX.
* **static float LEFT\_ALIGNMENT** – Константа простоти використання для getAlignmentX.
* **static float RIGHT\_ALIGNMENT** – Константа простоти використання для getAlignmentX.
* **static float TOP\_ALIGNMENT** – Константа простоти використання для getAlignmentY().
* **static float BOTTOM\_ALIGNMENT** – Константа простоти використання для getAlignmentY.
* **static float CENTER\_ALIGNMENT** – Константа простоти використання для getAlignmentY та getAlignmentX.
* **static float LEFT\_ALIGNMENT** – Константа простоти використання для getAlignmentX.
* **static float RIGHT\_ALIGNMENT** – Константа простоти використання для getAlignmentX.
* **static float TOP\_ALIGNMENT** – Константа простоти використання для getAlignmentY().
* **Клас AWT Label**

Мітка є пасивним елементом керування, оскільки вона не створює жодних подій під час доступу користувача. Елемент керування міткою є об'єктом мітки. Мітка відображає один рядок тексту лише для читання. Однак текст може бути змінений прикладним програмістом, але не може бути змінений кінцевим користувачем будь-яким чином.

***Оголошення класу* java.awt.Label** :

public class Label extends Component implements Accessible

**Поля:**

* **static int CENTER** – вказує, що мітка має бути центрованою.
* **static int LEFT** – вказує, що мітка має бути вирівняна по лівому краю.
* **static int RIGHT** – вказує, що мітка має бути правильно вирівняна.
* **static int CENTER** – вказує, що мітка має бути центрованою.
* **static int LEFT** – вказує, що мітка має бути вирівняна по лівому краю.
* **static int RIGHT** – вказує, що мітка має бути правильно вирівняна.
* **AWT Button Class**

Кнопка є компонентом керування, який має мітку та генерує подію при натисканні. Коли кнопка натиснута та відпущена, AWT відправляє екземпляр ActionEvent на кнопку, викликаючи processEvent для кнопки. Метод processEvent кнопки отримує всі події для кнопки; Він передає подію дії, викликаючи свій власний метод процесудіяльності. Останній метод передає подію дії всім слухачам дії, які зареєстрували інтерес до подій дії, що генерується цією кнопкою. Якщо програма хоче виконати будь-яку дію на основі натискання та відпускання кнопки, вона повинна реалізувати ActionListener та зареєструвати новий прослуховувач для отримання подій від цієї кнопки, викликаючи метод addActionListener кнопки. Програма може використовувати команду дії кнопки як протокол обміну повідомленнями.

***Оголошення класу* java.awt.Button** :

public class Button extends Component implements Accessible

* **AWT CheckBox Class**

Прапорець використовується для включення або вимкнення опції (true) або вимкнення (false). Для кожного прапорця є мітка, що означає його дію. Стан прапорця можна змінити, натиснувши на нього.

***Оголошення класу* java.awt.Checkbox** :

Public class Checkbox extends Component implements ItemSelectable , Accessible

* **AWT CheckBoxGroup Class**

Клас CheckboxGroup використовується для групування набору прапорців.

*Оголошення класу* java.awt.CheckboxGroup :

public class CheckboxGroup extends Object implements Serializable

* **AWT List Class**

Список є списком текстових елементів. Список може бути налаштований так, що користувач може вибрати один або кілька елементів.

***Оголошення класу* java.awt.List** :

Public class List extends Component implements ItemSelectable , Accessible

* **AWT TextField Class**

Компонент textField дозволяє користувачеві редагувати один рядок тексту. Коли користувач вводить ключ у текстовому полі, подія надсилається до TextField. Ключовою подією може бути натиснута клавіша, відпущена клавіша або набрана клавіша. Ключова подія передається зареєстрованому KeyListener. Також можливо для ActionEvent, якщо ActionEvent увімкнено в текстовому полі, тоді ActionEvent може бути запущено, натиснувши клавішу повернення.

*Оголошення класу* java.awt.TextField class:

public class TextField extends TextComponent

* **AWT TextArea Class**

Елемент керування TextArea в AWT надає нам багаторядкову область редактора. Користувач може друкувати тут скільки завгодно. Коли текст у текстовій області стає більшим за видиму область, автоматично з’являється смуга прокручування, яка допомагає нам прокручувати текст вгору та вниз, праворуч і ліворуч.

*Оголошення класу* java.awt.TextArea class:

public class TextArea extends TextComponent

**Поля**

* **static int SCROLLBARS\_BOTH** — cтворення та відображення як вертикальних, так і горизонтальних смуг прокручування.

**static int SCROLLBARS\_HORIZONTAL\_ONLY** — cтворення та відображення лише горизонтальної смуги прокрутки.

* **static int SCROLLBARS\_NONE** — не створюйте та не відображайте смуги прокрутки для текстової області.
* **static int SCROLLBARS\_VERTICAL\_ONLY** — створення та відображення лише вертикальної смуги прокручування..
* **static int SCROLLBARS\_BOTH** — створення та відображення як вертикальних, так і горизонтальних смуг прокручування.
* **static int SCROLLBARS\_HORIZONTAL\_ONLY** — створення та відображення лише горизонтальної смуги прокручування.
* **static int SCROLLBARS\_NONE** — не створювати та не відображати смуги прокрутки для текстової області.
* **static int SCROLLBARS\_VERTICAL\_ONLY** — Створення та відображення лише вертикальної смуги прокручування.
  + **AWT Choice Class**

Choice control використовується як show pop up menu of choices. Вибраний варіант відображається у верхній частині меню.

Оголошення класу

Following is the declaration for **java.awt.Choice** class:

Public class Choice extends Component implements ItemSelectable , Accessible

* **AWT Canvas Class**

Canvas control це порожня прямокутна область, де користувач може малювати або фіксувати введені користувачем дані. Клас Canvas успадковує клас Component.

*Оголошення класу* java.awt.Canvas class:

public class Canvas extends Component implements Accessible

* **AWT Image Class**

Image control є суперкласом для всіх класів зображень, що представляють графічні зображення.

*Оголошення класу* java.awt.Image class:

public abstract class Image

extends Object

**Поля**:

* **protected float accelerationPriority** — Priority for accelerating this image.
* **static int SCALE\_AREA\_AVERAGING** — Use the Area Averaging image scaling algorithm.
* **static int SCALE\_DEFAULT** — Use the default image-scaling algorithm.
* **static int SCALE\_FAST** — Choose an image-scaling algorithm that gives higher priority to scaling speed than smoothness of the scaled image.
* **static int SCALE\_REPLICATE** — Use the image scaling algorithm embodied in the ReplicateScaleFilter class.
* **static int SCALE\_SMOOTH** — Choose an image-scaling algorithm that gives higher priority to image smoothness than scaling speed.
* **static Object UndefinedProperty** — The UndefinedProperty object should be returned whenever a property which was not defined for a particular image is fetched.
* **protected float accelerationPriority** — Priority for accelerating this image.
* **static int SCALE\_AREA\_AVERAGING** — Use the Area Averaging image scaling algorithm.
* **static int SCALE\_DEFAULT** — Use the default image-scaling algorithm.
* **static int SCALE\_FAST** — Choose an image-scaling algorithm that gives higher priority to scaling speed than smoothness of the scaled image.
* **static int SCALE\_REPLICATE** — Use the image scaling algorithm embodied in the ReplicateScaleFilter class.
* **static int SCALE\_SMOOTH** — Choose an image-scaling algorithm that gives higher priority to image smoothness than scaling speed.
* **static Object UndefinedProperty** — The UndefinedProperty object should be returned whenever a property which was not defined for a particular image is fetched.
* **AWT Scrollbar Class**

Елемент керування смугою прокручування представляє компонент смуги прокручування, який дозволяє користувачеві вибирати з діапазону значень.

*Оголошення класу* java.awt.Scrollbar class:

public class Scrollbar

extends Component

implements Adjustable, Accessible

**Поля**

* **static int HORIZONTAL** –A constant that indicates a horizontal scroll bar.
* **static int VERTICAL** –A constant that indicates a vertical scroll bar.
* **static int HORIZONTAL** –A constant that indicates a horizontal scroll bar.
* **static int VERTICAL** –A constant that indicates a vertical scroll bar.
* **AWT Dialog Class**

Елемент керування діалоговим вікном являє собою вікно верхнього рівня із заголовком і рамкою, що використовується для прийняття певної форми введення від користувача.

*Оголошення класу* java.awt.Dialog class:

public class Dialog

extends Window

**Поля**

* **static Dialog.ModalityType DEFAULT\_MODALITY\_TYPE** — Default modality type for modal dialogs.
* **static Dialog.ModalityType DEFAULT\_MODALITY\_TYPE** — Default modality type for modal dialogs.
* **AWT FileDialog Class**

Елемент керування FileDialog представляє діалогове вікно, з якого користувач може вибрати файл.

*Оголошення класу* java.awt.FileDialog

public class FileDialog

extends Dialog

**Поля**

* **static int LOAD** — This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
* **static int SAVE** — This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.
* **static int LOAD** — This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
* **static int SAVE** — This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.
* **Класи подій AWT**

Класи Event представляють події. Java надає багато різних класів Event, я наведу деякі з них.

* **Клас EventObject**

Це кореневий клас, з якого повинні бути отримані всі об'єкти стану події. Усі події побудовані зі ссылкою на об’єкт, джерело , який логічно вважається об’єктом, з яким спочатку виникла розглядувана подія. Цей клас визначено в пакеті java.util.

*Оголошення класу* java.util.EventObject :

public class EventObject

extends Object

implements Serializable

**Поля**

* **Захищене джерело обєкту** – обєкт, на якому відбулась подія.

**Класи подій AWT:**

Список класів подій, які найчастіше використовуються.

* **AWT AWTEvent Class**

Це кореневий клас подій для всіх подій AWT. Цей клас і його підкласи замінюють вихідний клас java.awt.Event. Цей клас визначено в пакеті java.awt. Цей клас має метод під назвою getID(), який можна використовувати для визначення типу події.

*Оголошення класу* java.awt.AWTEvent class:

public class AWTEvent

extends EventObject

**Поля**

* **static int ACTION\_FIRST** — The first number in the range of ids used for action events.
* **static long ACTION\_EVENT\_MASK** — The event mask for selecting action events.
* **static long ADJUSTMENT\_EVENT\_MASK** — The event mask for selecting adjustment events.
* **static long COMPONENT\_EVENT\_MASK** — The event mask for selecting component events.
* **protected boolean consumed** — Controls whether or not the event is sent back down to the peer once the source has processed it – false means it’s sent to the peer; true means it’s not.
* **static long CONTAINER\_EVENT\_MASK** — The event mask for selecting container events.
* **static long FOCUS\_EVENT\_MASK** — The event mask for selecting focus events.
* **static long HIERARCHY\_BOUNDS\_EVENT\_MASK** — The event mask for selecting hierarchy bounds events.
* **static long HIERARCHY\_EVENT\_MASK** — The event mask for selecting hierarchy events.
* **protected int id** — The event’s id.
* **static long INPUT\_METHOD\_EVENT\_MASK** — The event mask for selecting input method events.
* **static long INVOCATION\_EVENT\_MASK** — The event mask for selecting invocation events.
* **static long ITEM\_EVENT\_MASK** — The event mask for selecting item events.
* **static long KEY\_EVENT\_MASK** — The event mask for selecting key events.
* **static long MOUSE\_EVENT\_MASK** — The event mask for selecting mouse events.
* **static long MOUSE\_MOTION\_EVENT\_MASK** — The event mask for selecting mouse motion events.
* **static long MOUSE\_WHEEL\_EVENT\_MASK** — The event mask for selecting mouse wheel events.
* **static long PAINT\_EVENT\_MASK** — The event mask for selecting paint events.
* **static int RESERVED\_ID\_MAX** — The maximum value for reserved AWT event IDs.
* **static long TEXT\_EVENT\_MASK** — The event mask for selecting text events.
* **static long WINDOW\_EVENT\_MASK** — The event mask for selecting window events.
* **static long WINDOW\_FOCUS\_EVENT\_MASK** — The event mask for selecting window focus events.
* **static long WINDOW\_STATE\_EVENT\_MASK** — The event mask for selecting window state events.
* **static int ACTION\_FIRST** — The first number in the range of ids used for action events.
* **static long ACTION\_EVENT\_MASK** — The event mask for selecting action events.
* **static long ADJUSTMENT\_EVENT\_MASK** — The event mask for selecting adjustment events.
* **static long COMPONENT\_EVENT\_MASK** — The event mask for selecting component events.
* **protected boolean consumed** — Controls whether or not the event is sent back down to the peer once the source has processed it – false means it’s sent to the peer; true means it’s not.
* **static long CONTAINER\_EVENT\_MASK** — The event mask for selecting container events.
* **static long FOCUS\_EVENT\_MASK** — The event mask for selecting focus events.
* **static long HIERARCHY\_BOUNDS\_EVENT\_MASK** — The event mask for selecting hierarchy bounds events.
* **static long HIERARCHY\_EVENT\_MASK** — The event mask for selecting hierarchy events.
* **protected int id** — The event’s id.
* **static long INPUT\_METHOD\_EVENT\_MASK** — The event mask for selecting input method events.
* **static long INVOCATION\_EVENT\_MASK** — The event mask for selecting invocation events.
* **static long ITEM\_EVENT\_MASK** — The event mask for selecting item events.
* **static long KEY\_EVENT\_MASK** — The event mask for selecting key events.
* **static long MOUSE\_EVENT\_MASK** — The event mask for selecting mouse events.
* **static long MOUSE\_MOTION\_EVENT\_MASK** — The event mask for selecting mouse motion events.
* **static long MOUSE\_WHEEL\_EVENT\_MASK** — The event mask for selecting mouse wheel events.
* **static long PAINT\_EVENT\_MASK** — The event mask for selecting paint events.
* **static int RESERVED\_ID\_MAX** — The maximum value for reserved AWT event IDs.
* **static long TEXT\_EVENT\_MASK** — The event mask for selecting text events.
* **static long WINDOW\_EVENT\_MASK** — The event mask for selecting window events.
* **static long WINDOW\_FOCUS\_EVENT\_MASK** — The event mask for selecting window focus events.
* **static long WINDOW\_STATE\_EVENT\_MASK** — The event mask for selecting window state events.
* **AWT ActionEvent Class**

Цей клас визначено в пакеті java.awt.event. Подія ActionEvent генерується при натисненій кнопці або подвійному кліку списку елементів.

*Оголошення класу* **java.awt.event.ActionEvent** class:

public class ActionEvent

extends AWTEvent

**Поля**

* static int ACTION\_FIRST — The first number in the range of ids used for action events.
* static int ACTION\_LAST — The last number in the range of ids used for action events.
* static int ACTION\_PERFORMED — This event id indicates that a meaningful action occured.
* static int ALT\_MASK — The alt modifier.
* static int CTRL\_MASK — The control modifier.
* static int META\_MASK — The meta modifier.
* static int SHIFT\_MASK — The shift modifier.
* **static int ACTION\_FIRST** — The first number in the range of ids used for action events.
* **static int ACTION\_LAST** — The last number in the range of ids used for action events.
* **static int ACTION\_PERFORMED** — This event id indicates that a meaningful action occured.
* **static int ALT\_MASK** — The alt modifier.
* **static int CTRL\_MASK** — The control modifier.
* **static int META\_MASK** — The meta modifier.
* **static int SHIFT\_MASK** — The shift modifier.
* **AWT InputEvent Class**

Клас InputEvent є кореневим класом подій для всіх вхідних подій рівня компонента. Вхідні події доставляються слухачам до того, як вони будуть нормально оброблені джерелом, з якого вони виникли. Це дозволяє слухачам і підкласам компонентів «споживати» подію, щоб джерело не обробляло їх за замовчуванням. Наприклад, використання подій mousePressed для компонента Button не дозволить активувати кнопку.

*Оголошення класу* java.awt.event.InputEvent class:

public abstract class InputEvent

extends ComponentEvent

**Поле**

* **static int ALT\_DOWN\_MASK** — The Alt key extended modifier constant.
* **static int ALT\_GRAPH\_DOWN\_MASK** — The AltGraph key extended modifier constant.
* **static int ALT\_GRAPH\_MASK** — The AltGraph key modifier constant.
* **static int ALT\_MASK** — The Alt key modifier constant.
* **static int BUTTON1\_DOWN\_MASK** — The Mouse Button1 extended modifier constant.
* **static int BUTTON1\_MASK** — The Mouse Button1 modifier constant.
* **static int BUTTON2\_DOWN\_MASK** — The Mouse Button2 extended modifier constant.
* **static int BUTTON2\_MASK** — The Mouse Button2 modifier constant.
* **static int BUTTON3\_DOWN\_MASK** — The Mouse Button3 extended modifier constant.
* **static int BUTTON3\_MASK** –The Mouse Button3 modifier constant.
* **static int CTRL\_DOWN\_MASK** — The Control key extended modifier constant.
* **static int CTRL\_MASK** — The Control key modifier constant.
* **static int META\_DOWN\_MASK** — The Meta key extended modifier constant.
* **static int META\_MASK** — The Meta key modifier constant.
* **static int SHIFT\_DOWN\_MASK** — The Shift key extended modifier constant.
* **static int SHIFT\_MASK** — The Shift key modifier constant.
* **static int ALT\_DOWN\_MASK** — The Alt key extended modifier constant.
* **static int ALT\_GRAPH\_DOWN\_MASK** — The AltGraph key extended modifier constant.
* **static int ALT\_GRAPH\_MASK** — The AltGraph key modifier constant.
* **static int ALT\_MASK** — The Alt key modifier constant.
* **static int BUTTON1\_DOWN\_MASK** — The Mouse Button1 extended modifier constant.
* **static int BUTTON1\_MASK** — The Mouse Button1 modifier constant.
* **static int BUTTON2\_DOWN\_MASK** — The Mouse Button2 extended modifier constant.
* **static int BUTTON2\_MASK** — The Mouse Button2 modifier constant.
* **static int BUTTON3\_DOWN\_MASK** — The Mouse Button3 extended modifier constant.
* **static int BUTTON3\_MASK** –The Mouse Button3 modifier constant.
* **static int CTRL\_DOWN\_MASK** — The Control key extended modifier constant.
* **static int CTRL\_MASK** — The Control key modifier constant.
* **static int META\_DOWN\_MASK** — The Meta key extended modifier constant.
* **static int META\_MASK** — The Meta key modifier constant.
* **static int SHIFT\_DOWN\_MASK** — The Shift key extended modifier constant.
* **static int SHIFT\_MASK** — The Shift key modifier constant.
* **AWT KeyEvent Class**

Після введення символу генерується подія Key. Існує три типи ключових подій, які представлені цілочисельними константами. Ці ключові події наступні

* KEY\_PRESSED
* KEY\_RELASED
* KEY\_TYPED

*Оголошення класу* **java.awt.event.KeyEvent** class:

public class KeyEvent

extends InputEvent

**Поля**

* **static char CHAR\_UNDEFINED** –KEY\_PRESSED and KEY\_RELEASED events which do not map to a valid Unicode character use this for the keyChar value.
* **static int KEY\_FIRST** –The first number in the range of ids used for key events.
* **static int KEY\_LAST** –The last number in the range of ids used for key events.
* **static int KEY\_LOCATION\_LEFT** –A constant indicating that the key pressed or released is in the left key location (there is more than one possible location for this key).
* **static int KEY\_LOCATION\_NUMPAD** –A constant indicating that the key event originated on the numeric keypad or with a virtual key corresponding to the numeric keypad.
* **static int KEY\_LOCATION\_RIGHT** — A constant indicating that the key pressed or released is in the right key location (there is more than one possible location for this key).
* **static int KEY\_LOCATION\_STANDARD** –A constant indicating that the key pressed or released is not distinguished as the left or right version of a key, and did not originate on the numeric keypad (or did not originate with a virtual key corresponding to the numeric keypad).
* **static int KEY\_LOCATION\_UNKNOWN** — A constant indicating that the keyLocation is indeterminate or not relevant.
* **static int KEY\_PRESSED** –The “key pressed” event.
* **static int KEY\_RELEASED** –The “key released” event.
* **static int KEY\_TYPED** –The “key typed” event.
* static int VK\_0 –VK\_0 thru VK\_9 are the same as ASCII ‘0’ thru ‘9’ (0x30 – 0x39)
* **static int VK\_1**
* **static int VK\_2**
* **static int VK\_3**
* **static int VK\_4**
* **static int VK\_5**
* **static int VK\_6**
* **static int VK\_7**
* **static int VK\_8**
* **static int VK\_9**
* **static int VK\_A** –VK\_A thru VK\_Z are the same as ASCII ‘A’ thru ‘Z’ (0x41 – 0x5A)
* **static int VK\_ACCEPT** –Constant for the Accept or Commit function key.
* **static int VK\_ADD**
* **static int VK\_AGAIN**
* **static int VK\_ALL\_CANDIDATES** –Constant for the All Candidates function key.
* **static int VK\_ALPHANUMERIC** –Constant for the Alphanumeric function key.
* **static int VK\_ALT**
* **static int VK\_ALT\_GRAPH** –Constant for the AltGraph function key.
* **static int VK\_AMPERSAND**
* **static int VK\_ASTERISK**
* **static int VK\_AT** –constant for the “@” key.
* **static int VK\_B**
* **static int VK\_BACK\_QUOTE**
* **static int VK\_BACK\_SLASH** –Constant for the back slash key, “\”
* **static int VK\_BACK\_SPACE**
* **static int VK\_BEGIN** –Constant for the Begin key.
* **static int VK\_BRACELEFT**
* **static int VK\_BRACERIGHT**
* **static int VK\_C**
* **static int VK\_CANCEL**
* **static int VK\_CAPS\_LOCK**
* **static int VK\_CIRCUMFLEX** –Constant for the “^” key.
* **static int VK\_CLEAR**
* **static int VK\_CLOSE\_BRACKET** –Constant for the close bracket key, “]”
* **static int VK\_CODE\_INPUT** –Constant for the Code Input function key.
* **static int VK\_COLON** –Constant for the “:” key.
* **static int VK\_COMMA** –Constant for the comma key, “,”
* **static int VK\_COMPOSE** –Constant for the Compose function key.
* **static int VK\_CONTEXT\_MENU** –Constant for the Microsoft Windows Context Menu key.
* **static int VK\_CONTROL**
* **static int VK\_CONVERT** — Constant for the Convert function key.
* **static int VK\_COPY**
* **static int VK\_CUT**
* **static int VK\_D**
* **static int VK\_DEAD\_ABOVEDOT**
* **static int VK\_DEAD\_ABOVERING**
* **static int VK\_DEAD\_ACUTE**
* **static int VK\_DEAD\_BREVE**
* **static int VK\_DEAD\_CARON**
* **static int VK\_DEAD\_CEDILLA**
* **static int VK\_DEAD\_CIRCUMFLEX**
* **static int VK\_DEAD\_DIAERESIS**
* **static int VK\_DEAD\_DOUBLEACUTE**
* **static int VK\_DEAD\_GRAVE**
* **static int VK\_DEAD\_IOTA**
* **static int VK\_DEAD\_MACRON**
* **static int VK\_DEAD\_OGONEK**
* **static int VK\_DEAD\_SEMIVOICED\_SOUND**
* **static int VK\_DEAD\_TILDE**
* **static int VK\_DEAD\_VOICED\_SOUND**
* **static int VK\_DECIMAL**
* **static int VK\_DELETE**
* **static int VK\_DIVIDE**
* **static int VK\_DOLLAR** –Constant for the “$” key.
* **static int VK\_DOWN** — Constant for the non-numpad down arrow key.
* **static int VK\_E**
* **static int VK\_END**
* **static int VK\_ENTER**
* **static int VK\_EQUALS** –Constant for the equals key, “=”
* **static int VK\_ESCAPE**
* **static int VK\_EURO\_SIGN** –Constant for the Euro currency sign key.
* **static int VK\_EXCLAMATION\_MARK** –Constant for the “!” ключ.
* **static int VK\_F**
* **static int VK\_F1** –Constant for the F1 function key.
* **static int VK\_F10** –Constant for the F10 function key.
* **static int VK\_F11** –Constant for the F11 function key.
* **static int VK\_F12** –Constant for the F12 function key.
* **static int VK\_F13** –Constant for the F13 function key.
* **static int VK\_F14** –Constant for the F14 function key.
* **static int VK\_F15** –Constant for the F15 function key.
* **static int VK\_F16** –Constant for the F16 function key.
* **static int VK\_F17** –Constant for the F17 function key.
* **static int VK\_F18** –Constant for the F18 function key.
* **static int VK\_F19** –Constant for the F19 function key.
* **static int VK\_F2** –Constant for the F2 function key.
* **static int VK\_F20** –Constant for the F20 function key.
* **static int VK\_F21** — Constant for the F21 function key.
* **static int VK\_F22** –Constant for the F22 function key.
* **static int VK\_F23** –Constant for the F23 function key.
* **static int VK\_F24** –Constant for the F24 function key.
* **static int VK\_F3** –Constant for the F3 function key.
* **static int VK\_F4** –Constant for the F4 function key.
* **static int VK\_F5** — Constant for the F5 function key.
* **static int VK\_F6** –Constant for the F6 function key.
* **static int VK\_F7** –Constant for the F7 function key.
* **static int VK\_F8** –Constant for the F8 function key.
* **static int VK\_F9** –Constant for the F9 function key.
* **static int VK\_FINAL**
* **static int VK\_FIND**
* **static int VK\_FULL\_WIDTH** –Constant for the Full-Width Characters function key.
* **static int VK\_G**
* **static int VK\_GREATER**
* **static int VK\_H**
* **static int VK\_HALF\_WIDTH** –Constant for the Half-Width Characters function key.
* **static int VK\_HELP**
* **static int VK\_HIRAGANA** –Constant for the Hiragana function key.
* **static int VK\_HOME**
* **static int VK\_I**
* **static int VK\_INPUT\_METHOD\_ON\_OFF** — Constant for the input method on/off key.
* **static int VK\_INSERT**
* **static int VK\_INVERTED\_EXCLAMATION\_MARK** –Constant for the inverted exclamation mark key.
* **static int VK\_J**
* **static int VK\_JAPANESE\_HIRAGANA** –Constant for the Japanese-Hiragana function key.
* **static int VK\_JAPANESE\_KATAKANA** –Constant for the Japanese-Katakana function key.
* **static int VK\_JAPANESE\_ROMAN** –Constant for the Japanese-Roman function key.
* **static int VK\_K**
* **static int VK\_KANA**
* **static int VK\_KANA\_LOCK** — Constant for the locking Kana function key.
* **static int VK\_KANJI**
* **static int VK\_KATAKANA** –Constant for the Katakana function key.
* **static int VK\_KP\_DOWN** — Constant for the numeric keypad down arrow key.
* **static int VK\_KP\_LEFT** –Constant for the numeric keypad left arrow key.
* **static int VK\_KP\_RIGHT** –Constant for the numeric keypad right arrow key.
* **static int VK\_KP\_UP** –Constant for the numeric keypad up arrow key.
* **static int VK\_L**
* **static int VK\_LEFT** –Constant for the non-numpad left arrow key.
* **static int VK\_LEFT\_PARENTHESIS** –Constant for the “(” key.
* **static int VK\_LESS**
* **static int VK\_M**
* **static int VK\_META**
* **static int VK\_MINUS** — Constant for the minus key, “-”
* **static int VK\_MODECHANGE**
* **static int VK\_MULTIPLY**
* **static int VK\_N**
* **static int VK\_NONCONVERT** –Constant for the Don’t Convert function key.
* **static int VK\_NUM\_LOCK**
* **static int VK\_NUMBER\_SIGN** –Constant for the “#” key.
* **static int VK\_NUMPAD0**
* **static int VK\_NUMPAD1**
* **static int VK\_NUMPAD2**
* **static int VK\_NUMPAD3**
* **static int VK\_NUMPAD4**
* **static int VK\_NUMPAD5**
* **static int VK\_NUMPAD6**
* **static int VK\_NUMPAD7**
* **static int VK\_NUMPAD8**
* **static int VK\_NUMPAD9**
* **static int VK\_O**
* **static int VK\_OPEN\_BRACKET** –Constant for the open bracket key, “[”
* **static int VK\_P**
* **static int VK\_PAGE\_DOWN**
* **static int VK\_PAGE\_UP**
* **static int VK\_PASTE**
* **static int VK\_PAUSE**
* **static int VK\_PERIOD** –Constant for the period key, “.”
* **static int VK\_PLUS** — Constant for the “+” key.
* **static int VK\_PREVIOUS\_CANDIDATE** — Constant for the Previous Candidate function key.
* **static int VK\_PRINTSCREEN**
* **static int VK\_PROPS**
* **static int VK\_Q**
* **static int VK\_QUOTE**
* **static int VK\_QUOTEDBL**
* **static int VK\_R**
* **static int VK\_RIGHT** — Constant for the non-numpad right arrow key.
* **static int VK\_RIGHT\_PARENTHESIS** –Constant for the “)” key.
* **static int VK\_ROMAN\_CHARACTERS** –Constant for the Roman Characters function key.
* **static int VK\_S**
* **static int VK\_SCROLL\_LOCK**
* **static int VK\_SEMICOLON** — Constant for the semicolon key, “;”
* **static int VK\_SEPARATER** –This constant is obsolete, and is included only for backwards compatibility.
* **static int VK\_SEPARATOR** –Constant for the Numpad Separator key.
* **static int VK\_SHIFT**
* **static int VK\_SLASH** — Constant for the forward slash key, “/”
* **static int VK\_SPACE**
* **static int VK\_STOP**
* **static int VK\_SUBTRACT**
* **static int VK\_T**
* **static int VK\_TAB**
* **static int VK\_U**
* **static int VK\_UNDEFINED** — This value is used to indicate that the keyCode is unknown.
* **static int VK\_UNDERSCORE** –Constant for the “\_” key.
* **static int VK\_UNDO**
* **static int VK\_UP** –Constant for the non-numpad up arrow key.
* **static int VK\_V**
* **static int VK\_W**
* **static int VK\_WINDOWS** –Constant for the Microsoft Windows “Windows” key.
* **static int VK\_X**
* **static int VK\_Y**
* **static int VK\_Z**
* **static char CHAR\_UNDEFINED** –KEY\_PRESSED and KEY\_RELEASED events which do not map to a valid Unicode character use this for the keyChar value.
* **static int KEY\_FIRST** –The first number in the range of ids used for key events.
* **static int KEY\_LAST** –The last number in the range of ids used for key events.
* **static int KEY\_LOCATION\_LEFT** –A constant indicating that the key pressed or released is in the left key location (there is more than one possible location for this key).
* **static int KEY\_LOCATION\_NUMPAD** –A constant indicating that the key event originated on the numeric keypad or with a virtual key corresponding to the numeric keypad.
* **static int KEY\_LOCATION\_RIGHT** — A constant indicating that the key pressed or released is in the right key location (there is more than one possible location for this key).
* **static int KEY\_LOCATION\_STANDARD** –A constant indicating that the key pressed or released is not distinguished as the left or right version of a key, and did not originate on the numeric keypad (or did not originate with a virtual key corresponding to the numeric keypad).
* **static int KEY\_LOCATION\_UNKNOWN** — A constant indicating that the keyLocation is indeterminate or not relevant.
* **static int KEY\_PRESSED** –The “key pressed” event.
* **static int KEY\_RELEASED** –The “key released” event.
* **static int KEY\_TYPED** –The “key typed” event.
* **static int VK\_1**
* **static int VK\_2**
* **static int VK\_3**
* **static int VK\_4**
* **static int VK\_5**
* **static int VK\_6**
* **static int VK\_7**
* **static int VK\_8**
* **static int VK\_9**
* **static int VK\_A** –VK\_A thru VK\_Z are the same as ASCII ‘A’ thru ‘Z’ (0x41 – 0x5A)
* **static int VK\_ACCEPT** –Constant for the Accept or Commit function key.
* **static int VK\_ADD**
* **static int VK\_AGAIN**
* **static int VK\_ALL\_CANDIDATES** –Constant for the All Candidates function key.
* **static int VK\_ALPHANUMERIC** –Constant for the Alphanumeric function key.
* **static int VK\_ALT**
* **static int VK\_ALT\_GRAPH** –Constant for the AltGraph function key.
* **static int VK\_AMPERSAND**
* **static int VK\_ASTERISK**
* **static int VK\_AT** –constant for the “@” key.
* **static int VK\_B**
* **static int VK\_BACK\_QUOTE**
* **static int VK\_BACK\_SLASH** –Constant for the back slash key, “\”
* **static int VK\_BACK\_SPACE**
* **static int VK\_BEGIN** –Constant for the Begin key.
* **static int VK\_BRACELEFT**
* **static int VK\_BRACERIGHT**
* **static int VK\_C**
* **static int VK\_CANCEL**
* **static int VK\_CAPS\_LOCK**
* **static int VK\_CIRCUMFLEX** –Constant for the “^” key.
* **static int VK\_CLEAR**
* **static int VK\_CLOSE\_BRACKET** –Constant for the close bracket key, “]”
* **static int VK\_CODE\_INPUT** –Constant for the Code Input function key.
* **static int VK\_COLON** –Constant for the “:” key.
* **static int VK\_COMMA** –Constant for the comma key, “,”
* **static int VK\_COMPOSE** –Constant for the Compose function key.
* **static int VK\_CONTEXT\_MENU** –Constant for the Microsoft Windows Context Menu key.
* **static int VK\_CONTROL**
* **static int VK\_CONVERT** — Constant for the Convert function key.
* **static int VK\_COPY**
* **static int VK\_CUT**
* **static int VK\_D**
* **static int VK\_DEAD\_ABOVEDOT**
* **static int VK\_DEAD\_ABOVERING**
* **static int VK\_DEAD\_ACUTE**
* **static int VK\_DEAD\_BREVE**
* **static int VK\_DEAD\_CARON**
* **static int VK\_DEAD\_CEDILLA**
* **static int VK\_DEAD\_CIRCUMFLEX**
* **static int VK\_DEAD\_DIAERESIS**
* **static int VK\_DEAD\_DOUBLEACUTE**
* **static int VK\_DEAD\_GRAVE**
* **static int VK\_DEAD\_IOTA**
* **static int VK\_DEAD\_MACRON**
* **static int VK\_DEAD\_OGONEK**
* **static int VK\_DEAD\_SEMIVOICED\_SOUND**
* **static int VK\_DEAD\_TILDE**
* **static int VK\_DEAD\_VOICED\_SOUND**
* **static int VK\_DECIMAL**
* **static int VK\_DELETE**
* **static int VK\_DIVIDE**
* **static int VK\_DOLLAR** –Constant for the “$” key.
* **static int VK\_DOWN** — Constant for the non-numpad down arrow key.
* **static int VK\_E**
* **static int VK\_END**
* **static int VK\_ENTER**
* **static int VK\_EQUALS** –Constant for the equals key, “=”
* **static int VK\_ESCAPE**
* **static int VK\_EURO\_SIGN** –Constant for the Euro currency sign key.
* **static int VK\_EXCLAMATION\_MARK** –Constant for the “!” ключ.
* **static int VK\_F**
* **static int VK\_F1** –Constant for the F1 function key.
* **static int VK\_F10** –Constant for the F10 function key.
* **static int VK\_F11** –Constant for the F11 function key.
* **static int VK\_F12** –Constant for the F12 function key.
* **static int VK\_F13** –Constant for the F13 function key.
* **static int VK\_F14** –Constant for the F14 function key.
* **static int VK\_F15** –Constant for the F15 function key.
* **static int VK\_F16** –Constant for the F16 function key.
* **static int VK\_F17** –Constant for the F17 function key.
* **static int VK\_F18** –Constant for the F18 function key.
* **static int VK\_F19** –Constant for the F19 function key.
* **static int VK\_F2** –Constant for the F2 function key.
* **static int VK\_F20** –Constant for the F20 function key.
* **static int VK\_F21** — Constant for the F21 function key.
* **static int VK\_F22** –Constant for the F22 function key.
* **static int VK\_F23** –Constant for the F23 function key.
* **static int VK\_F24** –Constant for the F24 function key.
* **static int VK\_F3** –Constant for the F3 function key.
* **static int VK\_F4** –Constant for the F4 function key.
* **static int VK\_F5** — Constant for the F5 function key.
* **static int VK\_F6** –Constant for the F6 function key.
* **static int VK\_F7** –Constant for the F7 function key.
* **static int VK\_F8** –Constant for the F8 function key.
* **static int VK\_F9** –Constant for the F9 function key.
* **static int VK\_FINAL**
* **static int VK\_FIND**
* **static int VK\_FULL\_WIDTH** –Constant for the Full-Width Characters function key.
* **static int VK\_G**
* **static int VK\_GREATER**
* **static int VK\_H**
* **static int VK\_HALF\_WIDTH** –Constant for the Half-Width Characters function key.
* **static int VK\_HELP**
* **static int VK\_HIRAGANA** –Constant for the Hiragana function key.
* **static int VK\_HOME**
* **static int VK\_I**
* **static int VK\_INPUT\_METHOD\_ON\_OFF** — Constant for the input method on/off key.
* **static int VK\_INSERT**
* **static int VK\_INVERTED\_EXCLAMATION\_MARK** –Constant for the inverted exclamation mark key.
* **static int VK\_J**
* **static int VK\_JAPANESE\_HIRAGANA** –Constant for the Japanese-Hiragana function key.
* **static int VK\_JAPANESE\_KATAKANA** –Constant for the Japanese-Katakana function key.
* **static int VK\_JAPANESE\_ROMAN** –Constant for the Japanese-Roman function key.
* **static int VK\_K**
* **static int VK\_KANA**
* **static int VK\_KANA\_LOCK** — Constant for the locking Kana function key.
* **static int VK\_KANJI**
* **static int VK\_KATAKANA** –Constant for the Katakana function key.
* **static int VK\_KP\_DOWN** — Constant for the numeric keypad down arrow key.
* **static int VK\_KP\_LEFT** –Constant for the numeric keypad left arrow key.
* **static int VK\_KP\_RIGHT** –Constant for the numeric keypad right arrow key.
* **static int VK\_KP\_UP** –Constant for the numeric keypad up arrow key.
* **static int VK\_L**
* **static int VK\_LEFT** –Constant for the non-numpad left arrow key.
* **static int VK\_LEFT\_PARENTHESIS** –Constant for the “(” key.
* **static int VK\_LESS**
* **static int VK\_M**
* **static int VK\_META**
* **static int VK\_MINUS** — Constant for the minus key, “-”
* **static int VK\_MODECHANGE**
* **static int VK\_MULTIPLY**
* **static int VK\_N**
* **static int VK\_NONCONVERT** –Constant for the Don’t Convert function key.
* **static int VK\_NUM\_LOCK**
* **static int VK\_NUMBER\_SIGN** –Constant for the “#” key.
* **static int VK\_NUMPAD0**
* **static int VK\_NUMPAD1**
* **static int VK\_NUMPAD2**
* **static int VK\_NUMPAD3**
* **static int VK\_NUMPAD4**
* **static int VK\_NUMPAD5**
* **static int VK\_NUMPAD6**
* **static int VK\_NUMPAD7**
* **static int VK\_NUMPAD8**
* **static int VK\_NUMPAD9**
* **static int VK\_O**
* **static int VK\_OPEN\_BRACKET** –Constant for the open bracket key, “[”
* **static int VK\_P**
* **static int VK\_PAGE\_DOWN**
* **static int VK\_PAGE\_UP**
* **static int VK\_PASTE**
* **static int VK\_PAUSE**
* **static int VK\_PERIOD** –Constant for the period key, “.”
* **static int VK\_PLUS** — Constant for the “+” key.
* **static int VK\_PREVIOUS\_CANDIDATE** — Constant for the Previous Candidate function key.
* **static int VK\_PRINTSCREEN**
* **static int VK\_PROPS**
* **static int VK\_Q**
* **static int VK\_QUOTE**
* **static int VK\_QUOTEDBL**
* **static int VK\_R**
* **static int VK\_RIGHT** — Constant for the non-numpad right arrow key.
* **static int VK\_RIGHT\_PARENTHESIS** –Constant for the “)” key.
* **static int VK\_ROMAN\_CHARACTERS** –Constant for the Roman Characters function key.
* **static int VK\_S**
* **static int VK\_SCROLL\_LOCK**
* **static int VK\_SEMICOLON** — Constant for the semicolon key, “;”
* **static int VK\_SEPARATER** –This constant is obsolete, and is included only for backwards compatibility.
* **static int VK\_SEPARATOR** –Constant for the Numpad Separator key.
* **static int VK\_SHIFT**
* **static int VK\_SLASH** — Constant for the forward slash key, “/”
* **static int VK\_SPACE**
* **static int VK\_STOP**
* **static int VK\_SUBTRACT**
* **static int VK\_T**
* **static int VK\_TAB**
* **static int VK\_U**
* **static int VK\_UNDEFINED** — This value is used to indicate that the keyCode is unknown.
* **static int VK\_UNDERSCORE** –Constant for the “\_” key.
* **static int VK\_UNDO**
* **static int VK\_UP** –Constant for the non-numpad up arrow key.
* **static int VK\_V**
* **static int VK\_W**
* **static int VK\_WINDOWS** –Constant for the Microsoft Windows “Windows” key.
* **static int VK\_X**
* **static int VK\_Y**
* **static int VK\_Z**
* **WT MouseEvent Class**

Ця подія генерується об’єктом-компонентом для подій миші та подій руху миші.

* a mouse button is pressed
* a mouse button is released
* a mouse button is clicked (pressed and released)
* a mouse cursor enters the unobscured part of component’s geometry
* a mouse cursor exits the unobscured part of component’s geometry
* a mouse is moved
* the mouse is dragged
* a mouse button is pressed
* a mouse button is released
* a mouse button is clicked (pressed and released)
* a mouse cursor enters the unobscured part of component’s geometry
* a mouse cursor exits the unobscured part of component’s geometry
* a mouse is moved
* the mouse is dragged

***Оголошення класу*** **java.awt.event.MouseEvent** class:

public class MouseEvent

extends InputEvent

**Поля**

* **static int BUTTON1** –Indicates mouse button #1; used by getButton()
* **static int BUTTON2** –Indicates mouse button #2; used by getButton()
* **static int BUTTON3** –Indicates mouse button #3; used by getButton()
* **static int MOUSE\_CLICKED** –The “mouse clicked” event
* **static int MOUSE\_DRAGGED** –The “mouse dragged” event
* **static int MOUSE\_ENTERED** –The “mouse entered” event
* **static int MOUSE\_EXITED** –The “mouse exited” event
* **static int MOUSE\_FIRST** –The first number in the range of ids used for mouse events
* **static int MOUSE\_LAST** — The last number in the range of ids used for mouse events
* **static int MOUSE\_MOVED** –The “mouse moved” event
* **static int MOUSE\_PRESSED** — The “mouse pressed” event
* **static int MOUSE\_RELEASED** –The “mouse released” event
* **static int MOUSE\_WHEEL** –The “mouse wheel” event
* **static int NOBUTTON** –Indicates no mouse buttons; used by getButton()
* **static int VK\_WINDOWS** –Constant for the Microsoft Windows “Windows” key.
* **static int BUTTON1** –Indicates mouse button #1; used by getButton()
* **static int BUTTON2** –Indicates mouse button #2; used by getButton()
* **static int BUTTON3** –Indicates mouse button #3; used by getButton()
* **static int MOUSE\_CLICKED** –The “mouse clicked” event
* **static int MOUSE\_DRAGGED** –The “mouse dragged” event
* **static int MOUSE\_ENTERED** –The “mouse entered” event
* **static int MOUSE\_EXITED** –The “mouse exited” event
* **static int MOUSE\_FIRST** –The first number in the range of ids used for mouse events
* **static int MOUSE\_LAST** — The last number in the range of ids used for mouse events
* **static int MOUSE\_MOVED** –The “mouse moved” event
* **static int MOUSE\_PRESSED** — The “mouse pressed” event
* **static int MOUSE\_RELEASED** –The “mouse released” event
* **static int MOUSE\_WHEEL** –The “mouse wheel” event
* **static int NOBUTTON** –Indicates no mouse buttons; used by getButton()
* **static int VK\_WINDOWS** –Constant for the Microsoft Windows “Windows” key.
* **AWT TextEvent Class**

Об’єкт цього класу представляє текстові події. TextEvent генерується, коли символ вводиться в текстові поля або текстову область. Екземпляр TextEvent не містить символів, які наразі містяться в текстовому компоненті, який створив подію, а ми надаємо інші методи для отримання цієї інформації.

*Оголошення класу* **java.awt.event.TextEvent** class:

public class TextEvent

extends AWTEvent

**Поля**

* **static int TEXT\_FIRST** –The first number in the range of ids used for text events.
* **static int TEXT\_LAST** –The last number in the range of ids used for text events.
* **static int TEXT\_VALUE\_CHANGED** –This event id indicates that object’s text changed.
* **static int TEXT\_FIRST** –The first number in the range of ids used for text events.
* **static int TEXT\_LAST** –The last number in the range of ids used for text events.
* **static int TEXT\_VALUE\_CHANGED** –This event id indicates that object’s text changed.
* **AWT WindowEvent Class**

Об’єкт цього класу представляє зміну стану вікна. Ця подія низького рівня генерується об’єктом Window, коли його відкривають, закривають, активують, дезактивують, позначають значком або дезображують значок, або коли фокус переміщується в або з нього вікно.

***Оголошення класу* java.awt.event.WindowEvent** class:

public class WindowEvent

extends ComponentEvent

**Поля**

* **static int WINDOW\_ACTIVATED** –The window-activated event type.
* **static int WINDOW\_CLOSED** — The window closed event.
* **static int WINDOW\_CLOSING** — The “window is closing” event.
* **static int WINDOW\_DEACTIVATED** — The window-deactivated event type.
* **static int WINDOW\_DEICONIFIED** — The window deiconified event type.
* **static int WINDOW\_FIRST** — The first number in the range of ids used for window events.
* **static int WINDOW\_GAINED\_FOCUS** — The window-gained-focus event type.
* **static int WINDOW\_ICONIFIED** — The window iconified event.
* **static int WINDOW\_LAST** — The last number in the range of ids used for window events.
* **static int WINDOW\_LOST\_FOCUS** — The window-lost-focus event type.
* **static int WINDOW\_OPENED** — The window opened event.
* **static int WINDOW\_STATE\_CHANGED** — The window-state-changed event type.
* **static int WINDOW\_ACTIVATED** –The window-activated event type.
* **static int WINDOW\_CLOSED** — The window closed event.
* **static int WINDOW\_CLOSING** — The “window is closing” event.
* **static int WINDOW\_DEACTIVATED** — The window-deactivated event type.
* **static int WINDOW\_DEICONIFIED** — The window deiconified event type.
* **static int WINDOW\_FIRST** — The first number in the range of ids used for window events.
* **static int WINDOW\_GAINED\_FOCUS** — The window-gained-focus event type.
* **static int WINDOW\_ICONIFIED** — The window iconified event.
* **static int WINDOW\_LAST** — The last number in the range of ids used for window events.
* **static int WINDOW\_LOST\_FOCUS** — The window-lost-focus event type.
* **static int WINDOW\_OPENED** — The window opened event.
* **static int WINDOW\_STATE\_CHANGED** — The window-state-changed event type.
* **EventListner AWT**

EventListner представляє інтерфейси, що відповідають за обробку подій. Java надає нам різні класи прослуховувачів подій. У цьому пункті я також наведу лише деякі з них. Кожен метод EventListner має єдиний аргумент як об'єкт, який є підкласом EventObject. Наприклад, EventListner миші приймаємо екземпляр MouseEvent, де MouseEvent є похідним від EventObject.

**Інтерфейс EventListner**

Це інтерфейс маркера, який має розширювати кожен інтерфейс слухача. Цей клас визначено у пакеті java.util.

***Оголошення класу* java.util.EventListener** :

public interface EventListener

**Інтерфейси EventListner AWT:**

* **AWT ActionListener Interface**

Клас, який обробляє ActionEvent, має реалізовувати цей інтерфейс. Об’єкт цього класу має бути зареєстрований у компоненті. Об’єкт можна зареєструвати за допомогою методу addActionListener(). Коли відбувається подія дії, викликається метод actionPerformed цього об’єкта.

***Оголошення класу* java.awt.event.ActionListener** interface:

public interface ActionListener

extends EventListener.

* **AWT ComponentListener Interface**

Клас, який обробляє ComponentEvent, має реалізувати цей інтерфейс. Об’єкт цього класу має бути зареєстрований у компоненті. Об’єкт можна зареєструвати за допомогою методу addComponentListener(). Подія компонента надається лише для інформації.

***Оголошення класу* java.awt.event.ComponentListener** interface:

public interface ComponentListener

extends EventListener.

* **AWT ItemListener Interface**

Клас, який обробляє ItemEvent, має реалізовувати цей інтерфейс. Об’єкт цього класу має бути зареєстрований у компоненті. Об’єкт можна зареєструвати за допомогою методу addItemListener(). Коли відбувається подія дії, викликається метод itemStateChanged цього об’єкта.

***Оголошення класу* java.awt.event.ItemListener** interface:

public interface ItemListener

extends EventListener.

* **AWT KeyListener Interface**

Клас, який обробляє KeyEvent, має реалізовувати цей інтерфейс. Об’єкт цього класу має бути зареєстрований у компоненті. Об’єкт можна зареєструвати за допомогою методу addKeyListener().

***Оголошення класу* java.awt.event.KeyListener** interface:

public interface KeyListener

extends EventListener.

* **AWT MouseListener Interface**

Клас, який обробляє MouseEvent, має реалізовувати цей інтерфейс. Об’єкт цього класу має бути зареєстрований у компоненті. Об’єкт можна зареєструвати за допомогою методу addMouseListener().

***Оголошення класу*** **java.awt.event.MouseListener** interface:

public interface MouseListener

extends EventListener.

* **AWT TextListener Interface**

Клас, який обробляє TextEvent, має реалізовувати цей інтерфейс. Об’єкт цього класу має бути зареєстрований у компоненті. Об’єкт можна зареєструвати за допомогою методу addTextListener().

***Оголошення класу* java.awt.event.TextListener** interface:

public interface TextListener

extends EventListener.

* **AWT WindowListener Interface**

Клас, який обробляє WindowEvent, має реалізовувати цей інтерфейс. Об’єкт цього класу має бути зареєстрований у компоненті. Об’єкт можна зареєструвати за допомогою методу addWindowListener().

***Оголошення класу*** **java.awt.event.WindowListener** interface:

public interface WindowListener

extends EventListener.

* **AWT AdjustmentListener Interface**

Інтерфейс AdjustmentListener використовується для отримання подій коригування. Клас, який обробляє події коригування, повинен реалізовувати цей інтерфейс.

***Оголошення класу* java.awt.event.AdjustmentListener** interface:

public interface AdjustmentListener

extends EventListener.

* **AWT ContainerListener Interface**

Інтерфейс ContainerListener використовується для отримання подій контейнера. Клас, який обробляє події контейнера, повинен реалізовувати цей інтерфейс.

***Оголошення класу* java.awt.event.ContainerListener** interface:

public interface ContainerListener

extends EventListener.

**Додатки**

**Приклади програм створення елемнетів**

**Додаток № 1**

public class AwtFrame {  
 public static void main(String[] args) {  
 FrameWin fw = new FrameWin("Example of frame");  
 fw.setVisible(true);  
 }  
}

import java.awt.FlowLayout;  
import java.awt.Frame;  
import java.awt.Label;  
import java.awt.event.WindowEvent;  
import java.awt.event.WindowListener;  
public class FrameWin extends Frame implements WindowListener {  
 public FrameWin(String FrameTitle){  
 super(FrameTitle);  
 setLayout(new FlowLayout());  
  
 //Create two label  
 Label L1 = new Label("There are two label");  
 Label L2 = new Label("both contain text");  
  
 //Add Label to the  
 add(L1);  
 add(L2);  
  
 //Set Size of the Frame  
 setSize(400, 300);  
 setLocation(100,100);  
  
 //Register with the Listener  
 addWindowListener(this);  
  
 }  
 //Implement the Listeners  
 public void windowOpened(WindowEvent e) {}  
 public void windowClosing(WindowEvent e) {  
 this.dispose();  
 }  
 public void windowClosed(WindowEvent e) {}  
 public void windowIconified(WindowEvent e) {}  
 public void windowDeiconified(WindowEvent e) {}  
 public void windowActivated(WindowEvent e) {}  
 public void windowDeactivated(WindowEvent e) {}  
}

**Додаток № 2**

import java.awt.\*;  
public class PanelExample {  
 PanelExample() {  
 Frame f= new Frame("Panel Example");  
 Panel panel=new Panel();  
 panel.setBounds(40,80,200,200);  
 panel.setBackground(Color.*gray*);  
 Button b1=new Button("Button 1");  
 b1.setBounds(50,100,80,30);  
 b1.setBackground(Color.*yellow*);  
 Button b2=new Button("Button 2");  
 b2.setBounds(100,100,80,30);  
 b2.setBackground(Color.*green*);  
 panel.add(b1); panel.add(b2);  
 f.add(panel);  
 f.setSize(400,400);  
 f.setLayout(null);  
 f.setVisible(true);  
 }  
 public static void main(String[] args)  
 {  
 new PanelExample();  
 }  
}

**Додаток № 3**

import java.awt.\*;  
import java.awt.event.\*;  
  
public class AwtContainer {  
 private Frame mainFrame;  
 private Label headerLabel;  
 private Label statusLabel;  
 private Panel controlPanel;  
 private Label msglabel;  
  
 public AwtContainer(){  
 prepareGUI();  
 }  
  
 public static void main(String[] args){  
 AwtContainer awtContainerDemo = new AwtContainer();  
 awtContainerDemo.showFrameDemo();  
 }  
  
 private void prepareGUI(){  
 mainFrame = new Frame("Examples");  
 mainFrame.setSize(400,400);  
 mainFrame.setLayout(new GridLayout(3, 1));  
 mainFrame.addWindowListener(new WindowAdapter() {  
 public void windowClosing(WindowEvent windowEvent){  
 System.*exit*(0);  
 }  
 });  
 headerLabel = new Label();  
 headerLabel.setAlignment(Label.*CENTER*);  
 statusLabel = new Label();  
 statusLabel.setAlignment(Label.*CENTER*);  
 statusLabel.setSize(350,100);  
  
 msglabel = new Label();  
 msglabel.setAlignment(Label.*CENTER*);  
 msglabel.setText("frame");  
  
 controlPanel = new Panel();  
 controlPanel.setLayout(new FlowLayout());  
  
 mainFrame.add(headerLabel);  
 mainFrame.add(controlPanel);  
 mainFrame.add(statusLabel);  
 mainFrame.setVisible(true);  
 }  
  
 private void showFrameDemo(){  
 headerLabel.setText("Container in action: Frame");  
  
 final Frame frame = new Frame();  
 frame.setSize(300, 300);  
 frame.setLayout(new FlowLayout());  
 frame.add(msglabel);  
 frame.addWindowListener(new WindowAdapter() {  
 public void windowClosing(WindowEvent windowEvent){  
 frame.dispose();  
 }  
 });  
 Button okButton = new Button("Open a Frame");  
  
 okButton.addActionListener(e -> {  
 statusLabel.setText("A Frame shown to the user.");  
 frame.setVisible(true);  
 });  
 controlPanel.add(okButton);  
  
 mainFrame.setVisible(true);  
 }  
}

**Додаток № 4**

import java.awt.\*;  
import java.awt.event.\*;  
  
public class AwtControl {  
  
 private Frame mainFrame;  
 private Label headerLabel;  
 private Panel controlPanel;  
  
 public AwtControl(){  
 prepareGUI();  
 }  
  
 public static void main(String[] args){  
 AwtControl awtControl = new AwtControl();  
 awtControl.showDialogDemo();  
 }  
  
 private void prepareGUI(){  
 mainFrame = new Frame("Examples");  
 mainFrame.setSize(400,400);  
 mainFrame.setLayout(new GridLayout(3, 1));  
 mainFrame.addWindowListener(new WindowAdapter() {  
 public void windowClosing(WindowEvent windowEvent){  
 System.*exit*(0);  
 }  
 });  
 headerLabel = new Label();  
 headerLabel.setAlignment(Label.*CENTER*);  
 Label statusLabel = new Label();  
 statusLabel.setAlignment(Label.*CENTER*);  
 statusLabel.setSize(350,100);  
  
 controlPanel = new Panel();  
 controlPanel.setLayout(new FlowLayout());  
  
 mainFrame.add(headerLabel);  
 mainFrame.add(controlPanel);  
 mainFrame.add(statusLabel);  
 mainFrame.setVisible(true);  
 }  
  
 private void showDialogDemo(){  
 headerLabel.setText("Control in action: Dialog");  
 Button showAboutDialogButton = new Button("Show About Dialog");  
 showAboutDialogButton.addActionListener(e -> {  
 AboutDialog aboutDialog = new AboutDialog(mainFrame);  
 aboutDialog.setVisible(true);  
 });  
  
 controlPanel.add(showAboutDialogButton);  
 mainFrame.setVisible(true);  
 }  
  
 static class AboutDialog extends Dialog {  
 public AboutDialog(Frame parent){  
 super(parent, true);  
 setBackground(Color.*gray*);  
 setLayout(new BorderLayout());  
 Panel panel = new Panel();  
 panel.add(new Button("Close"));  
 add("South", panel);  
 setSize(200,200);  
  
 addWindowListener(new WindowAdapter() {  
 public void windowClosing(WindowEvent windowEvent){  
 dispose();  
 }  
 });  
 }  
  
 public boolean action(Event evt, Object arg){  
 if(arg.equals("Close")){  
 dispose();  
 return true;  
 }  
 return false;  
 }  
  
 public void paint(Graphics g){  
 g.setColor(Color.*white*);  
 g.drawString("Point", 25,70 );  
 g.drawString("Version 1.0", 60, 90);  
 }  
 }  
}

**Додаток № 5**

import java.awt.\*;  
class TextFieldDemo{  
 public static void main(String args[]){  
 Frame TextF\_f= new Frame("TextField");  
 TextField text1,text2;  
 text1=new TextField("Enter few words");  
 text1.setBounds(60,100, 230,40);  
 text2=new TextField("");  
 text2.setBounds(60,150, 230,40);  
 TextF\_f.add(text1);  
 TextF\_f.add(text2);  
 TextF\_f.setSize(500,500);  
 TextF\_f.setLayout(null);  
 TextF\_f.setVisible(true);  
 }  
}

**Додаток № 6**

import java.awt.\*;  
public class Checkbox {  
 Checkbox(){  
 Frame checkB\_f= new Frame("Checkbox");  
 java.awt.Checkbox ckbox1 = new java.awt.Checkbox("Yes", true);  
 ckbox1.setBounds(100,100, 60,60);  
 java.awt.Checkbox ckbox2 = new java.awt.Checkbox("No");  
 ckbox2.setBounds(100,150, 60,60);  
 checkB\_f.add(ckbox1);  
 checkB\_f.add(ckbox2);  
 checkB\_f.setSize(400,400);  
 checkB\_f.setLayout(null);  
 checkB\_f.setVisible(true);  
 }  
 public static void main(String args[])  
 {  
 new Checkbox();  
 }  
}

**Додаток № 7**

import java.awt.\*;  
public class CheckboxGroup {  
 CheckboxGroup(){  
 Frame ck\_groupf= new Frame("CheckboxGroup");  
 java.awt.CheckboxGroup obj = new java.awt.CheckboxGroup();  
 Checkbox ckBox1 = new Checkbox("Yes", obj, true);  
 ckBox1.setBounds(100,100, 50,50);  
 Checkbox ckBox2 = new Checkbox("No", obj, false);  
 ckBox2.setBounds(100,150, 50,50);  
 ck\_groupf.add(ckBox1);  
 ck\_groupf.add(ckBox2);  
 ck\_groupf.setSize(400,400);  
 ck\_groupf.setLayout(null);  
 ck\_groupf.setVisible(true);  
 }  
 public static void main(String args[])  
 {  
 new CheckboxGroup();  
 }  
}

**Додаток № 8**

import javax.swing.\*;  
import javax.swing.event.ChangeEvent;  
import javax.swing.event.ChangeListener;  
  
  
class Color1 implements ChangeListener {  
 JFrame window;  
 JColorChooser colorChooser ;  
 JPanel colorChooserPanel, colorPanel;  
  
 public static void main(String[] args) {  
 new Color1();  
  
 }  
  
 public Color1(){  
 /\*  
 код для виклику вікна, та зображення різних кольорових моделей в цьомі вікно  
 \*/  
 window = new JFrame();  
 window.setSize(1200, 700);  
 window.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 window.getContentPane().setBackground(java.awt.Color.*black*);  
 window.setLayout(null);  
  
 colorChooserPanel = new JPanel();  
 colorChooserPanel.setBounds(50, 50, 600, 300);  
 colorChooserPanel.setBackground(java.awt.Color.*BLACK* );  
 window.add(colorChooserPanel);  
  
 colorChooser = new JColorChooser();  
 colorChooser.getSelectionModel().addChangeListener(this);  
  
 colorChooser.setPreviewPanel(new JPanel());  
  
 colorChooserPanel.add(colorChooser);  
  
 colorPanel = new JPanel();  
 colorPanel.setBounds(200, 420, 400, 100);  
 colorPanel.setBackground(java.awt.Color.*white*);  
 window.add(colorPanel);  
  
 window.setVisible(true);  
  
  
 }  
  
 public void stateChanged(ChangeEvent e){  
 /\*  
 код для відображення кольору на окремому квадраті, щоб можна було побачити вживу цей колір  
 також виведення компонентів цього кольору(прописано тільки для Rgb, но інші аналогічно робляться,  
 тому немає сенсу прописувати ідентичний код)  
 \*/  
  
 java.awt.Color newColor = colorChooser.getColor();  
 colorPanel.setBackground(newColor);  
 System.*out*.println("red: " + newColor.getRed() + "\n" + "green: " + newColor.getGreen() + "\n" +  
 "blue: " + newColor.getBlue() + "\n" + "alpha: " + newColor.getAlpha() + "\n");  
  
 JButton b = new JButton();  
 b.setBounds(450,350,200,50);  
 window.add(b);  
 b.setBackground(newColor);  
 b.setVisible(true);  
  
 }  
  
  
}

**Додаток № 9**

import java.awt.\*;  
public class Choice {  
 Choice()  
 {  
 Frame choice\_f= new Frame();  
 java.awt.Choice obj=new java.awt.Choice();  
 obj.setBounds(80,80, 100,100);  
 obj.add("Red");  
 obj.add("Blue");  
 obj.add("Black");  
 obj.add("Pink");  
 obj.add("White");  
 obj.add("Green");  
 choice\_f.add(obj);  
 choice\_f.setSize(400,400);  
 choice\_f.setLayout(null);  
 choice\_f.setVisible(true);  
 }  
 public static void main(String args[])  
 {  
 new Choice();  
 }  
}