КЛАСИ AWT

Лекція 8

Доцент кафедри Інформатики
Сінельнікова Т.Ф.

Зміст

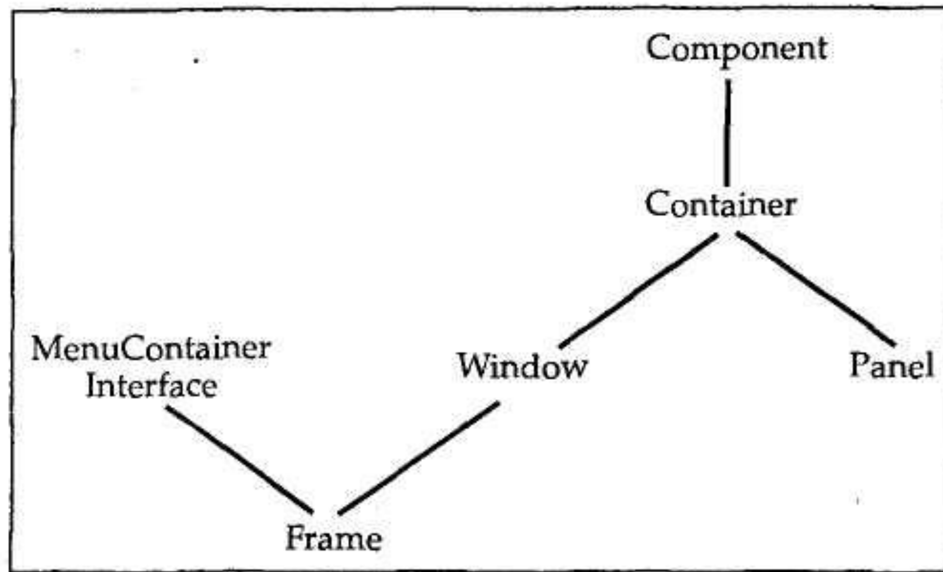
- AWT
- Основи віконної графіки
- Клас Component
- Клас Container
- Клас Зфтуд
- Клас Window
- Клас Frame
- Клас Canvas
- Робота з фреймовими вікнами
- Установка розмірів вікна
- Приховування та показ вікна
- Установка заголовка вікна
- Закриття фрейм-вікна
- Створення фрейм-вікна в аплеті
- Обробка подій фрейм-вікна
- Відображення інформації у вікні
- Робота з графікою
- Малювання ліній
- Малювання прямокутників
- Малювання еліпсів і кругів
- Малювання дуг
- Малювання багатокутників
- Установка розмірів графіки
- Методи управління кольором
- Використання тону, насиченості і яскравості
- Установка поточного кольору графіки
- Робота зі шрифтами
- Змінні класу Font
- Створення та вибір шрифту
- Управління текстовим виводом
- Відображення багаторядкового тексту
- Використання елементів управління, менеджерів компонування і меню AWT
- Елементи управління
- Додавання та видалення елементів управління
- Текстові мітки
- Використання кнопок
- Застосування прапорців
- Клас CheckboxGroup
- Елемент управління Choice
- Використання списків
- Управління смугами прокрутки
- Використання класу TextField
- Використання TextArea
- Поняття менеджера компоновки
- Менеджер FlowLayout
- Клас BorderLayout
- Використання вставок
- Менеджер GridLayout
- Клас CardLayout
- Панелі меню
- Діалогові вікна
- Клас FileDialog
- Робота з зображеннями
- Формати графічних файлів
- Створення об'єкта зображення
- Завантаження зображення
- Перегляд зображення
- Інтерфейс ImageObserver
- Подвійна буферизація
- Клас MediaTracker
- Інтерфейс ImageProducer
- Виробник зображень MemoryImageSource
- Інтерфейс ImageConsumer
- Клас PixelGrabber
- Клас ImageFilter
- Фільтр CropImageFilter
- Фільтр RGBImageFilter

AWT

AWT - Скорочення Abstract Window Toolkit (Абстрактний віконний інтерфейс).

Основи віконної графіки

AWT визначає вікна згідно ієрархії класів, яка з кожним рівнем додає функціональні можливості і специфіку.
Два найбільш загальних типу вікон є похідними від типу Panel, Який користується апплетами, і від типу Frame, Який створює стандартне вікно.
Багато що з функціональних можливостей цих вікон отримано від їх батьківських класів.



Клас Component

Самий верхній в AWT-Ієрархії - клас Component.
Це абстрактний клас, який інкапсулює всі атрибути візуального компонента.
Всі елементи інтерфейсу користувача, які відображені на екрані і взаємодіють з користувачем, - це підкласи Component.
У класі Component визначено більше сотні public-Методів, які є відповідальними за управління подіями, такими як введення за допомогою миші та клавіатури, позиціонування і зміна розмірів вікна, перемальовування і т. д. (багато з цих методів вже використовувалися при створенні аплетів в главах 19 і 20).
Об'єкт класу Component відповідає за запам'ятовування поточних кольорів переднього плану і фону і вибраного текстового шрифту.

Клас Container

Клас Container є підкласом Component.
Він містить додаткові методи, які дозволяють вкладати в нього інші Component-Об'єкти.
Усередині класу Container можуть зберігатися і його власні об'єкти. Це робить container *системою багаторівневого включення.*
Контейнер відповідає за розміщення (тобто позиціонування) будь-яких компонентів, які він містить.

Клас Panel

Клас Panel - Конкретний підклас Container.
Він не додає будь-яких нових методів. Це просто реалізація класу Container.
ПроPanel можна мислити як про рекурсивно вкладається конкретному екранному компоненті. Panel - Суперклас для Applet.
Коли екранний висновок направляється до аплету, він малюється на поверхні об'єкту Panel.
По суті, об'єкт Panel - Це вікно, яке не містить області заголовка, рядка меню і обрамлення.

Клас Window

Клас Window створює вікно верхнього рівня. *Вікно верхнього рівня* не міститься в будь-якому іншому об'єкті.
Воно знаходиться безпосередньо на Робочому столі.
Взагалі, ви не будете створювати Window-Об'єкти безпосередньо.
Замість цього, ви будете використовувати підклас Window з ім'ям Frame.

Клас Frame

Клас Frame інкапсулює те, що зазвичай представляють як "вікно".
Це підклас Window і його вікно має рядок заголовка, рядок меню, обрамлення і кути, що змінюють розміри вікна.
Якщо ви спробуєте створити в аплеті об'єкт типу Frame, То він буде містити повідомлення виду "Warning:Applet Window", Що говорить про те, що вікно було створено аплетом. Це повідомлення попереджає користувачів, що вікно, яке вони бачать, було запущено аплетом, а не програмним забезпеченням, що виконується на їх комп'ютері.

Клас Canvas

Хоча Canvas не є частиною container-Ієрархії, існує ще один тип вікна, який може виявитися корисним. Це - клас Canvas.

Canvas-Об'єкт моделює *пусте вікно,* в якому можна малювати.

Робота з фреймовими вікнами

Тип вікна, який ви будете найчастіше створювати, є похідним від Frame.
Він використовується для створення вікон верхнього рівня і дочірніх вікон для аплетів і додатків.
Фрейм - Це вікно із стандартним стилем (з заголовком, меню, обрамленням керуючими куточками). Frame підтримує два конструктора:

- **Frame ()**
- **Frame (String *заголовок)***

Перша форма створює стандартне вікно, яке не містить заголовка. Друга форма створює вікно з заголовком, зазначеним у параметрі *заголовок.*

Установка розмірів вікна

Щоб встановити розміри вікна, використовується метод setSize(). Існує дві форми цього методу (з різними списками параметрів):

- **void setSize (int *newWidth,*   int *newHeight)***
- **void setSize (Dimension *newSize)***

Новий розмір вікна специфікується параметрами **newWidth inewHeight** (Перша форма), або полями width iheight об'єкту класу Dimension, переданими параметру ***newSize.*** Розміри задаються в пікселах.
Метод getsize () використовується для отримання поточного розміру вікна. Його сигнатура:

- **Dimension getsize ()**

Даний метод повертає поточний розмір вікна в полях width і height об'єкту класу Dimension.

Приховування та показ вікна

Після створення фрейм-вікно залишається невидимим до тих пір, поки ви не викличете метод setVisible () . Сигнатура цього методу має вигляд:

- **void setVisible (boolean *visibleFlag*)**

Компонент стає видимим, якщо параметр цього методу отримує значення true, інакше він залишається прихованим (невидимим).

Установка заголовка вікна

Можна змінити заголовок фрейм-вікна, якщо викликати метод setTitie (). Він має такий формат:

- **void setTitle (String *newTitle*)**

де *newTitle* - Новий заголовок вікна.

Закриття фрейм-вікна

Коли фрейм-вікно закривається, програма повинна видалити це вікно з екрана, викликаючи setvisible () З аргументом false:

- **setVisible (false);**

Щоб перехопити подія закриття вікна, потрібно реалізувати метод windowClosin () інтерфейсу windowListener. Всередині windowClosing () необхідно видалити вікно з екрана за допомогою виклику setVisible (false).

Створення фрейм-вікна в аплеті

Створити нове фрейм-вікно всередині аплета насправді дуже просто. Спочатку створіть підклас Frame. Потім перевизначите необхідні для роботи з вікном стандартні методи, такі як init (), start (), stop () I paint (). Нарешті, реалізуйте метод wmdowClosing () Інтерфейсу WindowListener, викликаючи setvisible (false), щоб закрити вікно.
Визначивши підклас Frame, потрібно створити об'єкт цього класу. Однак новозбудоване фрейм-вікно спочатку не буде видимим (на екрані). Щоб зробити його видимим, потрібно викликати setvisible () з аргументом false. При створенні вікна задають висоту і ширину за замовчуванням. Щоб встановити необхідний розмір вікна, потрібно викликати метод setsize ().

Створення фрейм-вікна в аплеті

```
/ / Створює дочірнє фрейм-вікно всередині аплета.
import j ava.awt. *;
import java.awt.event. *;
import j ava.applet. *;
/ *
<applet code="AppletFrame" width=300 height=50>
</ Applet> * /
/ / Створити підклас Frame, class SampleFrame
extends Frame і SampleFrame (String title) {super (title);
/ / Створити об'єкт для обробки window-Подій
MyWindowAdapter adapter =new MyWindowAdapter
(this);
/ / Реєструвати його для прийому цих подій
addWindowListener (adapter);
}
public void paint (Graphics g) {
g.drawstring ("This is in frame window", 10, 40);}}
class MyWindowAdapter extends WindowAdapter
{SampleFrame SampleFrame; public MyWindowAdapter
(SampleFrame SampleFrame) {
this.SampleFrame =SampleFrame; }public void
windowClosing (WindowEvent we) {
SampleFrame.setvisible (false);}}
```

```
/ / Створити фрейм-вікно.
public class AppletFrarae extends Applet {Frame f;
public void init() {
f=new SampieFrame ("A Frame Window");
f.setSize (250, 250);
f.setVisible (true);} public void start () {
f.setVisible (true);} public void stop () {
f.setVisible (false);} public void paint (Graphics g) {
g.drawstring ("This is in applet window", 10, 20);};
```

Обробка подій фрейм-вікна

Клас Frame успадковує всі можливості свого суперкласу (тобто класу Component).
Це означає, що ви можете керувати (і користуватися) створюваним фрейм-вікном точно так само, як ви керуєте головним вікном аплету.
Наприклад, можна перевизначити paint (), щоб відобразити висновок, викликати repaint (), Коли потрібно відновити вікно, а також перевизначити всі обробники подій.
Всякий раз, коли відбувається подія, пов'язана з вікном, будуть викликатися обробники подій, визначені для цього вікна. Кожне вікно обробляє свої власні події.

Відображення інформації у вікні

У найзагальнішому сенсі вікно є контейнером для різноманітної інформації.
Велика частина коштів AWT орієнтована на підтримку саме цих можливостей.
Далі обговорюються можливості обробки тексту, графіки та шрифтів мовою Java.

Робота з графікою

AWT підтримує багатий набір графічних методів.
Вся графіка малюється щодо вікна.
Це може бути головне або дочірнє вікно аплета, а також вікно автономного програми.
Початок координат кожного вікна-в його верхньому лівому куті і позначається як (0, 0).
Координати визначаються в пікселах.
Весь висновок у вікно виконується через графічний контекст.
Графічний контекст инкапсулирован в класі і виходить двома способами:

- передається аплету, коли викликається один з його численних методів, таких як paint() Або update();
- повертається методом getGraphics () Класу Component.

Клас Graphics визначає ряд функцій малювання.
Кожна форма може бути мальованої або заповненою.
Об'єкти малюються і заповнюються обраним в поточний момент графічним кольором, який за замовчуванням є чорним.
Коли графічний об'єкт перевищує розміри вікна, висновок автоматично буде скорочуватися.

Малювання ліній

Лінії малюються методом drawLine() формату:

- **void drawLina (int startX, int *startY,* int *endX,* int *endY*)**

DrawLine () відображає лінію (в поточному кольорі малювання), яка починається в координатах *startX, startY* і закінчується в endX, *endY.*

Малювання прямокутників

Методи drawRect() ifillRect () відображають відповідно мальований і заповнений прямокутник. Їх формат:

- **void drawRect (int *top*, int *left*, int *width*, int *height*)**
- **void fillRect (int *top*, int *left*, int *width*, int *height*)**

Координати лівого верхнього кута прямокутника - в параметрах *top* і***left, width*** і***height*** -вказують розміри прямокутника (в пікселах).
Щоб малювати округлений прямокутник, використовуйте drawRoundRect () Або filiRoundRect про з форматами:

- **void drawRoundRect (int *top*, int *left*, int *width*, int *height*, int *xDiam*, int *yDiam*)**
- **void filiRoundRect (int top, int *left*, int *width*, int *height*, int *xDiam*, int *yDiam*)**

Малювання еліпсів і кругів

Для малювання еліпса використовуйте drawOvaK), а для його заповнення - filiovaio. Ці методи мають формат:

- **void drawOval (int *top*, int *left*, int *width*, int *height*)**
- **void fillOval (int *top*, int *left*, int *width*, int *height*)**

Еліпс малюється в межах обмежувального прямокутника, чий лівий верхній кут визначається параметрами *top* і*left,* а ширина і висота вказуються в ***width*** і ***height.*** Щоб намалювати коло, як обмежувального прямокутника вказуйте квадрат.

Малювання дуг

Дуги можна малювати методами drawArc () і fillArc (), використовуючи формати:

- **void drawArc (int *top,* int *left,* int *width,* mi height, int *початок,* int *кінець*)**
- **void fillArc (int top, int *left,* int width, int *height,* int *початок,* int *кінець*)**

Дуга обмежена прямокутником, чий лівий верхній кут визначається параметрами ***top, left,*** а ширина і висота - параметрами ***width*** і***height****.*Дуга малюється від ***початку*** до кутового відстані, зазначеного в ***кінець****.*Кути вказуються в градусах і відраховуються від горизонтальної осі проти годинникової стрілки. Дуга малюється проти годинникової стрілки, якщо ***кінець*** позитивний, і за годинниковою стрілкою, якщо *кінець* від'ємний. Тому, щоб намалювати дугу від дванадцятигодинного до шестигодинного положень, початковий кут повинен бути 90 ° і кут розгортки 180 °.

Малювання багатокутників

Фігури довільної форми можна малювати, використовуючи методи drawPolygon() I fillPolygon про з наступними форматами:

- **void drawPoiygon (int x [ ], int y [ ], int *numPoints*)**
- **void fillPolygon (int x [ ], int y [ ], int *numPoints*)**

Кінцеві точки багатокутника визначаються координатними парами, що містяться в масивах x[ ] I y []. Число точок, визначених у цих масивах, вказується параметром **numPoints.** Є альтернативні форми цих методів, в яких багатокутник визначається об'єктом класу Polygon.

Установка розмірів графіки

Часто потрібно встановити розміри графічного об'єкта в певній пропорції з поточними розмірами вікна, в якому він малюється.
Для цього спочатку отримують поточні розміри вікна, викликаючи метод getsize () для віконного об'єкта.
Він повертає розміри вікна, інкапсульовані в об'єкт класу Dimension.
Раз ви знаєте поточні розміри вікна, то можете потрібним чином отмасштабовані графічний висновок.

Методи управління кольором

Java забезпечує переносимість кольору, незалежно від пристрою виводу об'єкта. Колірна система AWT дозволяє вказувати в програмі будь-який бажаний колір.
Більш того, AWT знаходить найкраще узгодження цього кольору із заданими апаратними обмеженнями дисплея, що виконує вашу програму або аплет.
Робота з кольором підтримується класом Color.

Методи управління кольором

- **Color (int *red,* int *green,* int blue)**
- **Color (int *rgbValue)***
- **Color (float red, float green, float blue)**

Перший конструктор одержує (через вказані параметри) три цілих числа, які визначають колір як суміш червоного, зеленого і синього кольору. Ці значення повинні бути між 0 і 255, як у наступному прикладі:

- **new Color (255, 100, 100); / / Світло-червоний**

Другий колірної конструктор отримує одиночне ціле число, яке містить суміш червоного, зеленого і синього кольорів, упаковану в ціле число. Ціле число організовано з червоним кольором - в бітах від 16 до 23, зеленим кольором - в бітах від 8 до 15 і синім кольором - в бітах від 0 до 7. Приклад цього конструктора:

- **int newRed = (0xff000000) | (0xc0 <<16) l(0x00 << 8) l0x00);**
- **Color darkRed =new Color (newRed);**

Третій конструктор отримує три float-Значення (між 0.0 і 1.0), в яких визначається відносна суміш червоного, зеленого і синього кольору.

Використання тону, насиченості і яскравості

Для визначення специфічних квітів використовується дві альтернативні колірні моделі: HSB (Hue-Saturation-Brightness, колірної тон-насиченість-яскравість) і RGB (Red-Green-Blue, червоний-зелений-синій).

Тон (Відтінок) визначається числом між 0.0 і 1.0 (для квітів веселки, розташованих в порядку зростання: червоний, оранжевий, жовтий, зелений, блакитний, синій (індиго) і фіолетовий).

Насиченість - Інша шкала, ранжируваних від 0.0 до 1.0, що представляє зміни тону від світлого (пастельного) до інтенсивного.

Значення *яскравості* також ранжовані від 0.0 до 1.0, де 1 - яскраво-білий, а 0 - чорний.

Використання тону, насичености і яскравості

Клас color поставляє два методи, які виконують взаємні перетворення RGB- iHSB-Моделей:

- **static int HSBtoRGB (float hue, float *saturation,*   float *brightness*)**
- **static float [ ] RGBtoHSB (int *red,*   int *green,*   int *blue*, Float values [ ])**

Метод HSBtoRGB про повертає упаковане RGB-Значення, сумісний з конструктором Color (int).
Метод RGBtoHSB () повертає масив HSB-Значень з плаваючою точкою, відповідних цілим числам RGB. Якщо параметр **values-** НЕ null, то цей масив містить HSB-Значення, які повертаються в зухвалу програму. В іншому випадку створюється новий масив, і в ньому повертаються HSB-Значення.
У будь-якому випадку масив містить тон в елементі з індексом 0, насиченість - в елементі з індексом 1 і яскравість - в елементі з індексом 2.

Установка поточного кольору графіки

За замовчуванням, графічні об'єкти малюються в поточному кольорі переднього плану. Можна змінити цей колір, викликаючи метод setcoior () Класу Graphics:

- **void setColor (Color *newColor*)**

де параметр *nevcoior* визначає новий колір малюнка.

Ви можете отримати поточний колір, викликаючи метод get3olor ():

- **Color getColor ()**

Робота зі шрифтами

Пакет AWT підтримує безліч типів шрифтів.
Шрифти з'явилися з області традиційного набору текстів і стали важливою частиною комп'ютерних документів і дисплеїв. AWT забезпечує гнучкість програмування за рахунок того, що бере на себе операції маніпулювання шрифтами і допускає їх динамічний вибір.
Ім'я сімейства - Загальна назва шрифту, наприклад, Courier (Кур'єр).
Логічне ім'я визначає категорію шрифту, наприклад Monospaced (Фіксованої ширини).
Ім'я гарнітури специфікує певний шрифт, наприклад, Courier Italic (Кур'єр курсивний).
Шрифти інкапсульовані в класі Font.

Змінні класу Font

Змінна	Значення
String name	ім'я шрифта
Float pointSize	Розмір шрифту в пунктах (дробовий)
Intsize	Розмір шрифту в пунктах (цілий)
Int style	Стиль (накреслення) шрифта

Створення та вибір шрифту

Перед вибором нового шрифту потрібно спочатку створити об'єкт класу Font, Який описує цей шрифт.

- **Font (String *fontName*,int *fontStyle,*int *pointSize*)**

Тут *fontName* визначає ім'я бажаного шрифту. Ім'я можна вказувати, використовуючи або логічне ім'я, або ім'я гарнітури. Все середовища Java підтримують наступні шрифти: Dialog,DialogInput,Sans Serif.Serif,Monospaced iSymbol.

Стиль шрифту вказується параметром *fontStyle.*Він може складатися з однієї або декількох констант: Font.plain,Font.bold iFont,italic. Стилі можна комбінувати.

Розмір шрифту вказується параметром *poinSize* (В пунктах).

Щоб використовувати шрифт, який ви створили, слід *вибрати* його за допомогою методу setFont () .

- **void setFont(Font *fontObj*)**

Тут *fontObj -* об'єкт, який містить бажаний шрифт.

Управління текстовим виводом

Для більшості шрифтів не всі символи мають однакову ширину (такі шрифти називають *пропорційними).*
Крім того, висота кожного символу, довжина *виносних елементів* (Звисаючих частин, як у символів gабо p, наприклад) і величина пробілу між горизонтальними рядками змінюється від шрифту до шрифту.
Може бути змінений розмір шрифту (в пунктах).
Враховуючи, що розміри кожного шрифту можуть відрізнятися і що шрифти можуть бути змінені під час виконання програми, повинен існувати певний спосіб для визначення розмірів і різних інших, атрибутів поточного шрифту. Наприклад, для запису одного рядка тексту після іншої необхідно якось дізнатися, яка висота шрифту і скільки пікселів необхідно мати між рядками.

Управління текстовим виводом

Щоб заповнити цю потребу, AWT включає клас FontMetrics, Який інкапсулює різну інформацію про шрифт.
Загальна термінологія, яка використовується при описі шрифтів:

- Висота (Height) - Розмір (від верху до низу) найвищого символу в шрифті.
- Базова лінія (Baseline) - Лінія, по якій вирівняний низ всіх символів.
- Висота надрядкового елемента, асцендер (Ascent) - Відстань від базової лінії до верху символу.
- Висота підрядкового елемента, десцендер (Descent) - Відстань від базової лінії до низу символу.
- Інтерлін'яж (Leading) - Відстані між самим низом одного рядка тексту і самим верхом наступного рядка.

Відображення багаторядкового тексту

Для відображення багаторядкового тексту ваша програма повинна вручну відстежувати поточну позицію виведення.
Коли потрібно вивести новий рядок, координата Yповинна бути зміщена до початку наступного рядка.
Коли рядок відображається, координата Хповинна бути встановлена в точку, де закінчується попередній рядок. Це дозволяє записувати наступний рядок, починаючи з кінця попередньої.
Для визначення інтерліньяжу ви можете використовувати значення, що повертається методом getLeading ().
Щоб визначати повну висоту шрифту, додайте значення, повернене методом getAscent(), До значення, повернетого методом getDescent().
Потім ви можете використовувати ці значення, щоб позиціонувати кожен рядок тексту, яку ви виводите. Щоб почати виведення з кінця попереднього висновку на тому ж рядку, ви повинні знати довжину (в пікселах) кожної відображуваної рядка.

Використання елементів управління, менеджерів компонування і меню AWT

Елементи управління (controls) - Це компоненти, які надають користувачеві різні способи взаємодії з додатком (наприклад, командна *кнопка* (push button)).
Менеджер компоновки (layout manager) Автоматично позиціонує (розміщує, розташовує) компоненти в контейнері. Вид вікна, таким чином, визначається комбінацією елементів управління, що містяться у вікні, і менеджера компоновки, використовуваного для їх розміщення.
На додаток до елементів управління, фрейм-вікно може також включати *рядок меню* (menu bar) Стандартного стилю. Кожен вхід в рядку меню активізує меню, що розкривається елементів, які користувач може вибирати. Рядок меню завжди позиціонується нагорі вікна. Рядки меню, хоча і розрізняються по виду, обробляються схожим способом, що й інші елементи управління.
Хоча компоненти вікна можна позиціонувати вручну, це дуже втомлює. Менеджер компоновки призначений для автоматизації цього завдання.

Елементи управління

AWT підтримує такі типи елементів управління:

- Текстові мітки (Labels)
- Кнопки (Push buttons)
- Прапорці (Check boxes)
- Списки з вибором елементів (Choice lists)
- Списки (Lists)
- Смуги прокрутки (Scroll bars)
- Елементи редагування тексту: текстові поля (Text fields) І текстові області (Text areas).

Елементи управління представлені спеціальними класами пакета AWT, Які є підкласами класу Component.

Додавання та видалення елементів управління

Для включення елемента управління у вікно потрібно додати його до вікна. Для цього необхідно спочатку створити екземпляр бажаного елемента керування і потім додати його до вікна викликом методу add (), Що визначений у класі Container. Метод add () Має кілька форм. У першій частині цієї глави використовується наступна форма:

- **Component add(Component *compObj*)**

Тут ***compObj*** - Примірник елемента керування, який ви хочете додати. Метод повертає посилання на об'єкт, який передається параметром *campObj.*Відразу після додавання елемент управління буде автоматично виводитися на екран всякий раз, коли відображається його батьківське вікно.
Якщо ви захочете видалити елемент керування з вікна, коли він більше не потрібний, викликайте метод remove о, що визначений у класі Container.

- **void remove(Component o*bj*)**

Тут o*bj* - посилання на елемент керування, який потрібно видалити.
Викликаючи метод removeAii (), Можна видалити всі елементи управління.

Текстові мітки

Текстова мітка - Це об'єкт класу Label, Що містить рядок, який вона відображає. Теги - пасивні елементи управління, які не підтримують жодного взаємодії з користувачем.

- **Label ()**
- **Label(String *str*)**
- **Label (String *str,*int *how*)**

Перша версія створює порожню мітку, друга - позначку, яка містить рядок, специфіковані параметром *str.*Цей рядок вирівняна по лівому краю. Третя версія створює позначку, яка містить рядок, специфіковані параметром *str,*використовуючи вирівнювання, вказане в параметрі *how.*Значенням *how* повинна бути одна з трьох констант: Label.LEFT, Label.RIGHT або Label.CENTER.

Текстові мітки

Текст в мітці можна встановлювати або змінювати, використовуючи метод setText (), А поточну мітку можна отримати, викликаючи getText (). Формат цих методів:

- **void setText (String str)**
- **String getText ()**

Параметр *str* вsetText () Визначає нову позначку. Метод getText () Повертає поточну мітку.

Викликаючи метод setAiignmento, Можна встановлювати вирівнювання рядка в межах позначки. Щоб отримати поточне вирівнювання, викличте метод getAlignment ().

- **void setAlignment (int *how*)**
- **int getAlignment ()**

Параметр *how* повинен бути однією з констант вирівнювання, представлення раніше.

Використання кнопок

Кнопки використовуються для введення команд, тому їх часто називають *кнопками команд* або *командними кнопками.*
Кнопка - Це компонент, який містить текстову мітку і генерує подія, коли її натискають. Кнопки є об'єктами класу Button. У класі Button визначаються два конструктора:

- **Button ()**
- **Button(String *str*)**

Перша версія створює порожню кнопку, друга - кнопку з текстовою міткою, яка передається через параметр str.
Після того як кнопка була створена, можна встановити її позначку, викликаючи метод setLabei (). Витягти її мітку можна викликом getLabel ().

- **void setLabel (String str)**
- **String getLabel ()**

Параметр **str** вказує нову позначку для кнопки.

Застосування прапорців

Прапорець (check box) - Це елемент управління, який використовується для включення або виключення деякої функції (режиму, параметра і т. п.).

З кожним прапорцем зв'язується текстова мітка, яка описує, яку опцію представляє прапорець. Ви можете змінювати стан прапорця клацанням миші по ньому.

Застосування прапорців

Прапорці є об'єктами типу Checkbox, Який підтримує наступні конструктори:

- **Checkbox()**
- **Checkbox (String *str*)**
- **Checkbox (String *str,* boolean on)**
- **Checkbox (String *str,* boolean *on,* CheckboxGroup *cbGroup*)**
- **Checkbox (String *str,* CheckboxGroup *cbGroup,* boolean *on*)**

Перша форма створює прапорець, чия текстова мітка спочатку - пробіл. Стан прапорця в цьому випадку - "вимкнений" (off,unchecked).
Друга форма створює прапорець, чия текстова мітка визначається параметром *str.* Стан прапорця тут також "off ".
Третя форма дозволяє встановлювати початковий стан прапорця. Якщо параметр *on* дорівнює true, Створюється прапорець з початковим станом "включений" ("on"); Інакше (коли *on* дорівнює false) Створюється "чистий" прапорець - без маркера перевірки (тобто в стані "off").
Четверті і п'яті форми створюють прапорець, чия текстова мітка визначена параметром str, А група вказана параметром *cbGroup.*Якщо цей прапорець не є частиною групи, то *cbGroup* повинен мати значення null (Порожній посилання). Значення параметра *on* визначає початковий стан прапорця.

Клас *CheckboxGroup*

Можливе створення набору (групи) взаємовиключних прапорців, в якому може бути включений один і тільки один з них. Такі прапорці часто називаються "радіокнопки" *(Radio buttons),* тому що вони діють подібно селектору станцій на автомобільному радіоприймачі - в будь-який момент може бути обрана тільки одна станція.
Щоб створити набір взаємовиключних прапорців, потрібно спочатку визначити групу, до якої вони будуть належати, і потім вказати цю групу, коли прапорці будуть створюватися. Групи прапорців є об'єктами типу checkboxGroup.
Встановити прапорець можна викликом методу setSelectedCheckbox ().

- **Checkbox getSelectedCheckbox ()**
- **void setseiectedcheckbox (Checkbox *which*)**

де ***which*** - Параметр, який вказує прапорець, який потрібно вибрати. При цьому попередньо обраний прапорець буде вимкнений.

Елемент управління *Choice*

Клас Choice використовується для того, щоб створювати список, що розкривається елементів, з яких користувач може робити вибір. Таким чином, елемент управління "вибір" (Choice) має форму меню.
Клас Choice визначає тільки замовчуваний конструктор, який створює порожній список. Щоб додати елемент вибору до списку, викличте метод addItem () або add () .

- **void addItem (String *name*)**
- **void add (String name)**

Тут *name* - Ім'я додається елемента.
Для визначення обраного в даний час елемента.

- **String getSelectedItem()**
- **int getSelectedIndex()**

Метод getSelectedItem () повертає рядок, що містить ім'я елемента, а метод getSelectedindex () - Індекс (номер) елемента.

Елемент управління *Choice*

Щоб отримати кількість елементів у списку

- **int getItemCount ()**
- **void select (int index)**
- **void select (String name)**

Знаючи індекс, можна отримати ім'я елемента з цим індексом.

- **String getItem (int *index*)**

де index специфікує індекс бажаного елемента.

Використання списків

Клас List забезпечує компактний багатоелементний список з множинним вибором і прокруткою.
На відміну від об'єкта типу Choice, який показує в меню тільки один вибраний елемент, List-Об'єкт може бути сконструйований так, щоб відображати будь-яке число елементів вибору у видимому вікні.

- **List ()**
- **List (int *nuwRows*)**
- **List (int *numRows,* boolean *multipleSelect*)**

Перша форма створює елемент управління List, який дозволяє вибирати тільки один елемент.
У другій формі значення параметра **numRows** визначає число рядків у списку, які будуть завжди видимі в панелі списку (інші можуть прокручуватися в панелі по мірі необхідності).
У третій формі, якщо параметр ***muitipieSelect*** дорівнює true, то користувач може вибирати два або кілька елементів одночасно. Якщо його значення - false, то можна вибрати тільки один елемент.

Використання списків

Щоб додати елемент вибору до списку, викликайте метод add (), Який має такі форми:

- **void add (String *name*)**
- **void add (String *name,* int *index*)**

Тут ***name*** - Ім'я елемента, який додається до списку.

- Перша форма додає елементи до кінця списку.
- Друга - додає елементи за індексом (номером), вказується параметр ***index.*** Індексація починається з нуля. Щоб додати елемент в кінець списку, потрібно вказати індекс, рівний -1.

Управління смугами прокрутки

Смуги прокрутки (Scroll bars) використовуються для вибору безперервних значень з деякого інтервалу з кінцевими межами.
Смуги прокрутки можуть бути орієнтовані горизонтально або вертикально.
Поточне значення смуги прокручування щодо її мінімальних і максимальних значень позначено *повзунком '* (Або *бігунком)* смуги прокрутки.
Смуги прокрутки інкапсульовані в класі Scrollbar. BScrollbar визначено такі конструктори:

- **Scrollbar ()**
- **Scrollbar (int *style*)**
- **Scrollbar (int *style,* int *initialValue,* int *thumbSize,* int *min,* int *max*)**

- Перша форма створює вертикальну смугу прокрутки.
- Друга і третя - дозволяють вказати орієнтацію смуги прокручування.

Використання класу *TextField*

Клас TextField реалізує однорядковий область введення тексту, зазвичай звану *елементами редагування* (Edit control). Текстові поля дають можливість користувачеві вводити рядки і редагувати текст, використовуючи клавіші-стрілки, клавіші для операцій "вирізати" і "вставити", а також вибірки мишею.

- **TextField()**
- **TextField (int numChars)**
- **TextField (String str)**
- **TextField (String *str,* int numChars**)

- Перша форма створює заданий текстове поле за замовчуванням.
- Друга - створює текстове поле шириною ***numchars*** символів.
- Третя форма ініціалізує текстове поле з рядком, що міститься в **str.**
- Четверта - ініціалізує текстове поле і встановлює його ширину.

Використання *TextArea*

Іноді однорядковий текстовий введення не достатній для даного завдання. Щоб обробляти ці ситуації, AWT включає простий багаторядковий редактор:

- **TextArea ()**
- **TextArea (int *numLines,* int *numChars*)**
- **TextArea (String *str*)**
- **TextArea (String *str,* int *nvmLines,* int numChars)**
- **TextArea (String str, int *nvmLines,* int *numChars,* int *sBars*)**

Тут numlines визначає висоту текстової області (в рядках); *numChars* -її ширину (в символах); str- початковий текст. У п'ятій формі можна визначити смуги прокрутки, якщо ви хочете, щоб елемент керування їх мав. SBars повинен приймати одне з наступних значень:

- SCROLLBARS_BOTH
- SCROLLBARS_HORIZONTAL_ONLY
- SCROLLBARS_NONE
- SCROLLBARS_VERTICAL_ONLY

Поняття менеджера компоновки

Всі компоненти, які ми показували до сих пір, були розміщені (позиціоновані) в контейнері менеджером компоновки, заданим за умовчанням.
Менеджер компоновки автоматично розміщує елементи управління в межах вікна, використовуючи деякий тип алгоритму.
Хоча елементи управління Java теж можна розміщувати вручну, в цьому немає необхідності з двох причин. По-перше, дуже важко вручну розміщувати велику кількість компонентів. По-друге, іноді, в той момент, коли потрібно розміщувати певний елемент, інформація про його розміри виявляється недоступною (частіше за все тому, що необхідні для отримання цієї інформації компоненти комплекту інструментів ще не були реалізовані).
Кожен об'єкт типу Container має пов'язаний з ним менеджер компоновки.
Менеджер компонування - це екземпляр деякого класу, який реалізує інтерфейс LayoutManager. Менеджер компоновки встановлюється методом setLayout ().

- **void setLayout(LayoutManager *layoutobj)***

Тут *layoutobj* -посилання на необхідний менеджер компоновки. Якщо ви хочете відключити менеджер компоновки і позиціонувати компоненти вручну, передавайте в якості параметра *layoutobj* значення null (Порожній покажчик).
Кожен менеджер компоновки зберігає і відслідковує список імен компонентів.

Менеджер *FlowLayout*

FiowLayout - Це менеджер *потокової* компонування.
Якщо метод setLayout() не встановлює ніякої іншої компонувальник, то даний компонувальник використовується за умовчанням. Цей менеджер використовували всі попередні приклади.
FiowLayout реалізує простий стиль розміщення, схожий на потік слів в текстовому редакторі. Компоненти розміщуються від лівого верхнього кута вікна, зліва направо і зверху вниз. Коли немає більше компонентів, придатних для розміщення на рядку, черговий компонент розміщується на початку наступного рядка.

- **FlowLayout ()**
- **FlowLayout (int *how*)**
- **FlowLayout (int *how,* int *horz,* int vert)**

Перша форма створює розміщення за замовчуванням, яке вирівнює компоненти по центру і залишає п'ять пікселів пропусків між кожним компонентом. Друга форма дозволяє визначити, як вирівнюється кожен рядок. Допустимі наступні значення параметра *how.*

- FlowLayout.LEFT
- FlowLayout.CENTER
- FlowLayout.RIGHT

Третя форма дозволяє задавати (в параметрах *horz* і*vert)*горизонтальний і вертикальний пробіл, що залишається між компонентами (у формі цілого числа пікселів).

Клас *BorderLayout*

Клас BorderLayout реалізує *граничний* стиль компонування, використовуваний для вікон верхнього рівня. Він має чотири вузьких компонента фіксованої ширини по краях і один - у вигляді великої області - в центрі. Чотири крайових компонента називають Північ (North), Південь (South), Схід (East) І Захід (West). Середня область називається Центр (Center).

- **BorderLayout()**
- **BorderLayout (int *horz,* int *vert*)**

Перша форма створює граничну розміщення, що використовується за умовчанням. Друга - дозволяє вказувати кількість (в параметрах *horz* ivert, Відповідно) горизонтальних і вертикальних прогалин, що залишаються між компонентами. BorderLayout визначає наступні константи:

- BorderLayout.CENTER
- BorderLayout.EAST
- BorderLayout.NORTH
- BorderLayout.SOUTH
- BorderLayout.WEST

Використання вставок

Іноді потрібно залишити трохи порожнього місця між контейнером, який зберігає компоненти, і вікном, що містить контейнер. Для цього необхідно перевизначити метод getInsets (), який визначений в класі Container. Ця функція повертає об'єкт типу insets, Що містить верхню, нижню, ліву і праву вставки, які використовуються під час відображення контейнера. Менеджер компоновки використовує ці значення при вставці компонентів, коли розміщує вікно навколо контейнера.

- **Insets (int *top,* int *left,* int *bottom,* int *right*)**

Значення, що пересилаються параметрами *top,left,bottom,righ* визначають кількість пробельного простору (в пікселах) між контейнером і включає його вікном.

Менеджер *GridLayout*

Менеджер GridLayout розпорядженні компоненти в двовимірної сітці. Число рядків і стовпців сітки слід визначати при створенні екземпляра (об'єкта) GridLayout.

- **GridLayout ()**
- **GridLayout(int numRows,int numColumns)**
- **GridLayout (int numRovs, int numColumns, int *horz,* int *vert*)**

Перша форма створює сіткове розміщення з одиночним стовпцем. Друга - сіткове розміщення з зазначеним числом рядків і стовпців. Третя форма дозволяє визначати горизонтальний і вертикальний прогалини, що залишаються між компонентами (в параметрах ***horz*** і***vert,*** відповідно).
Параметр **numRows** або **numColumns** може бути нульовим. Нульова специфікація **numRows** допускає стовпці з необмеженою довжиною. Нульова специфікація **numColumns** допускає рядки з необмеженою довжиною.

Клас *CardLayout*

Клас CardLayout унікальний серед інших менеджерів компонування тим, що зберігає кілька різних розміщень.
Кожне розміщення можна предствлять окремої пронумерованою картою в колоді, яка може бути перетасувала так, щоб в даний момент нагорі перебувала будь карта.
Це може бути корисно для інтерфейсів користувача з необов'язковими компонентами, які можна динамічно включати і вимикати при введенні користувача.
Ви можете підготувати інші компонування і зробити їх прихованими, але готовими до активізації, коли це необхідно.

- **CardLayout()**
- **CardLayout (int *horz,* int *vert*)**

- Перша форма створює карткову компоновку за замовчуванням.
- Друга - дозволяє вказувати горизонтальний і вертикальний пробіл, що залишається між компонентами (в параметрах horz ivert, Відповідно).

Панелі меню і меню

Вікно верхнього рівня може містити пов'язану з ним панель меню (menu bar) Верхнього рівня.
У рядку цього меню відображається список обираних елементів (виборів, choices).
Кожен обираний елемент панелі меню верхнього рівня пов'язаний з випадним (drop-down) Меню, що містить свої обрані елементи. Ці елементи меню реалізовані в Java наступними класами: MenuBar,Menu iMenuitem.
Щоб створити панель меню, спочатку створюють примірник типу MenuBar.

- **Menu ()**
- **Menu (String *optionName*)**
- **Menu (String *optionName,* boolean *removable*)**

Тут optionName специфікує ім'я обраного елемента меню-панелі.
Якщо параметр removable - true, Що випадає меню може бути видалено (з ієрархії) і відправлено "у вільне плавання" (по вікну). В іншому випадку це меню залишиться прикріпленим до панелі меню. (Відкріпляли меню залежать від реалізації.) Перша форма створює пусте меню.
Індивідуальні пункти меню мають тип Menuitem.

- **Menuitem ()**
- **Menuitem (String itemName)**
- **Menuitem (String itemName, ManuShortcut *keyAccel*)**

де itemName - Ім'я, показане в меню; *keyAccel* -клавіші швидкого доступу для цього пункту.

Діалогові вікна

Часто необхідно використовувати *діалогове вікно,* містить набір пов'язаних елементів управління.
Вони подібні фрейм-вікнам, за винятком того, що діалогові вікна - завжди дочірні вікна для вікна верхнього рівня.
Крім того, діалогові вікна не мають рядки меню.
В інших відносинах вони функціонують подібно фреймових вікон.
Діалогові вікна можуть бути модальними або немодальному.
Коли *модальне* діалогове вікно активно, весь ввід прямує до нього, поки воно не буде закрито. Це означає, що ви не можете звертатися до інших частин програми до тих пір, поки не закрили діалогове вікно.
Коли *немодальному* діалогове вікно активно, фокус введення може бути направлений іншого вікна вашої програми. Таким чином, інші частини вашої програми залишаються активними і доступними.

- **Dialog (Frame *parentWindow,* boolean *mode*)**
- **Dialog (Frame parent*Window,* String *title,* boolean *mode*)**

Тут *parentWindow*- Власник діалогового вікна. Якщо *mode* має значення true, Діалогове вікно є модальним. Інакше, воно - немодальному. Тема діалогового вікна можна передати через параметр title. В загальному випадку, ваша програма буде підкласом класу Dialog, Що додає функціональні можливості, необхідні вашому додатком.

Клас *FileDialog*

Java забезпечує вбудоване діалогове вікно, яке дає можливість користувачу специфікувати файл.
Для відкриття цього вікна програмі досить створити конкретний екземпляр об'єкта типу FileDialog.

- **FileDialog (Frame *parent,* String *boxName*)**
- **FileDialog (Frame parent. String *boxName,* int *how*)**
- **FileDialog (Frame *parent*)**

Тут *parent -* власник діалогового вікна;*boxName* -ім'я,відображуване в області заголовка вікна.Якщо *boxName* опущений, заголовок діалогового вікна залишається порожнім. Якщо *how* має значення FileDialog.load,to вікно вибирає файл для читання, а якщо *how* має значення FileDialog.save, Вікно вибирає файл для запису (з метою збереження).
Третій конструктор створює діалогове вікно з вибором файлу для читання.

Робота з зображеннями

Розглядається AWT-Клас image і пакет java.awt.image. Разом вони підтримують роботу із зображеннями (відображення і маніпуляції з графічними зображеннями).
Під *зображенням* розуміють прямокутний *графічний об'єкт.*
Зображення є ключовим компонентом Web-Дизайну.
Зображення - це об'єкти класу image, Який є частиною пакета java.awt.
Для маніпулювання зображеннями використовуються класи пакета java.awt.image, Який містить велику кількість класів і інтерфейсів зображень.

Формати графічних файлів

Спочатку, Web-Зображення могли бути тільки в форматі GIF. Формат растрових зображень GIF (Graphics Interchange Format, Формат обміну графічними даними) був створений в CompuServe Incorporation в 1987 р., для можливості перегляду вбудованих зображень, що добре підходило для Internet. Кожне GIF-Зображення може мати не більше 256 квітів.
Це обмеження змусило головних постачальників браузерів в 1995 р. додати підтримку зображень у форматі JPEG. Формат JPEG (Joint Photographic Expert Group) Був створений групою фотографічних експертів для зберігання зображень з повним колірним спектром і безперервним тоном. Ці зображення, якщо вони створені належним чином, можуть мати набагато більш високу точність кольоровідтворення і більш високу ступінь стиснення у порівнянні з GIF-Кодуванням.
У більшості випадків вас не буде навіть цікавити, який формат ви використовуєте в своїх програмах.

Створення об'єкта зображення

Щоб зображення стали видимими, їх потрібно *малювати* у вікні. Проте клас image не має достатньої інформації для того, щоб створити належний формат даних для екрану.
Тому клас component (З пакету java.awt) Містить спеціальний "виробничий" (factory) Метод з ім'ям createImage (), Який використовується для створення image-Об'єктів.

- **Image createImage (ImageProducer *imgProd*)**
- **Image createImage (int *width,*   int *height*)**

Перша форма повертає зображення, виготовлене параметром **imgProd,** який є об'єктом класу, що реалізовує інтерфейс ImageProducer.
Друга форма повертає порожнє зображення, яке має зазначену ширину і висоту.

Завантаження зображення

Інший спосіб отримання зображення - його завантаження. Для цього використовуйте метод getImage (), Визначений класом Applet.

- **Image gatImage (URL url)**
- **Image getImage (UKL *url,* String imageName)**

Перша версія повертає image-Об'єкт, який інкапсулює зображення, знайдене по (універсального) адресою, вказаною в параметрі **url.** Друга версія повертає image-Об'єкт, який інкапсулює зображення, знайдене за адресою, вказаною в **url,**і має ім'я, вказане в **ImageName.**

Перегляд зображення

Маючи зображення, ви можете виводити його (на екран), використовуючи метод drawImage (), Який є членом класу Graphics.

- **boolean drawImage (Image imgObj, int left, int *top,* ImageObserver *imgOb*)**

Він виводить зображення, передане йому параметром ***imgObj****,* розміщуючи його лівий верхній кут з позиції, зазначеної в ***left*** і ***top.imgcb*** *-* посилання на клас, який реалізує інтерфейс ImageObserver.

Цей інтерфейс реалізується усіма AWT-Компонентами.

Спостерігач зображення (image observer) - Це об'єкт, який може контролювати зображення, поки воно завантажується.

Інтерфейс ImageObserver

Imageobserver - Це інтерфейс, який використовується для прийому повідомлень про те, як генеруються зображення.
Використання спостерігача зображення дозволяє виконувати (паралельно із завантаженням зображення) інші дії, такі як показ індикатора ходу роботи (Progress-Індикатора) або додаткового екрану, які інформують вас про хід завантаження. Подібний вид повідомлення дуже корисний, коли зображення завантажується по мережі, де проектувальник вмісту рідко бере до уваги, що люди часто пробують завантажувати аплети через повільний модем.

- **boolean ImageUpdate (Image *imgObj,* int *flags,* int *left,* int *top,* int *width,* int *height*)**

Тут imgObj - Завантажувати зображення, a*flags* -ціле число, яке повідомляє стан звіту оновлення. Чотири цілих параметра *left, top, width* i*height* представляють прямокутник, який містить різні значення в залежності від переданих в flags-Значень, ImageUpdate () повинен повернути false, якщо він завершив завантаження, і true, якщо ще є залишок зображення для обробки.

Подвійна буферизація

Мало того, що зображення корисні для зберігання картинок, але ви можете також використовувати їх для малювання поверхонь поза екраном.
Це дозволяє вам передавати будь-яке зображення, включаючи текст і графіку, у позаекранного буфер, який ви можете відображати в більш пізній час.
Перевага даного механізму полягає в тому, що зображення стає видимим тільки тоді, коли воно вже остаточно побудовано.
Малювання складного зображення може займати кілька мілісекунд або більше, що може спостерігатися користувачем як миготіння або мерехтіння. Це миготіння відволікає користувача і змушує його відчувати вашу візуалізацію як більш повільну, ніж вона є насправді.
Використання позаекранного зображення для ослаблення мерехтіння називається *подвійний буферизацією,* через те що екран розглядається як буфер для пікселів, а позаекранного зображення - це другий буфер, де ви можете готувати пікселі для показу на екрані.

Клас *MediaTracker*

MediaTracker створює об'єкт, який буде паралельно перевіряти стан довільного числа зображень.
Для використання MediaTracker ви створюєте його новий екземпляр і застосовуєте його метод addImage (), Щоб простежувати стан завантаження зображення. Загальний формат addImage ():

- **void addImage (Image imgObj, int *imgID*)**
- **void addImage (Image *imgObj,* int *imgID,* int *width,* int *height*)**

Тут *imgObj* - Простежується зображення. Його ідентифікаційний номер передається в imgID. ID (Ідентифікатор) номера не повинні бути унікальними. Ви можете використовувати той же номер з кількома зображеннями, як засіб їх ідентифікації в якості частини групи.
У другій формі параметри *width* і*height* визначають розміри відображуваного об'єкта.

Інтерфейс *ImageProducer*

ImageProducer - Це інтерфейс для об'єктів, які хочуть виробляти дані для зображень.
Об'єкт, який реалізує інтерфейс ImageProducer, поставляє цілочисельні або байтові масиви, які представляють дані зображення і виробляють image-Об'єкти.
Як ви бачили раніше, одна з форм методу createImage () має об'єкт imageProducer в якості свого параметра.
Існують два *виробника зображень* (Image piodusers), містяться В java.awt. image: MemoryImageSource iFilteredImageSource.

Виробник зображень *MemoryImageSource*

MemoryImageSource - Це клас, який створює новий image-Об'єкт з масиву даних.

- **MemoryImageSource (int *width,* int height, int *pixel [ ],* int *offset,* int *scanLineWidth*)**

Об'єкт MemoryImageSource створюється з масиву цілих чисел (у форматі замовчуваний колірної моделі RGB), вказаного в параметрі *pixel*. В замовчуваний колірної моделі піксель - це ціле число формату 0xAARRGGBB, де A - Alpha, R -Red, G -Green, і B - Blue.
Значення Alpha представляє ступінь прозорості пікселя (0 - повністю прозорий, 255 - повністю непрозорий). Ширина і висота результуючого зображення передається в параметрах *width* і*height.* Вихідну точку для початку читання даних в масиві пікселів задає параметр *offset.* Ширина рядка сканування (яка часто збігається із шириною зображення) задає параметр *scanLinewidth.*

Інтерфейс *ImageConsumer*

ImageConsumer - Це абстрактний інтерфейс для об'єктів, які хочуть отримувати пикселной дані зображень (скажімо, від *виробника)* і постачати їх (скажімо, на екран) вже як інший вид даних.
Цей інтерфейс є протилежністю інтерфейсу ImageProducer.
Об'єкт, який реалізує інтерфейс ImageConsumer, збирається створювати масиви int або byte, які представляють пікселі image-Об'єкта.

Клас *PixelGrabber*

Клас PixeiGrabber визначено в java.iang.image. Це інверсія класу MemoryImageSource.
Замість побудови зображення з масиву пикселной значень, він бере існуюче зображення і будує з нього масив пікселів.
Для використання PixeiGrabber ви спочатку організуєте int-Масив достатньо великий, щоб утримувати дані пікселів, і потім створюєте екземпляр pixeiGrabber, передаючи йому прямокутну область, яку ви хочете перетворити в пикселной подання. Нарешті, ви викликаєте метод grabPixels () Цього примірника.

- **PixelGrabber (Image *imgObj,* int *left,* int *top,* int *width,* int *height,* int *pixel*[], int *offset*, Int *scanLineWidth*)**

Тут *imgObj* -об'єкт, чиї пікселі перетворюються. Значення *left* і*top* визначають лівий верхній кут прямокутника, а*width* і*height* - Його розміри, з якого будуть отримані пікселі. Пікселі будуть збережені в масиві *pixel,* зі зміщенням *offset.* Ширину рядка сканування (яка часто така ж, як ширина зображення) задають в *scanLineWidth.*

Клас *ImageFilter*

- На базі Інтерфейсів Image Produce г I ImageConsumer можна створити довільний набір перетворюють фільтрів, кожний з яких бере джерело пікселів, модифікує згенеровані ними пікселі і передає безпідставного споживачеві.
- Цей механізм аналогічний способу, за допомогою якого створюються конкретні абстрактні класи введення / виводу InputStream, OutputStream, Reader IWriter.
- Така *потокова модель зображень* завершується введенням класу ImageFilter.
- Клас ImageFilter пакета java.awt.image має декілька підкласів - AreaAveragingScaleFilter, CropImageFilter, ReplicateScaleFilter IRGBImageFilter.

Фільтр *CropImageFilter*

Фільтр CropImageFilter просто вирізає з вихідного зображення невелику прямокутну область.
Його зручно використовувати, наприклад, коли потрібно працювати не з цілим великим зображенням, а з більш дрібними його частинами.
Якщо кожне подізображеніе має один і той же розмір, то можна легко витягувати їх, використовуючи для розбирання блоку зображення фільтр CropImageFilter.

Фільтр *RGBImageFilter*

Фільтр RGBImageFilter використовується для піксельну перетворення одного зображення в інше, трансформуючи по шляху колір пікселів. Даний фільтр можна використовувати для прояснення зображення, збільшення його контрасту, або навіть для перетворення кольорового зображення до півтонове.

Дякую за увагу!