Елементи об'єктно-орієнтованого програмування.

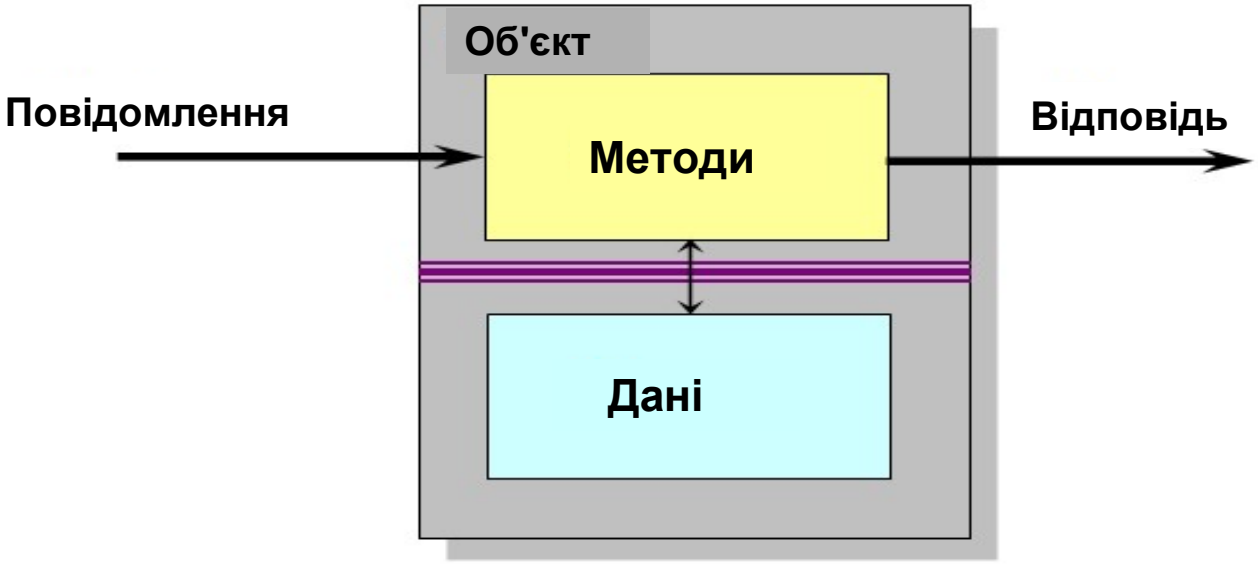
Об'єктно-орієнтований підхід в Java

Об'єкти програмування мають дві характеристики: стан і поведінка: стан об'єкта визначається його властивостями, а поведінка - виконуваними їм діями.

Для об'єктно-орієнтованих мов характерні наступні підходи до програмування:

- все є об'єктом;
- програма є набором об'єктів, що вказують один одному, що робити за допомогою посилки повідомлень;
- кожен об'єкт має певний тип;
- всі об'єкти одного типу можуть отримувати одні і ті ж повідомлення.

Об'єкт - це набір даних і процедур, які працюють з цими даними. Ці процедури (підпрограми) обробки даних об'єкта називаються в Java **методами**.



Методи об'єкта мають повний доступ до даних свого об'єкта. Поза об'єкта дані, які оголошуються в даному об'єкті, не видно, а вся взаємодія з об'єктом здійснюється тільки через методи.

Об'єкт може бути розділений на дві компоненти: зовнішню і внутрішню. Зовнішню частину (**інтерфейс**) складають методи, які здійснюють взаємодію з іншою частиною програми. Внутрішню частину складають дані і методи, доступні тільки всередині об'єкта.

Інкапсуляція - це процес упаковки даних об'єкта разом з його методами. Результатом інкапсуляції є запобігання несанкціонованого доступу ззовні до даних і методів всередині об'єкта і можливість зміни внутрішньої реалізації об'єкта без зміни інших частин програми.

Об'єкти програми взаємодіють і зв'язуються один з одним за допомогою **повідомлень**.

Інформація, необхідна об'єкту для виконання тих чи інших методів даного об'єкта передається в повідомленні за допомогою **параметрів**.

Таким чином, **повідомлення повинно містити три компоненти**:

- об'єкта - одержувача повідомлення;
- імені виконуваної дії;
- параметрів, необхідних для виконання дії.

Інтерфейси дозволяють об'єктам передавати і приймати повідомлення, навіть якщо вони розташовані в різних вузлах мережі

Щоб визначити об'єкт, в Java і інших об'єктно-орієнтованих мовах використовується поняття: **клас**.

Клас - це шаблон або прототип, який визначає тип об'єкта. Клас містить опис змінних і констант, що характеризують властивості об'єкта. Вони називаються **полями класу** (**class fields**). Процедури, які описують поведінку об'єкта, називаються **методами класу** (**class methods**).

Коли об'єкт створюється з класу, змінні, оголошені для цього класу, розміщуються в пам'яті. Потім змінні можуть модифікуватися за допомогою методів об'єкта. Реалізації одного класу спільно використовують реалізації методів класу, але кожна з них використовує свої власні дані об'єкта, тобто клас можна **повторно використовувати** для реалізації декількох об'єктів з однаковими методами, але різними значеннями даних.

Оголошення класу в Java:

```
class ідентифікатор-класу
{
        тіло-класу
}
```

ідентифікатор визначає ім'я класу.

У тілі класу (і тільки в тілі класу) визначаються його змінні і методи.

Програма на мові Java є оголошення одного або декількох класів. Кожен клас в програмі компілюється в окремий файл з ім'ям **ідентифікатор-класу.class**.

```
class Circle
{
…
}
```

Приклади створення об'єктів класу Circle

```
Circle obj1;
```

- створення змінної obj1 типу **Circle**

```
obj1 = new Circle();
```

- створюється новий об'єкт типу Circle, посилання на нього (адреса об'єкта) записується в змінну **obj1**

```
Circle obj1 = new Circle();
```

- поєднане завдання об'єктної змінної і призначення їй об'єкта

```
Circle obj1 = new Circle(130,120,50);
```

- створення змінної зі списком параметрів

```
Circle circle1 = new Circle(130,120,50);
Circle circle2 = new Circle(130,120,50);
```

- створення двох незалежних об'єктів одного класу з однаковими початковими параметрами

```
obj1.x = 5;
```

- звернення до полю даних **x** об'єктної змінної **obj1**

```
obj1.show();
```

- звернення методу **show()** об'єктної змінної **obj1**

Методи в Java

Визначення методу:

```
повертаний-тип ідентифікатор-методу (параметри)
{
тіло-методу
}
```

Повертаний-тип визначає тип даних, які повертає метод при виклику.

Ідентифікатор-методу визначає ім'я методу, а параметри - список параметрів, які необхідно передати методу при його виклику.

Тіло-методу містить оператори, що реалізують дії, які виконуються даним методом.

Якщо тип значення, що повертається, не void, в тілі методу повинен бути хоча б один оператор

```
return вираз;
```

Приклад

Визначення методу:

```
int sumOfTwoValues(int a, int b)
{
        int x;
        x = a + b;
        return x;
}
```

Виклик методу:

```
int y;
y=sumOfTwoValues(5, 3); // y = 8
```

У мові Java в межах одного класу можна визначити два або більше методів, які спільно використовують одне і те ж ім'я, але мають різну кількість параметрів. Такі методи називають **перевантаженими**, а про процес кажуть як про **перевантаження методу** (method overloading).

У Java можна використовувати методи, реалізація яких, виконана в зовнішньому файлі, в програмі написаному на мові C / C ++. Для цього перед визначенням методу (але без тіла методу) вказується модифікатор **native**

Змінні типу класів в Java

```
MyClass obj1;
MyClass obj2;
```

- дві змінних класу **MyClass** з іменами **obj1** і **obj2**

Для оголошеної змінної типу класу необхідно створити **об'єкт**, **екземпляр** (**instance**) описаного класу.

Коли створюється об'єкт, необхідно ініціалізувати його змінні. Для цього в класі визначається спеціальний метод (перевантажені методи), ім'я якого збігається з ім'ям класу. Ці методи називають **конструкторами.**

Конструктор відрізняється від звичайних методів наступними основними особливостями:

- конструктор не повинен повертати ніякого значення;
- тип значення для конструктора не вказується

Якщо конструктор в класі не визначений, компілятор Java створює конструктор за замовчуванням.

Створення об'єкта

```
ідентифікатор-змінної = new ідентифікатор-конструктора (параметри);
```

ідентифікатор-змінної - ім'я створюваного об'єкта

ідентифікатор-конструктора - ім'я конструктора класу, що викликається

параметри - список параметрів, переданих конструктору класу

```
obj1 = new MyClass();
obj2 = new MyClass(12, 5);
```

Операції оголошення і ініціалізації змінних типу класу можуть бути об'єднані в одному операторі

```
MyClass obj3 = new MyClass(8, 4);
```

Змінні і методи об'єкта

Звернення до змінних і виклик методів об'єкта

```
ім'я-об'єкта.імя-змінної
і
ім'я-об'єкта.імя-методу(аргументи)
```

ім'я-об'єкта - це ідентифікатор змінної об'єкта класу

ім'я-змінної - ідентифікатор змінної класу

ім'я-методу - ідентифікатор методу класу

аргументи - значення, що задаються при виклику методу

```
int var1InObj1 = obj1.var1;
```

```
obj2.var1 = 12;
```

```
obj1.setVar1(2);
obj2.setVar1(2);
```

Якщо ім'я об'єкта для змінної або методу не задано, то компілятор Java вважає, що дана змінна або метод визначені в даному класі. Можна в цьому випадку замість імені об'єкта вказати ключове слово **this**.

Зазвичай таку вказівку використовується в тих випадках, якщо **ім'я змінної класу і аргумент методу в класі збігаються**

```
class MyClass
{
        int var1;
        …
        void setVar1(int var1)
        {
                this.var1=var1;
        }
}
```

Змінні і методи класу

Модифікатор **static** задає змінну, загальну для всіх об'єктів даного класу.

Для роботи зі статичними змінними зазвичай створюються **статичні методи**, помічені модифікатором **static**.
Статичні методи і змінні називають також **методами і змінними класу** (class variables and methods), оскільки до них можна звертатися, вказуючи не ім'я об'єкта, а ім'я класу.

Приклад

Визначення

```
static int var2 = 0;
…
static void setVar2(int var)
{
        var2 = var;
}
```

Звернення

```
int val2Value = MyClass.var2;
MyClass.setVar2(12);
```

Основна особливість статичних змінних і методів: доступ до них виконується, навіть якщо не створено жодного примірника класу.

Для статичних методів діють наступні основні обмеження:

- в статичному методі не можна використовувати посилання **this**;
- в статичному методі не можна звертатися до нестатичних змінним (всі змінні, оголошені поза статичного методу і використовуються всередині нього, повинні бути оголошені з модифікатором **static**);
- в статичному методі не можна прямо викликати нестатичні методи (всі методи, які викликаються з статичного методу, повинні бути оголошені з модифікатором **static**).

Операції над об'єктами

присвоювання "=" - привласнення покажчика на об'єкт посилальній змінній (при цьому нової копії об'єкта не створюється);

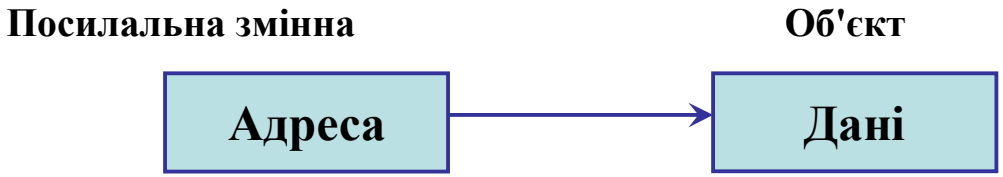
перевірка на рівність "==" і на нерівність "! =" - результатом цих операцій буде **true** або **false**, залежно від того, чи вказують порівнювані змінні на один і той же об'єкт в пам'яті.

Операція *ім'я-об'єкта instanceof ім'я-класу*

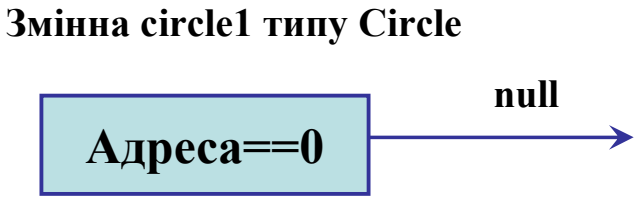
Результатом цієї операції є **true**, якщо об'єкт з ідентифікатором ***ім'я-об'єкта*** є реалізацією класу з ідентифікатором ***ім'я-класу***, і **false** - в протилежному випадку.

```
MyClass obj4 = new MyClass(15, 7);
MyClass obj5 = new MyClass(15, 7);
MyClass obj6 = obj4;
boolean isEqual1, isEqual2, isInstance;
isEqual = obj6 == obj4;                    // isEqual1 - true
isEqual2 = obj4 == obj5;                   // isEqual2 - false
isInstance = obj4 instanceof MyClass;      // isInstance - true
```

Робота зі посилальними змінними

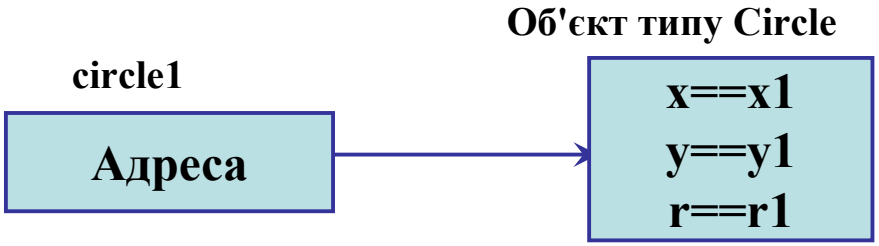


Посилальна змінна і пов'язаний з нею об'єкт



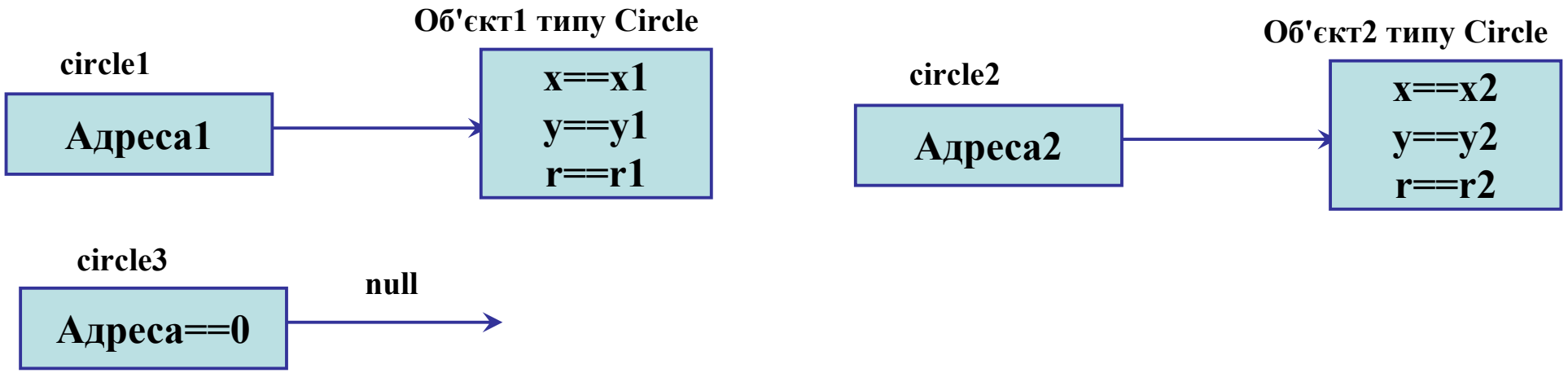
Початковий стан посилальної змінної **circle1**

```
circle1 = new Circle(x1, y1, r1);
```

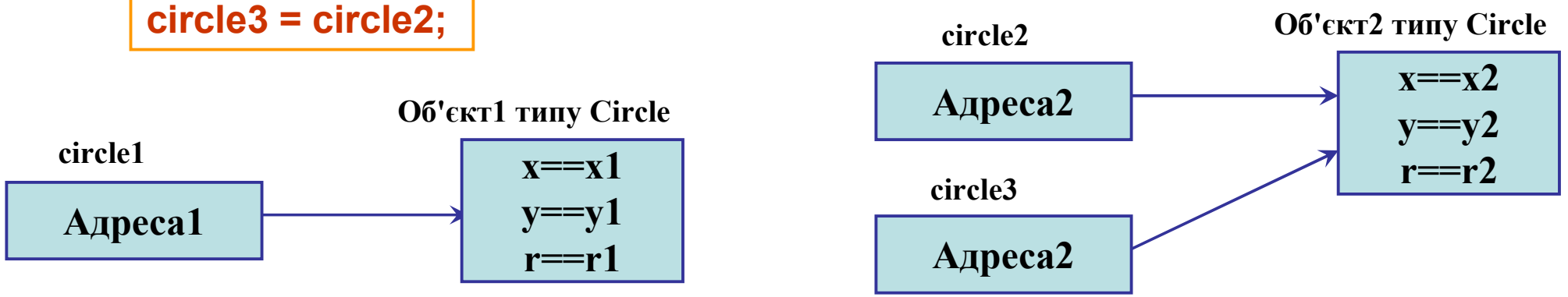


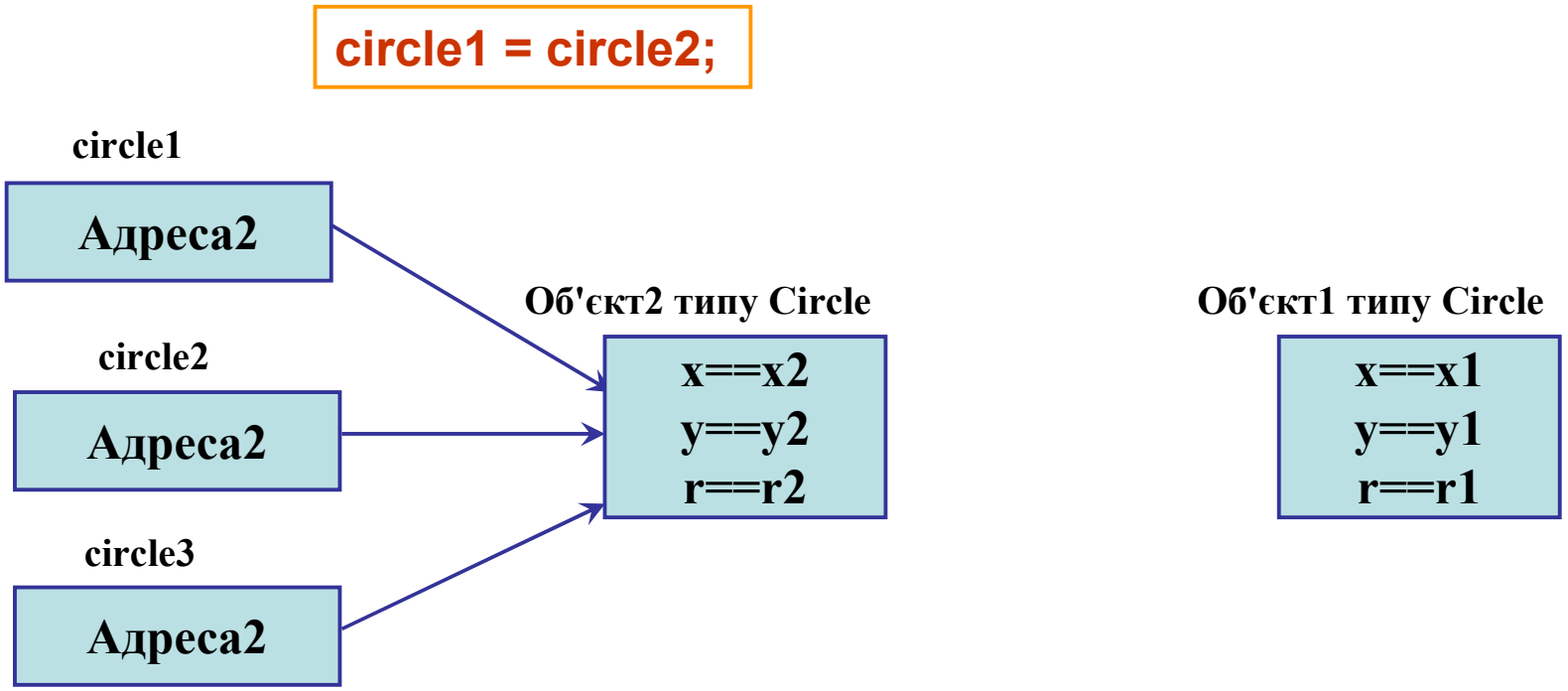
Посилальна змінна **circle1** і пов'язаний з нею об'єкт типу **Circle**

```
Circle circle1 = new Circle(x1, y1, r1);
Circle circle2 = new Circle(x2, y2, r2);
Circle circle3;
```




```
circle3 = circle2;
```




circle1 = circle2;
circle1
Адреса2
circle2
Адреса2
circle3
Адреса2
Об'єкт2 типу Circle
x==x2
y==y2
r==r2
Об'єкт1 типу Circle
x==x1
y==y1
r==r1

Об'єктні надбудови примітивних типів

Багато класів в Java працюють не з примітивними типами даних, а з об'єктами, оскільки для об'єктів можна задавати властивості і методи.

З цією метою в Java введені класи - **об'єктні надбудови над примітивними типами**. Нові об'єкти, наприклад, числа в цих класах задаються за допомогою оператора **new** і над ними не можна виконувати операції, визначені для примітивних типів (наприклад, складання), проте вони часто використовуються для виконання деяких операцій, що реалізуються за допомогою методів відповідного класу-надбудови.

Всі класи-надбудови автоматично доступні програмі, оскільки вони знаходяться в пакеті **java.lang**.

Оболонкові класи.
Упаковка (boxing) і розпакування (unboxing)

Примітивний тип	Оболонковий клас
byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double

```
Integer obj1 = 10;        //упаковка
int i1 = obj1*2;          //розпакування при обчисленні виразу
```

```
Byte b = 1;               //упаковка
obj1 = i1/10;             //упаковка
b = 2;                    //упаковка
```

Пакети

Всі класи Java розподілені по **пакетам** (зазвичай за функціональною ознакою, наприклад, класи-утиліти або класи введення-виведення). Крім класів, пакети можуть включати в себе **інтерфейси** і вкладені **підпакети** (subpackages). В результаті утворюється деревоподібна структура пакетів і підпакетів, яка відповідає структурі файлової системи. Всі файли з розширенням **.class**, що утворюють пакет, зберігаються в одному каталозі, а підпакети зберігаються в підкаталогах цього каталогу.

Пакет однозначно ідентифікується своїм ім'ям, перед яким, відокремлені один від одного крапкою, йдуть імена всіх пакетів, які перебувають вище даного пакета в рівнях ієрархії.

java.awt.event — позначає ім'я підпакету **event**, що знаходиться в підпакету **awt**, який знаходиться в пакеті **java**.

javax.swing.event — позначає ім'я іншого підпакету з тим же ім'ям **event**, але який знаходиться в підпакету **swing** пакета **javax**.

Пакети

```
package ім'я_пакета;
```

щоб помістити клас в пакет, потрібно задекларувати ім'я пакета на початку файлу, в якому оголошено клас

Також необхідно помістити вихідний код класу відповідну папку

```
package pkg1.pkg2.pkg3;
class MyClass1 {
....
}
```

Вкладеним пакетам відповідають складові імена

```
public class MyClass2 {
....
}
```

При декларації класу можна вказувати, що він є загальнодоступним, за допомогою **модифікатора** доступу **public**.

При цьому можливий доступ до даного класу з інших пакетів.
Якщо ж модифікатор **public** відсутній, то доступ до класу дозволений тільки з класів, які перебувають з ним у одному пакеті.

Всі імена класів, інтерфейсів і підпакетів в пакеті повинні бути унікальні.

В Java прийнято назви пакетів писати тільки малими літерами, імена класів починати з великої літери, а імена полів даних і методів починати з малої літери. Якщо ім'я класу, поля даних або методу складається з кількох слів, то кожне нове слово прийнято писати з великої літери.

```
javax.swing.JMenuItem
```

javax и **swing** — пакети,
JMenuItem — ім'я класу.

При компіляції необхідні для виконання програми класи пакетів Java, за винятком пакета **java.lang**, автоматично не включаються. Щоб зробити їх доступними в програмі, можна або вказувати повне ім'я класу, або використовувати оператор **import** з ім'ям пакета і ім'ям використовуваного класу даного пакета:

```
import java.util.Date;
```

Якщо необхідно використовувати кілька класів з пакета, зазвичай замість імені класу ставиться символ "*", що вказує, що даною програмою будуть доступні всі класи і інтерфейси даного пакета.

```
import java.awt.*;
```

- робить доступними програмі все класи з пакета **java.awt**.

Всі оператори import прийнято розташовувати на самому початку програми

Імпортуються лише імена файлів, що знаходяться на рівні зазначеного пакета. Імпорту імен з вкладених в нього пакетів не відбувається.

```
import pkg1.pkg2.pkg3.MyClass2;
import pkg1.pkg2.pkg3.*;
```

MyClass2 імпортовано

```
import pkg1.*;
import pkg1.pkg2.*;
```

MyClass2 не імпортовано

Приклад

Оголосити графічний об'єкт **g**, що має тип **java.awt.Graphics**, можна трьома способами:

1) безпосередньо із зазначенням імені пакета і класу:

```
java.awt.Graphics g;
```

2) з попередніми імпортом класу **Graphics** з пакета **java.awt** і подальшою вказівкою імені цього класу без його специфікації ім'ям пакета:

```
import java.awt.Graphics;
...
Graphics g;
```

3) з попередніми імпортом всіх класів з пакета **java.awt** і подальшою вказівкою імені цього класу без його специфікації ім'ям пакета:

```
import java.awt.*;
...
Graphics g;
```

Звернення до змінних класу і методів класу має йти тільки через ім'я класу

java.lang.Math

$$y = \frac{\sin \pi x}{\pi x}$$

```
y=Math.sin(Math.PI*x)/(Math.PI*x);
```

Статичний імпорт з пакета класів змінних класу і методів класу:

```
import static java.lang.Math.PI;
import static java.lang.Math.sin;
```

або

```
import static java.lang.Math.*;
```

```
y = sin(PI*x)/(PI*x);
```

Базові пакети і класи Java

Вміст пакету java

java.applet Підтримка роботи з апплетами

java.awt Базовий пакет роботи з графічним призначеним для користувача інтерфейсом (**Abstract Window Toolkit** - Абстрактний інструментарій графічного вікна)

java.beans Підтримка компонентної моделі JavaBeans

java.io Підтримка базових засобів вводу / виводу

java.lang Містить базові класи мови Java. Автоматично імпортується в будь-яку програму без вказівки імені пакета

java.lang.reflect Підтримує механізм доступу до класів як до метаоб'єктів, забезпечує динамічне з'ясування програмою, які можливості підтримує клас. Даний механізм називається **reflection** - "відображення"

java.lang.Math Клас, що забезпечує підтримку основних математичних функцій, а також найпростіший засіб генерації псевдовипадкових чисел

java.math Підтримка обчислень з цілими числами довільної довжини, а також числами в форматі з плаваючою точкою довільної точності

java.net Підтримка роботи в Інтернеті, а також з'єднань через сокети (**sockets**)

java.nio Містить класи та пакети для підтримки мережних з'єднань, що розширюють можливості пакета **java.io**. Зокрема, містить класи контейнерів (буферів) для створення списків з даними різних примітивних типів, а також пакети ***channels*** (канали з'єднання, коннекції) і ***charset*** (національний набір символів). Пакет ***charset*** забезпечує підтримку перекодування із символів Unicode в послідовність байтів для передачі через канал зв'язку, а також зворотне перетворення

java.rmi Підтримка викликів віддалених методів

java.security Підтримка спеціальних засобів, що забезпечують безпеку додатку, в тому числі при роботі в комп'ютерних мережах (списки доступу, сертифікати безпеки, шифрування і т.д.)

java.sql Підтримка SQL-запитів до баз даних

java.text Підтримка спеціальних засобів, що забезпечують локалізацію програм, - класи, які підтримують роботу з текстом, датами, текстовим поданням чисел. Крім того, містить засоби залежного від локалізації порівняння рядків

java.util Містить найважливіші класи для роботи зі структурами даних (в тому числі необхідних для роботи з подіями і датами). Зокрема - підтримку роботи з масивами (сортування, пошук), а також розширені засоби генерації псевдовипадкових чисел

java.util.jar Підтримка роботи з jar-архівами (базовим видом архівів в Java)

java.util.zip Підтримка роботи з zip-архівами

Вміст пакету javax

підтримка можливостей, що з'явилися в Java 2

javax.accessibility Забезпечує налаштування спеціальних можливостей подання інформації для людей з поганим зором, слухом і т.п., а також інших випадків, коли потрібен спеціалізований доступ для управління інформаційними об'єктами

javax.activity Допоміжний пакет для роботи з компонентами

javax.crypto Підтримка шифрування-розшифровки даних

javax.imageio Підтримка роботи з зображеннями (введення / виведення)

javax.management Підтримка роботи з керуючими компонентами (MBean - Management Bean)

javax.naming Підтримка роботи з простором імен компонентів

javax.net Підтримка роботи в Інтернеті, а також з'єднань через сокети (sockets). розширення можливостей **java.net**

javax.print Підтримка роботи з друком документів

javax.rmi Підтримка викликів віддалених методів. Розширення можливостей **java.rmi**

javax.security Підтримка спеціальних засобів, що забезпечують безпеку додатку. Розширення можливостей **java.security**

javax.sound Підтримка роботи зі звуковими потоками і файлами

javax.sql Підтримка SQL-запитів до баз даних. Розширення можливостей **java.sql**

javax.swing Бібліотека основних графічних компонентів в Java2

javax.transaction Підтримка роботи з транзакціями

javax.xml Підтримка роботи з XML-документами і парсерами

Вміст пакету org

пакети, що надаються вільною спільнотою розробників

org.ietf Підтримка захищених з'єднань за протоколом GSS (Kerberos v5)

org.omg Засоби для використання з програм на Java технології CORBA, яка застосовується для створення розподілених об'єктних додатків

org.w3c Інтерфейси для роботи з XML-документами відповідно до специфікації DOM

org.xml Підтримка роботи з XML-документами

Вміст пакету com.sun

забезпечує розширення можливостей пакету javax

com.sun.accessibility Доповнення до пакету **javax.accessibility**

com.sun.beans Доповнення до пакету **java.beans**

com.sun.corba Підтримка роботи в комп'ютерних мережах з базами даних за технологією CORBA (Common Object Request Broker Architecture)

com.sun.crypto Доповнення до пакету **javax.crypto**

com.sun.image Підтримка роботи з зображеннями

com.sun.imageio Доповнення до пакету **javax.imageio**

com.sun.java Підтримка стилів показу додатків, а також службові утиліти для роботи з браузерами і WWW-документами

com.sun.java_cup Підтримка технології JavaCup

com.sun.jlex Підтримка роботи лексичного аналізатора

com.sun.jmx Доповнення до пакету **javax.management**

com.sun.management Доповнення до пакету **javax.management**

com.sun.media Підтримка роботи зі звуком

com.sun.naming Доповнення до пакету **javax.naming**

com.sun.net Доповнення до пакету **javax.net**

com.sun.org Підтримка взаємодії з сервером Apache, засоби роботи з базами даних за технологією CORBA

com.sun.rmi Доповнення до пакету **javax.rmi**

Лектор:

Старший викладач кафедри Електроніки и комп'ютерної техніки Сумського державного університету

Горячев О. Є.

В лекції використано матеріали авторів:

Шонін В.А.
Монахов В.В.