



ІНКАПСУЛЯЦІЯ ТА ВЗАЄМОДІЯ КЛАСІВ У JAVA-ДОДАТКАХ

Лекція 02
Java-програмування



План лекції

- Об'єктна модель Java.
- Породжені типи даних
- Інкапсуляція та приховування інформації в Java.
- Огляд відношень між класами. Вкладені типи.
- Наслідування. Ієрархія та обробка виключень



ОБ'ЄКТНА МОДЕЛЬ JAVA

Питання 2.1.

Клас

- Клас виступає контейнером для застосунку та шаблоном для створення об'єктів.
- Для оголошення класу необхідно використати зарезервоване слово `class`, після якого йде назва класу та тіло класу, яке знаходиться у фігурних дужках.

```
class Image
{
    // various member declarations
}
```

- За домовленістю ім'я класу має починатись з великої літери,
 - записуватись згідно зі стилем «верблюду» - кожне слово з назви починається з великої букви.

Публічні класи

- Називати відповідний файл з первинним кодом (*.java) можна по-різному, проте якщо перед оголошенням класу використовується зарезервоване слово `public`, імена класу та відповідного файлу повинні співпадати.
 - Це пояснюється можливістю доступу до публічних класів з інших пакетів.

```
public class SavingsAccount
{
    // various member declarations
}
```

- Такий клас обов'язково зберігати у файл з назвою `SavingAccount.java`.

Мультикласові файли

```
class A
{
    // various member declarations
}

public class B
{
    // various member declarations
}

class C
{
    // various member declarations
}
```

- В одному файлі можна оголошувати декілька класів.
 - Повинен бути лише один публічний клас.
 - Відповідно до його імені треба називати і файл .java.
- Коли застосунок має мультикласову структуру, кожен з класів може оголошувати власний метод `main()`.
 - Проте це призводить до плутанини, для початківців так робити небажано.
 - Такий метод додають з метою тестування конкретних класів.

```
class A
{
    public static void main(String[] args)
    {
        // statements to execute
    }
}

class B
{
    public static void main(String[] args)
    {
        // statements to execute
    }
}
```

Конструювання об'єктів

- Клас = користувацький тип
- Для того, щоб мати об'єкт, потрібно створити екземпляр класу за допомогою оператора `new` та спеціального методу класу – конструктора.
 - `new constructor`
- Оператор `new` виділяє пам'ять для зберігання об'єкта, тип якого задається **конструктором**.
 - Об'єкти зберігаються в кучі (heap), а ініціалізація відбувається за допомогою виклику конструктора.
 - Після закінчення роботи конструктора оператор `new` повертає посилання (*reference*), по якому можна отримати доступ до об'єкта з будь-якого місця застосунку.

```
class_name(parameter_list)
{
    // statements to execute
}
```

На відміну від інших методів, конструктор не може повертати тип.

- Оператор `new` не може повертати і посилання, і дані від конструктора.
- Конструктор не має власного імені, а використовує назву класу, при цьому також може приймати набір параметрів (аргументів), розділених комами.
- Кількість переданих аргументів називається *арністю* конструктора, методу або оператора.

Конструктори

- `Image image = new Image();`
 - Список аргументів порожній, конструктор називають **безаргументним** (*nonargument*).
 - Посилання на новий об'єкт зберігається у змінній `image` з типом `Image`.
 - Часто під об'єктом мають на увазі таку змінну, хоча насправді це лише посилання на об'єкт у пам'яті.
 - Змінні, що зберігають посилання, часто називають **посилальними** (reference variable).
- Посилання, яке повертає оператор `new`, представляється в кодї за допомогою зарезервованого слова `this`.
 - Там, де з'являється `this`, ідентифікується поточний об'єкт.
- У випадку з `image` явно конструктор не задається.
 - Коли конструктор класу не прописується програмістом, Java неявно створює його для класу - **безаргументний конструктор за умовчанням** (*default noargument constructor*)
 - створюється лише за умови відсутності інших конструкторів класу.

Явне створення конструктора

```
class Image
{
    Image()
    {
        System.out.println("Image() called");
    }
}
```

- конструктор симулює ініціалізацію за замовчуванням шляхом використання методу `System.out.println()` для виведення повідомлення про виклик конструктора.
 - Тоді об'єкт буде створюватись командою
 - `Image image = new Image();`
 - і виведеться повідомлення «Image() called.»

Кілька конструкторів класу

```
class Image
{
    Image(String filename)
    {
        this(filename, null);
        System.out.println("Image(String filename) called");
    }

    Image(String filename, String imageType)
    {
        System.out.println("Image(String filename, String imageType) called");
        if (filename != null)
        {
            System.out.println("reading " + filename);
            if (imageType != null)
                System.out.println("interpreting " + filename + " as storing a " +
                    imageType + " image");
        }
        // Perform other initialization here.
    }
}
```

- Деякі конструктори використовують інші конструктори для спрощення ініціалізації.
 - Усуває надмірний код.
- Image(String filename) покладається на
- Image(String filename, String imageType)
- для зчитування контенту зображення в пам'ять.
 - Виклик одного конструктора з іншого відбувається за допомогою this, оскільки власної назви вони не мають.
 - В реалізації це повинен бути перший рядок, що не дає можливості створювати декілька викликів конструктора.

Створення кількох об'єктів, використовуючи різні конструктори

- Кожне посилання на об'єкт присвоюється посилальній змінній image:

```
Image image = new Image("image.png");  
image = new Image("image.png", "PNG");
```

- Результат виклику:

```
Image(String filename, String imageType) called  
reading image.png  
Image(String filename) called  
Image(String filename, String imageType) called  
reading image.png  
interpreting image.png as storing a PNG image
```

Власні поля в конструкторі

```
Image(String filename, String imageType)
{
    System.out.println("Image(String filename, String imageType) called");
    if (filename != null)
    {
        System.out.println("reading " + filename);
        File file = new File(filename);
        // Read file contents into object.
        if (imageType != null)
            System.out.println("interpreting " + filename + " as storing a " +
                imageType + " image");
        else
            // Inspect image contents to learn image type.
            ; // Empty statement is used to make if-else syntactically valid.
    }
    // Perform other initialization here.
}
```

- Доступ до параметрів filename та imageType можливий будь-де в конструкторі.
 - Доступ до поля file можливий з моменту його оголошення до кінця блоку, в якому відбулось оголошення.
 - Ця властивість параметру або локальної змінної відома як **область видимості (scope)**.
- file є локальною змінною, а не параметром.
 - Параметри filename та imageType виникають в момент, коли конструктор починає виконуватись та існують до виходу з конструктора.
 - поле file створюється в момент його оголошення та продовжує існувати, поки не відбудеться вихід з блоку, де оголошується поле (закривається }).
 - Ця властивість називається **тривалістю зберігання (lifetime)** параметру чи локальної змінної.

Об'єкти та додатки

- Представимо мультикласовий проект, який містить три класи (Circle, Rectangle, Shapes):

```
class Circle
{
    Circle()
    {
        System.out.println("Circle() called");
    }

    public static void main(String[] args)
    {
        new Circle();
    }
}
```

Клас Shapes оголошено публічним, тому файл з вихідним кодом буде називатись Shapes.java.

Публічність класу є ідентифікатором точки входу в застосунок.

```
class Rectangle
{
    Rectangle()
    {
        System.out.println("Rectangle() called");
    }

    public static void main(String[] args)
    {
        new Rectangle();
    }
}

public class Shapes
{
    public static void main(String[] args)
    {
        Circle c = new Circle();
        Rectangle r = new Rectangle();
    }
}
```

Представлення стану за допомогою полів

- Зазвичай класи комбінують стани та поведінку.
 - **Стани** відповідають атрибутам, які зчитуються та/або записуються в процесі виконання застосунку.
 - **Поведінка** позначає набір інструкцій, які зчитують/записують стани та виконують інші дії. Комбінація станів та поведінки відома як **інкапсуляція**.
- Java представляє стан за допомогою **полів (fields)** – змінних, оголошених у тілі класу.
 - Стан, що пов'язується з класом, описується з допомогою **полів класу (class fields)**,
 - Стан, пов'язаний з об'єктами, описується **полями об'єктів (object fields, instance fields)**.
- За домовленістю назва поля починається з маленької літери, а кожне наступне слово багатослівної назви починається з великої букви.

Оголошення полів класу та доступ до них

- Поле класу зберігає атрибут, що пов'язаний з класом.
 - Всі об'єкти, що створюються з цього класу, отримують поля цього класу.
 - Коли об'єкт змінює значення поля, воно буде видимим для всіх поточних та створених від цього класу в майбутньому об'єктів.
- Ви оголошуєте поле класу за допомогою наступного синтаксису:
 - `static ім'я_типу назва_змінної [= вираз] ;`
 - Опційно змінній можна присвоїти сумісний за типом вираз, який називають **ініціалізатором поля класу** (*class field initializer*).

Приклад: класи Car та Cars

```
class Car
{
    static int counter = 0;

    Car()
    {
        counter++;
    }
}
```

```
public class Cars
{
    public static void main(String[] args)
    {
        System.out.println(Car.counter);
        Car myCar = new Car();
        System.out.println(Car.counter);
        Car yourCar = new Car();
        System.out.println(Car.counter);
    }
}
```

- потрібно відстежувати кількість об'єктів, які створюються з цього класу.
 - Вводиться поле класу counter – лічильник, який ініціалізується нулем
 - Префікс static передбачає, що існує єдина копія цього поля, а не окрема для кожного об'єкту.
 - Поле counter явно ініціалізується нулем.
 - Кожного разу, коли створюється об'єкт, вираз counter++ у конструкторі класу Car() збільшує counter на 1.
- Явно присвоювати counter = 0 не обов'язково.
- Поля класу ініціалізуються шляхом обнулення їх бітів.
- Ці поля будуть інтепретуватися як false, 'u0000' , 0.0, 0.0f, 0, 0L або null.

0
1
2

Константні поля класу

```
class Employee
{
    final static int RETIREMENT_AGE = 65;
}
```

- Якщо Ви спробуєте змінити константу RETIREMENT_AGE, компілятор повідомить про помилку.
- Порядок зарезервованих слів final і static можна змінювати:
 - static final int RETIREMENT_AGE = 65;
- Оголошення RETIREMENT_AGE є прикладом *compile-time константи*.
 - Оскільки константа статична (існує лише одна копія її незмінюваного значення), компілятор може оптимізувати байткод, підставляючи значення константи в усі обчислення, де вона згадується.
 - Код швидше виконується, оскільки йому не потрібно кожного разу отримувати доступ до read-only поля класу.

Оголошення та доступ до полів екземпляру (Instance Fields)

- **Поле екземпляру (*instance field*)** містить атрибут, пов'язаний з об'єктом.
 - Кожен об'єкт володіє власною копією атрибуту.
 - Наприклад, один об'єкт може мати атрибут кольору із значенням «червоний», а інший об'єкт – із значенням «зелений».
- Оголошення поля екземпляру має наступний синтаксис:
 - *назва_типу ім'я_змінної [= вираз] ;*
 - Вираз із сумісним типом називають **ініціалізатором поля екземпляру**.
 - Наприклад, автомобіль описується виробником, моделлю, кількістю дверей.
 - Ці атрибути не зберігаються в полях класу, оскільки автомобілі можуть мати різних виробників, моделі та кількість дверей.

```
class Car
{
    String make;
    String model;
    int numDoors;
}
```

- Коли створюється об'єкт, його поля ініціалізуються нульовими значеннями.
- Наприклад, для `Car car = new Car()` ; поля `make` і `model` набудуть значення `null`, а `numDoors` буде ініціалізуватись `0`.

Клас Cars

Напрямку має доступ до своїх полів екземпляру.

```
public class Cars
{
    public static void main(String[] args)
    {
        Car myCar = new Car();
        myCar.make = "Toyota";
        myCar.model = "Camry";
        myCar.numDoors = 4;
        System.out.println("Make = " + myCar.make);
        System.out.println("Model = " + myCar.model);
        System.out.println("Number of doors = " + myCar.numDoors);
    }
}
```

```
Make = Toyota
Model = Camry
Number of doors = 4
```

- В оголошенні класу доступ до поля екземпляру відбувається напряму,
 - System.out.println(numDoors);

Явна ініціалізація посилальних полів

- При доступі до поля екземпляру ззовні об'єкту необхідно дописувати посилальну змінну (reference variable) потрібного об'єкта, після чого йде оператор доступу «.» для поля екземпляру name.
 - Наприклад, задамо myCar.make для доступу до поля make об'єкта myCar.
 - Можна явно ініціалізувати поле екземпляру при його оголошенні, щоб забезпечити ненульове значення за умовчанням, яке переозначить (override) дефолтне значення.
 - Можна прибрати запис myCar.numDoors = 4;

```
class Car
{
    String make;
    String model;
    int numDoors = 4;
}
```

Ініціалізація в конструкторах

```
class Car
{
    String make;
    String model;
    int numDoors;

    Car(String make, String model)
    {
        this(make, model, 4);
    }

    Car(String make, String model, int nDoors)
    {
        this.make = make;
        this.model = model;
        numDoors = nDoors;
    }
}
```

- Зазвичай пряма ініціалізація полів екземпляру об'єкту в класі не дуже хороша ідея.
 - Краще виконувати ініціалізацію в конструкторі(ах) класу.
- клас Car оголошує конструктори
 - Car(String make, String model)
 - Car(String make, String model, int nDoors).
- Для мінімізації помилок краще слідкувати, щоб назви полів та імена параметрів відрізнялись.
 - З іншого боку, можна додавати префікс «_» до назви (наприклад, _nDoors).

Ініціалізація полів екземпляру за допомогою конструктора

```
public class Cars
{
    public static void main(String[] args)
    {
        Car myCar = new Car("Toyota", "Camry");
        System.out.println("Make = " + myCar.make);
        System.out.println("Model = " + myCar.model);
        System.out.println("Number of doors = " + myCar.numDoors);
        System.out.println();
        Car yourCar = new Car("Mazda", "RX-8", 2);
        System.out.println("Make = " + yourCar.make);
        System.out.println("Model = " + yourCar.model);
        System.out.println("Number of doors = " + yourCar.numDoors);
    }
}
```

- кожен з об'єктів має різні значення полів екземпляру

```
Make = Toyota
Model = Camry
Number of doors = 4
```

```
Make = Mazda
Model = RX-8
Number of doors = 2
```

Загальні правила доступу до полів

- Інколи можна просто задати назву поля, а в решті випадків необхідно дописувати префікс з посиланням на об'єкт або назвою класу та оператором доступу.
 - Задавайте назву поля класу (без префіксів) будь-де всередині класу, в якому поле оголошено: `counter`.
 - Пишіть перед назвою *поля класу* назву класу та оператор доступу при доступі *ззовні класу*: `Car.counter`.
 - Записуйте назву поля класу **без префіксів** у **будь-якому** методі екземпляру, конструкторі або ініціалізаторі екземпляру в тому ж класі, де оголошено поле екземпляру: `numDoors`.
 - Задавайте посилання на об'єкт з оператором доступу та назвою поля екземпляру в **будь-якому** методі класу або ініціалізаторі класу в тому ж класі, що й оголошення поля екземпляру, або ззовні класу:
 - `Car car = new Car(); car.numDoors = 2;`
- Правила не є повними.

Ініціалізатори класу (Class Initializers)

```
class JDBCFilterDriver implements Driver
{
    static private Driver d;

    static
    {
        // Attempt to load JDBC-ODBC Bridge Driver and register that
        // driver.
        try
        {
            Class c = Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            d = (Driver) c.newInstance();
            DriverManager.registerDriver(new JDBCFilterDriver());
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
    //...
}
```

- Конструктори виконують задачі ініціалізації для об'єктів.
- Всередині класу їм відповідають **ініціалізатори класу (*class initializer*)**.
 - Ініціалізатор класу – статичний блок, представлений в тілі класу.
 - Використовується для ініціалізації завантаженого класу через послідовність інструкцій.

Мікс ініціалізаторів класу та ініціалізаторів полів класу

```
class C
{
    static
    {
        System.out.println("class initializer 1");
    }

    static int counter = 1;

    static
    {
        System.out.println("class initializer 2");
        System.out.println("counter = " + counter);
    }
}
```

- Клас C містить два ініціалізатори класу та один ініціалізатор поля класу.
 - При компіляції в class-файл у такому випадку буде створюватись спеціальний метод класу void <clinit>().
 - Він містить bytecode-еквівалент усіх ініціалізаторів класу та полів класу в порядку їх появи.
- При завантаженні класу C в пам'ять, <clinit>() негайно виконується і генерує вивід:

```
class initializer 1
class initializer 2
counter = 1
```

Ініціалізатори екземпляру класу (Instance Initializers)

```
class Graphics
{
    double[] sines;
    double[] cosines;

    {
        sines = new double[360];
        cosines = new double[sines.length];
        for (int degree = 0; degree < sines.length; degree++)
        {
            sines[degree] = Math.sin(Math.toRadians(degree));
            cosines[degree] = Math.cos(Math.toRadians(degree));
        }
    }
}
```

- У Java існують анонімні класи, які не мають конструкторів (розглянемо пізніше).
- Для цих класів Java пропонує ініціалізатор екземпляру для обробки задач ініціалізації.

Ініціалізатор екземпляру класу (*instance initializer*) – блок, представлений в тілі класу, проте не в тілі методу чи конструкторі.

міх ініціалізаторів екземпляру та ініціалізаторів полів екземпляру

```
class C
{
    {
        System.out.println("instance initializer 1");
    }

    int counter = 1;

    {
        System.out.println("instance initializer 2");
        System.out.println("counter = " + counter);
    }
}
```

Клас C задає два ініціалізатори екземпляру та один ініціалізатор поля класу.

- У class-файлі створюється спеціальний метод void <init>(), що представляє default noargument constructor, якщо явно не оголошено конструктор; інакше – створюється метод <init>() для кожного розглянутого конструктора.

Коли виконується new C(), негайно запускається <init>() та генерує вивід:

```
instance initializer 1
instance initializer 2
counter = 1
```

```
public class InitDemo
{
    static double double1;
    double double2;
    static int int1;
    int int2;
    static String string1;
    String string2;

    static
    {
        System.out.println("[class] double1 = " + double1);
        System.out.println("[class] int1 = " + int1);
        System.out.println("[class] string1 = " + string1);
        System.out.println();
    }

    {
        System.out.println("[instance] double2 = " + double2);
        System.out.println("[instance] int2 = " + int2);
        System.out.println("[instance] string2 = " + string2);
        System.out.println();
    }

    static
    {
        double1 = 1.0;
        int1 = 1000000000;
        string1 = "abc";
    }

    {
        double2 = 1.0;
        int2 = 1000000000;
        string2 = "abc";
    }
}
```

Демонстрація порядку ініціалізації

```
InitDemo()
{
    System.out.println("InitDemo() called");
    System.out.println();
}

static double double3 = 10.0;
double double4 = 10.0;

static
{
    System.out.println("[class] double3 = " + double3);
    System.out.println();
}

{
    System.out.println("[instance] double4 = " + double3);
    System.out.println();
}
```

Демонстрація порядку ініціалізації

```
public static void main(String[] args)
{
    System.out.println ("main() started");
    System.out.println();
    System.out.println("[class] double1 = " + double1);
    System.out.println("[class] double3 = " + double3);
    System.out.println("[class] int1 = " + int1);
    System.out.println("[class] string1 = " + string1);
    System.out.println();
    for (int i = 0; i < 2; i++)
    {
        System.out.println("About to create InitDemo object");
        System.out.println();
        InitDemo id = new InitDemo();
        System.out.println("id created");
        System.out.println();
        System.out.println("[instance] id.double2 = " + id.double2);
        System.out.println("[instance] id.double4 = " + id.double4);
        System.out.println("[instance] id.int2 = " + id.int2);
        System.out.println("[instance] id.string2 = " + id.string2);
        System.out.println();
    }
}
```

```
[class] double1 = 0.0
[class] int1 = 0
[class] string1 = null
```

```
[class] double3 = 10.0
```

```
main() started
```

```
[class] double1 = 1.0
[class] double3 = 10.0
[class] int1 = 1000000000
[class] string1 = abc
```

```
About to create InitDemo object
```

```
[instance] double2 = 0.0
[instance] int2 = 0
[instance] string2 = null
```

```
[instance] double4 = 10.0
```

```
InitDemo() called
```

```
id created
```

```
[instance] id.double2 = 1.0
[instance] id.double4 = 10.0
[instance] id.int2 = 1000000000
[instance] id.string2 = abc
```

```
About to create InitDemo object
```

```
[instance] double2 = 0.0
[instance] int2 = 0
[instance] string2 = null
```

```
[instance] double4 = 10.0
```

```
InitDemo() called
```

```
id created
```

```
[instance] id.double2 = 1.0
[instance] id.double4 = 10.0
[instance] id.int2 = 1000000000
[instance] id.string2 = abc
```

Цікаві факти про ініціалізацію

- Поля класу ініціалізуються явними або дефолтними значеннями лише після завантаження класу.
 - Відразу після завантаження всі поля класу обнуляються до значень за замовчуванням.
 - Код методу `<clinit>()` виконує явну ініціалізацію.
- Повна ініціалізація класу відбувається до повернення методу `<clinit>()`.
- Поля екземпляру класу ініціалізуються дефолтними або явними значеннями в процесі створення об'єкту.
 - Коли оператор `new` виділяє пам'ять об'єкту, він обнуляє поля екземпляру до значень за замовчуванням.
- Код у методі `<init>()` виконує явну ініціалізацію.
- Повна ініціалізація екземпляру відбувається до повернення методу `<init>()`.



ДЯКУЮ ЗА УВАГУ!

Наступне запитання: породжені типи даних