





Lesson 13

06.03.2025




```
public class Ex1 {  
    public static void main(String[] args) {  
        Salmon s = new Salmon();  
        System.out.println(s.count);  
    }  
}
```

```
class Salmon{  
    int count;  
  
    public Salmon() {  
        count = 4;  
    }  
}
```



```
public class Ex2 {  
    public static void main(String[] args) {  
        int x = 0;  
        while (x++ < 10){}  
        String message = x > 10 ? "Grather than" : false;  
        System.out.println(message+", "+x);  
    }  
}
```

```
public class Ex3 {  
    public static void main(String[] args) {  
        int rez = 5 * 4 % 3;  
        System.out.println(rez);  
    }  
}
```



```
public class Ex4 {  
    public static void main(String[] args) {  
        for (int i = 0; i < 10; ) {  
            i = i++;  
            System.out.println("Hello World");  
        }  
    }  
}
```

```
public class Ex5 {  
    public static void main(String[] args) {  
        int m = 9, n = 1, x = 0;  
        while (m > n) {  
            m--;  
            n += 2;  
            x += m + n;  
        }  
        System.out.println(x);  
    }  
}
```



Поліморфізм

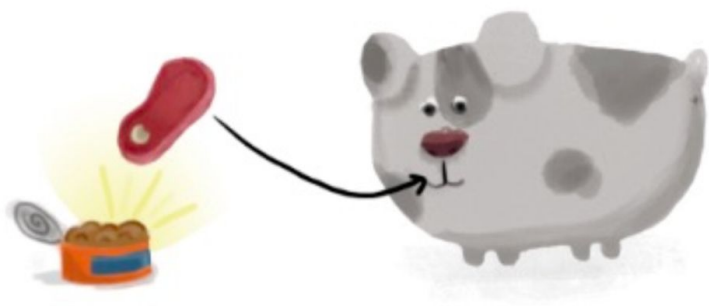
Поліморфізм (від грец. "багато форм") — це одна з ключових концепцій об'єктно-орієнтованого програмування (ООП), що дозволяє об'єктам різних класів мати однаковий інтерфейс для роботи з ними.

Основні переваги поліморфізму:

- Гнучкість та розширюваність коду
- Зменшення дублювання коду
- Спрощення підтримки та модифікації програмного забезпечення



Polymorphism



each animal can eat
its own type of food



Реалізація поліморфізму в Java

У Java поліморфізм реалізується через:

1. **Перевизначення методів (Method Overriding)** — підклас змінює поведінку методу батьківського класу.
2. **Перевантаження методів (Method Overloading)** — в одному класі створюються методи з однаковими іменами, але різними параметрами.

Інтерфейс – це контракт, в рамках якого частини програми, найчастіше написані різними людьми, взаємодіють між собою та із зовнішніми додатками. Інтерфейси працюють з шарами сервісів, безпеки, DAO та і т.д. Це дозволяє створювати модульні конструкції, у яких для зміни одного елемента не потрібно чіпати решту.

```
package com.hillel;


public interface Say {

    void sayHello();

    default void sayGoodbye() {
        System.out.println("Goodbye ... ");
    }
}
```

У класі, що імплементує інтерфейс, повинні бути реалізовані всі передбачені інтерфейсом методи, за винятком методів замовчуванням (**default**).

Методи за замовчуванням вперше з'явилися в Java 8. Їх позначають модифікатором default. У нашому прикладі це метод sayGoodbye, реалізація якого прописано прямо в інтерфейсі. Дефолтні методи спочатку готові до використання, але при необхідності їх можна перевизначати в застосовуючи інтерфейс класи.



Якщо інтерфейс має лише один абстрактний метод, перед нами функціональний інтерфейс. Його прийнято позначати інструкцією **@FunctionalInterface**, яка вказує компілятору, що при виявленні другого абстрактного методу у цьому інтерфейсі потрібно повідомити про помилку. Стандартних (default) методів у інтерфейсу може бути безліч – у тому числа що належать класу java.lang.Object

```
@FunctionalInterface
public interface Developer {

    boolean isDeveloper();
}
```




Інтерфейси та поліморфізм.

У Java поліморфізм можна реалізувати через:

успадкування - з перевизначенням параметрів та методів базового класу;

абстрактні класи - шаблони для роздільної реалізації у різних класах;

інтерфейси – для імплементзації класами.



Крім звичайних класів Java є **абстрактні класи**. Абстрактний клас схожий на клас. В абстрактному класі також можна визначити поля та методи, водночас не можна створити об'єкт або екземпляр абстрактного класу. Абстрактні класи покликані надавати базовий функціонал для класів спадкоємців. А похідні класи вже реалізують цей функціонал.

```
public abstract class Human {  
  
    abstract void see();  
  
    public void talk(String str){  
        System.out.println(str);  
    }  
  
    public void hear(String str){  
        System.out.println(str);  
    }  
}
```

- Інтерфейс описує лише поведінку. Він не має стану. А у абстрактного класу стан є: він описує і те, й інше.
- Абстрактний клас пов'язує між собою та об'єднує класи, що мають дуже близький зв'язок. У той же час, той самий інтерфейс можуть реалізовувати класи, які взагалі немає нічого спільного.
- Класи можуть реалізовувати скільки завгодно інтерфейсів, але успадковуватися можна лише від одного класу



Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance .	Interface supports multiple inheritance .
3) Abstract class can have final, non-final, static and non-static variables .	Interface has only static and final variables .
4) Abstract class can provide the implementation of interface .	Interface can't provide the implementation of abstract class .
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7) An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.