POLIMORFIZM

WYJAŚNIENIE:

Przyjmowanie wielu postaci. Jak to osiągnąć w programowaniu? Poprzez dziedziczenie albo poprzez implementowanie interfejsów.

Polimorfizm przez dziedziczenie

Mamy klasy z konstruktorami oraz z geterami i seterami. (nie pokazano wszystkiego)

```java
public class Book {
    private String title;
    private String author;
    private int pages;

    public Book(final String title, final Str
        this.title = title;
        this.author = author;
        this.pages = pages;
    }
```

```java
public class EBook extends Book{
    private String type; //EPUB, MOBI, PDF

    public EBook(final String title, final Str
        super(title, author, pages);
        this.type = type;
    }
```

```java
public class NormalBook extends Book{
    private String coverType; //Solid, soft

    public NormalBook(final String title, final Stri
        super(title, author, pages);
        this.coverType = coverType;
    }
```

Klasa EBook dziedziczy po klasie Book, oraz klasa NormalBook dziedziczy po klasie Book.
W każdej klasie istnieje dokładnie taka sama metoda różniąca się jednak wykonywanymi działaniami: (po kolei: Book, EBook i NormalBook.

```java
public void printInfo(){
    System.out.println("Title: "+title+" author: "+author+" number of page: "+pages);
}
public void printInfo(){
    System.out.println("Type: "+ type);
}
public void printInfo(){
    System.out.println("Cover Type: "+ coverType);
}
```

Teraz zróbmy działający program używający polimorfizmu.

```java
public static void main(String[] args) {
    Book b1 = new Book( title: "Java", author: "adams", pages: 4);
    //Użycie polimorfizmu!!!
    Book b2 = new EBook( title: "Python", author: "janek", pages: 12, type: "EPUB");
    Book b3 = new NormalBook( title: "C++", author: "apple", pages: 7, coverType: "solid");

    //Co w rezultacie otrzymamy?
    b1.printInfo();
    b2.printInfo();
    b3.printInfo();
}
```

Jak mogłoby się wydawać wynik nie jest oczywisty!

```
Title: Java author: adams number of page: 4
Type: EPUB
Cover Type: solid
```

Teraz wyjaśnienie!!!

Poniższy zapis jest jak najbardziej poprawny ponieważ klasa EBook **rozszerza/dziedziczy** klasę Book.

```
public class EBook extends Book{
```

(to samo tyczy się klasy NormalBook)

```
Book b2 = new EBook("Python","janek",12,"EPUB");
```

Można nawet użyć metody printInfo() - tylko która to będzie metoda??? Okazuje się, że JVM wie, że należy wybrać metodę z klasa EBook. Jednak jest to bardzo nieefektywne programowanie. Zastanówmy się, co się stanie, jeśli ktoś będzie chciał zmienić nazwę metody lub zwracany typ lub przyjmowane wartości lub doda nową metodę → będzie problem z kompilacją!!!

Klasy abstrakcyjne i polimorfizm.

Przykładowa klasa abstrakcyjna jest pokazana niżej:

```
//Z tej klasy nie można stworzyć OBIEKTU -> AbstractCar ac = new AbstractCar();  NIE DZIAŁA
public abstract class AbstractCar {
    //Może posiadać pola - FIELDS
    private String brand;
    private String type;

    public abstract String carType();  //Definiowanie abstrakcyjnej metody - bez ciała

    //Można stworzyć normalną metodę
    public void showStatus(boolean flag){
        if(flag) System.out.println("AVAILABLE");
        else System.out.println("NOT AVAILABLE");
    }

}
```

Po powyższej klasie możemy dziedziczyć

```
//Dziedziczenie
public class Car extends AbstractCar {
    //Konstruktor ustawiający wartości pól z AbstractCar
    public Car(String brand, String type) {
        //W klasie abstrakcyjnej muszą być GETTERY i SETTERY do pól prywatnych
        setBrand(brand);
        setType(type);
    }

    //Tą metodę musimy nadpisać @Override oraz umieścić w niej jakiś kod
    @Override
    public String carType() {
        return "Type: "+getType(); //W klasie abstrakcyjnej muszą być GETTERY i SETTERY do pól prywatnych
    }
}
```

Klasa uruchomieniowa – próba użycia POLIMORFIZMU – **wydaje się że działa!**

```
public class AbstractInterfaceDemo{
    public static void main(String[] args) {
        AbstractCar car = new Car( brand: "Porshe", type: "Macan");
        car.carType();
        car.showStatus( flag: true); //metoda z AbstractCar
    }
}
```

Wynik – o dziwo:

```
AVAILABLE
```

brakuje:

```
@Override
public String carType() {
    return "Type: "+getType();
}
```

Więc widać że polimorfizm z użyciem klas abstrakcyjnych nie do końca działa!!!

Polimorfizm z użyciem interfejsów

Mamy interfejs Samochód

```
public interface Samochod {
    double PI = 3.14;
    void move(int distance);
    boolean working(boolean working);
}
```

i pusty interfejs Vehicle

```
public interface Vehicle {
}
```

Zaimplementujmy je w klasach:

```
public class Track implements Samochod, Vehicle{
    @Override
    public void move(final int distance) {
        System.out.println("Track -> "+distance);
    }

    @Override
    public boolean working(final boolean working) {
        return false;
    }
}
```

```
public class Tir implements Samochod, Vehicle{
    @Override
    public void move(final int distance) {
        System.out.println("TIR "+distance);
    }

    @Override
    public boolean working(final boolean working) {
        return false;
    }
}
```

Jak widać należy stworzyć metody z interfejsów.

Teraz możemy użyć **POLIMORFIZMU** do bardziej responsywnego programu.

Kod klasy startowej:

```java
public class AbstractInterfaceDemo{
    public static void main(String[] args) {
        Samochod samochod = null; //Przygotowanie do polimorfizmu
        int chooser = 7;
        if(chooser == 0){   //W zależności co wybierzemy otrzymamy Tir lub Track
            samochod = new Tir();
        } else{
            samochod = new Track();
        }
        samochod.move( distance: 10);  //Użycie polimorfizmu
    }
}
```

Wynik działania powyższego programu:

```
Track -> 10
```

po zmianie wartości:

```java
int chooser = 0;
```

wynik działania programu:

```
TIR 10
```

Zadania

1. Napisz program – kalkulator. Stwórz interface Calculation z jedną metodą przyjmującą listę liczb zmiennoprzecinkowych i zwracającą wynik typu double. Następnie stwórz klasy implementujące ten interfejs o nazwach: Sum, Diff, Multi, Divid. Następnie napisz program, który będzie pobierał od użytkownika jakie działanie chce wykonać, następnie pobierze od użytkownika liczby (dowolną ilość – może być określona na początku lub zakończenie podawania liczb zrobione po wpisaniu jakiegoś znaku) następnie wykona odpowiednie działania używając polimorfizmu (patrz przykład w tym pliku).