PROGRAMOWANIE OBIEKTOWE JAVA – LABORATORIUM

KLASY ABSTRAKCYJNE I INTERFEJSY

Klasa abstrakcyjna jest klasą, z której nie można utworzyć instancji obiektu. Posiada ona następujące własności:

- Oznaczamy ją modyfikatorem abstract.
- Klasa może (ale nie musi) zawierać zarówno regularne metody Java jak i takie oznaczone modyfikatorem abstract.
- Klasa abstrakcyjna może być rozszerzana przez inne klasy Java - zarówno zwykłe jak i abstrakcyjne.

W Javie możemy rozszerzać tylko jedną klasę na raz i tak samo jest w przypadku klas abstrakcyjnych.

Przykład 1:

```
abstract class className {
// ...
}
```

Klasa abstrakcyjna może zawierać deklaracje metod abstrakcyjnych, zwykłe metody, stałe i składowe finalne.

```
abstract class Example
{
    public static final double PI = 3.14; // deklaracja stałej

    abstract String Something(); // metoda abstrakcyjna (bez ciała)

    public void Message() // zwykła metoda
    {
        System.out.println("Hello World!");
    }
}
```

Aby lepiej zdefiniować sens abstrakcji w programowaniu posłużmy się przykładem z życia. Klasą abstrakcyjną może być figura geometryczna. Nie wiemy jak wygląda "figura", dla nas to ogólne pojęcie, abstrakcyjny byt, którego nie potrafimy sobie wyobrazić. Możemy natomiast zdefiniować właściwości owej figury, np: pole powierzchni, obwód, które mogą być dziedziczone przez konkretne klasy potomne (i które możemy sobie wyobrazić np: Kwadrat, Prostokat, Trójkąt itd.). ***Klasa abstrakcyjna jest pewnego rodzaju wzorcem dla klas potomnych.***

```
package KlasyAbstrakcyjne;

public abstract class Figura {
    abstract double Pole();
    abstract double Obwod();
}
```

Klasa Figura zawiera dwie metody abstrakcyjne, które zwracają wartości zmiennoprzecinkowe – double. W odróżnieniu od interfejsów metody abstrakcyjne muszą określać typ zwracanych wartości. Klasy potomne, dziedziczące klasę abstrakcyjną muszą zaimplementować wszystkie jej metody abstrakcyjne. Ponadto klasa abstrakcyjna może posiadać również zwykłe metody, które nie muszą być implementowane w klasach potomnych.

Z powyższego przykładu wiemy już, że klasy potomne będą obliczać pole powierzchni i obwód danej figury. Klasa abstrakcyjna nie podaje konkretów dla metod obliczeniowych, bo każda figura ma przecież

inny wzór matematyczny na przeliczanie tych niewiadomych. Jak widać, jest to elastyczne podejście dające nam dużo swobody w projektowaniu. Przykłady klas potomnych implementujących klasę abstrakcyjną Figura:

```
package KlasyAbstrakcyjne;

public class Kwadrat extends
Figura{
    public double a;

    public double Pole()
    {
        return a*a;
    }

    public double Obwod()
    {
        return 4*a;
    }
}
```

```
package KlasyAbstrakcyjne;

public class Prostokat extends
Figura{
    public double a, b;

    public double Pole()
    {
        return a*b;
    }

    public double Obwod()
    {
        return 2*a + 2*b;
    }
}
```

Użycie klasy abstrakcyjnej i klas dziedziczących jej składowe:

```
package KlasyAbstrakcyjne;

public class FiguraTest {
    public static void main(String[] args) {
        Kwadrat kw1 = new Kwadrat();
        kw1.a = 10;
        System.out.println("Pole kwadrat = " + kw1.Pole());
        System.out.println("Obwód kwadrat = " + kw1.Obwod());

        Prostokat prostokat1 = new Prostokat();
        prostokat1.a = 5;
        prostokat1.b = 2;
        System.out.println("Pole prostokąt = " + prostokat1.Pole());
        System.out.println("Obwód prostokąt = " + prostokat1.Obwod());
    }
}
```

INTERFEJSY

Interfejs to “szablon” zawierający elementy, które muszą być użyte w klasach, które go zaimplementują. Interfejsy mogą zawierać tylko stałe i deklaracje metod. Implementacja interfejsu w danej klasie odbywa się za pomocą słowa kluczowgo implements.

Konstrukcja interfejsu:

```
modyfikator_dostepu interface Nazwa_interfejsu {
// deklaracja stałych i/lub metod
}
```

Implementacja interfejsu:

```
modyfikator_dostepu class Nazwa_klasy implements Nazwa_interfejsu {
// deklaracja składowych i/lub metod
}
```

W odróżnieniu od dziedziczenia dana klasa może implementować wiele interfejsów:

```
modyfikator_dostepu class Nazwa_klasy implements Interfejs1, Interfejs2 //
itd... {
```

```
// kod klasy
}
```
Interfejsy mogą również dziedziczyć po wielu interfejsach:

```
modyfikator_dostepu interface Nazwa_interfejsu extends Interfejs1,
Interfejs2 //... {
}
```
Przykład interfejsu:

```
package KlasyAbstrakcyjne;

public interface FiguraGeometryczna {

    double Pole();
    double Obwod();

}
```
Zwróć uwagę, że interfejs FiguraGeometryczna mówi klasie “co ma zrobić”, ale nie mówi w jaki sposób ma to zrobić. Mamy tylko deklaracje nazw metod: Pole() i Obwod(). Wszystkie stałe i metody interfejsu są domyślnie publiczne i abstrakcyjne dlatego nie jest wymagane pisanie przed nimi słów kluczowych np: public.

Analizując ten prosty interfejs możemy z łatwością domyślić się, że będzie on kazał klasie obliczyć pole i obwód danej figury geometrycznej.

Napiszmy więc klasę Kwadrat i wprowadźmy do niej odpowiednie wzory:

```
public class Kwadrat implements FiguraGeometryczna {

    private double a; // długość boku / podstawy

    public void setA(double a) {
        this.a = a;
    }

    public double getA() {
        return a;
    }

    public double Pole() {
        return a*a;
    }

    public double Obwod() {
        return 4*a;
    }

}
```
Obie metody (Pole i Obwod) zwracają wynik działania. Dodajmy jeszcze jedną figurę – prostokąt. Klasa Prostokat, podobnie jak Kwadrat implementuje interfejs FiguraGeometryczna.

```
public class Prostokat implements FiguraGeometryczna {

    private double a; // długość
    private double h; // wysokość

    public void setA(double a) {
        this.a = a;
    }

    public double getA() {
```

```
        return a;
    }

    public void setH(double h) {
        this.h = h;
    }

    public double getH() {
        return h;
    }

    public double Pole() {
        return a*h;
    }

    public double Obwod() {
        return 2*a+2*h;
    }

}
```

Klasy Kwadrat i Prostokat muszą zaimplementować wszystkie metody interfejsu FiguraGeometryczna, ale nic nie stoi na przeszkodzie, aby posiadały dodatkowe metody wewnątrz własnych klas np: wzór na przekątną. Na koniec stwórzmy klasę testującą nasz program – FiguraGeometrycznaTest:

```
public class FiguraGeometrycznaTest {
    public static void main(String[] args)
    {
        Kwadrat kw1 = new Kwadrat();
        kw1.setA(10);
        System.out.println("Pole kwadrata = " + kw1.Pole());
        System.out.println("Obwód kwadrata = " + kw1.Obwod());

        Prostokat prostokat1 = new Prostokat();
        prostokat1.setA(5);
        prostokat1.setH(2);
        System.out.println("Pole prostokąta = " + prostokat1.Pole());
        System.out.println("Obwód prostokąta = " + prostokat1.Obwod());

        // Wyświetli:
        // Pole kwadratu = 100.0
        // Obwód kwadratu = 40.0
        // Pole prostokąta = 10.0
        // Obwód prostokąta = 14.0
    }
}
```

Metody domyślne w interfejsach

W Javie jest możliwość dodawania do intefejsów metod zawierających ciało. Metody te używają słowa kluczowego **default** i są nazywane metodami domyślnymi. Jako, że są one od razu zaimplementowane (nie są abstrakcyjne), nie jest wymagane dostarczanie ich implementacji w klasach dziedziczących. W razie potrzeby mogą być one nadpisywane w tych klasach.

```
package KlasyAbstrakcyjne;

public interface
FiguraGeometryczna {

    double Pole();
    double Obwod();
```

```
package KlasyAbstrakcyjne;

public class FiguraGeometrycznaTest {
    public static void main(String[]
args)
    {
        Kwadrat kw1 = new Kwadrat();
```

```
    default String getFullName()
{
        return "Interfejs Figura
geometryczna";
    }

}
```

```
        kw1.setA(10);
        System.out.println("Pole
kwadrata = " + kw1.Pole());
        System.out.println("Obwód
kwadrata = " + kw1.Obwod());

System.out.println(kw1.getFullName());

        Prostokat prostokat1 = new
Prostokat();
        prostokat1.setA(5);
        prostokat1.setH(2);
        System.out.println("Pole
prostokąta = " + prostokat1.Pole());
        System.out.println("Obwód
prostokąta = " + prostokat1.Obwod());

        /*Wyświeli
        Pole kwadrata = 100.0
        Obwód kwadrata = 40.0
        Interfejs Figura geometryczna
        Pole prostokąta = 10.0
        Obwód prostokąta = 14.0*/
    }
}
```

Stałe wartości w interfejsach

W Javie oprócz zmiennych możemy wprowadzić do kodu także wartości stałe. Będziemy jeszcze o tym mówić w przyszłości, ale na ta chwilę istotne jest aby wiedzieć, że stałą definiujemy za pomocą połączenia dwóch słów kluczowych **static final**. Dodatkowo nazwa stałej powinna być pisana dużymi literami (w razie nazwy złożonej z kilku członów - rozdzielamy je znakiem podkreślenia _):

```
public class MovieItem {

    public static final String LABEL = "Default item name";

}
```

Stałą wartość możemy zainicjować wartością tylko w trakcie tworzenia. Możemy to zrobić albo od razu w polu, albo w konstruktorze. Nie możemy później zmienić takiej wartości. Przypisanie nowej wartości zakończy się błędem kompilacji.

```
public class MovieItem {

    public static final String LABEL = "Default item name";

    public void updateLabel() {
        this.LABEL = "Default item name created by JavAPPa"; //nie
skompiluje się
    }

}
```

Co to wszystko ma wspólnego z interfejsami? Okazuje się, że w interfejsach również możemy tworzyć stałe, ale z tą różnicą, że są one deklarowane automatycznie i niejawnie. Tak więc nie wpisujemy w kodzie słów static final:

```
package KlasyAbstrakcyjne;

public interface FiguraGeometryczna {
```

```
    double Pole();
    double Obwod();
    public String LABEL = "Moje figury geometryczne";
    default String getFullName() {
        return "Interfejs Figura geometryczna";
    }

}
```

Zadania do samodzielnego rozwiązania:

Zaimplementować aplikację za pomocą interfejsów i klas abstrakcyjnych.

Zadanie 1. Transport

Zbuduj system, który obsługuje różne typy pojazdów dostępnych w wypożyczalni (np. samochody, rowery, hulajnogi elektryczne). System powinien umożliwiać:

- dodawanie pojazdów,
- wypożyczanie,
- obliczanie kosztu wynajmu,
- zwracanie pojazdu.

Zadanie 2.

1. Uruchomić i zapoznać się z kodem plików znajdujących się w rozpakowanym pliku. Dołącz wszytki do nowego projektu.
2. W programie głównym utworzyć reprezentację każdej klasy, która jest w projekcie i zaproponuj uruchomienie przykładowych trzech funkcji.
3. Zamienić klasę Figura na klasę abstrakcyjną
4. Dodaj do klasy Figura metodę abstrakcyjną String opis(), zwracającą informację o obiekcie, np. „Obiekt klasy Prostokat”.
5. Zmodyfikować pozostałe klasy, tak, aby program kompilował się poprawnie.
6. W programie głównym utworzyć tablicę o nazwie tablicaFigur o rozmiarze 10 typu Figura. Do każdego elementu tablicy utworzyć nowy obiekt, lub przypisać istniejący.
7. Dla każdego obiektu tablicy wywołać metodę opis(). Z której klasy została wywołana metoda opis() i dlaczego?
8. W klasie Figura zdefiniować dodatkowe metody abstrakcyjne i zaimplementuj je w klasach potomnych:
   a) void skaluj(float skala) //skaluje wielkość figury
9. Zdefiniuj interfejs o nazwie IFigury, zawierający następujące metody:
   b) float getPowierzchnia();
   c) boolean wPolu(Punkt p);
10. Zaimplementuj interfejs IFigury we wszystkich klasach Prostokat, Trojkat, Kwadrat.
11. W programie głównym zadeklaruj listę obiektów typu IFigury, następnie przypisz do niej różne figury: Prostokat, Kwadrat, Trojkat.

12. Dla każdego obiektu tablicy IFigur wywołaj metodę getPowierzchnia() i w_polu().

Kolejno:

1. Utwórz klasę Okrag dziedziczącą po klasie Figura
2. Zdefiniuj interfejs o nazwie RuchFigury, zawierający następującą metodę:
3. void przesun(int x, int y);
4. Zaimplementuj interfejs RuchFigury w klasie Okrąg.
5. Sprawdź działanie interfejsu.