✸ INTERFEJSY

Interfejsy w programowaniu obiektowym to zbiór wymagań dotyczących klas, które będą go stosować. Wyrażają sposób opisu, co klasa powinna robić bez określania, w jaki sposób będzie to uzyskane. W skład interfejsów wchodzą metody oraz pola.
Deklaracja metod składa się wyłącznie z nagłówków (brak jest ciała metody), natomiast pola interfejsu to wyłącznie stałe statyczne z jawnie określoną wartością. Wszystkie składowe interfejsu muszą być publiczne.
Składnia definicji interfejsu podobna jest do definicji klasy. Zamiast słowa **class** stosowane jest słowo kluczowe **interface**. Możliwe jest równie dziedziczenie interfejsów.

```
interface Dzwoni {
public static String NUMER_ALARMOWY = "112";
public void zadzwon(String);
public void zadzwonNaNrAlarmowy();
}
```

Proces implementacji polega na umieszczeniu słowa kluczowego **implements** w nagłówku definicji klasy, po którym występują nazwy interfejsów, które klasa implementuje. Zatem składowymi klasy są wszelkie pola i metody występujące w definicji klasy oraz pola i metody określone w definicji implementowanych interfejsów. Poniższy kod programu ilustruje implementację interfejsu Dzwoni przez klasę Telefon.

```
class Telefon implements Dzwoni{

// deklaracja pól
private String numerTelefonu;
private int lacznyCzasRozmow;
private static double cenaRozmowy = 0.48; // z</min.

// konstruktor
public Telefon (String numer) {
numerTelefonu = numer;
}

// deklaracja metod
public double obliczKwoteDoZaplaty() {
return cenaRozmowy * (lacznyCzasRozmow / 60);
}
public static void ustawCeneRozmowy(double nowaCena){
cenaRozmowy = nowaCena;
}

// metody wymagane przez interfejs
public void zadzwon(String nrTelefonu) {
System.out.println ("Dzwoni? do: " + nrTelefonu);
System.out.println ("Dry@, dry@...");
System.out.println ("Rozmowa w toku...");
int czasRozmowy = (int) (Math.random()*3600);
lacznyCzasRozmow += czasRozmowy;
System.out.println ("Rozmowa zako@czona. ");
System.out.printf ("Czas rozmowy: %d min. %d sek.",
czasRozmowy/60, czasRozmowy%60);
}
public void zadzwonNaNrAlarmowy() {
System.out.println ("Dzwoni? do: " + Dzwoni.NUMER_ALARMOWY);
System.out.println ("Dry@, dry@...");
```

```
System.out.println ("Centrum pomocy, s<ucham");
}
}
public class Test {
public static void main(String args[]) {
Telefon telefonKasi = new Telefon("1276594633");
telefonKasi.zadzwon("606342765");
telefonKasi.zadzwonNaNrAlarmowy();
}
}
```

Interfejsy, podobnie jak klasy, mogą być używane jako typ danych przy deklaracji zmiennej. Wartością takiej zmiennej może być odwołanie do obiektu dowolnej klasy, która implementuje dany interfejs

✸ KLASY I METODY ABSTRAKCYJNE

Klasy abstrakcyjne wykorzystywane są do modelowania rzeczywistości na dość ogólnym poziomie. Zawierają najbardziej znaczące cechy i zachowania konkretnego typu. Nie wszystkie zachowania (metody) muszą zostać zaimplementowane. Część z nich (nawet wszystkie) może składać się wyłącznie z nagłówka (metody abstrakcyjne). Umożliwia to stworzenie ogólnego modelu danego typu bez konieczności zagłębiania się w szczegóły implementacyjne.
Klasa abstrakcyjna nie może stanowić podstawy do utworzenia obiektu, możliwe jest natomiast tworzenie klas pochodnych. Klasa, która dziedziczy po klasie abstrakcyjnej musi zaimplementować wszystkie metody abstrakcyjne występujące w klasie bazowej.
Deklaracja klasy abstrakcyjnej odbywa się za pomocą słowa kluczowego **abstract** umieszczonego przed nazwą klasy. W podobny sposób tworzone są metody abstrakcyjne.

```
abstract class Figura {
protected String kolor;
public Figura(String kolor) {
this.kolor = kolor;
}

public abstract double obliczPowierzchnie();
}

class Kwadrat extends Figura {
private double bok;
public Kwadrat(String kolor, double bok) {
super(kolor);
this.bok = bok;
}
public double obliczPowierzchnie() {
return bok * bok;
}
}

class Prostokat extends Figura {
private double szerokosc;
private double wysokosc;

public Prostokat(String kolor, double szerokosc, double wysokosc) {
```

```
super(kolor);
this.szerokosc = szerokosc;
this.wysokosc = wysokosc;
}

public double obliczPowierzchnie() {
return szerokosc * wysokosc;
}
}
```

KLASY I METODY FINALNE

Klasy, z których nie jest możliwe dziedziczenie nazywane są klasami finalnymi, co wyrażane jest za pomocą słowa kluczowego **final** występującego w definicji klasy:

```
public final class Procesor {
// ciało klasy (składowe klasy)
}
```

Użycie słowa kluczowego **final** w deklaracji metody powoduje, że nie jest możliwe jej przesłonięcie w klasach dziedziczących.

KLASY WEWNĘTRZNE

Możliwe jest umieszczenie definicji klasy w obrębie innej klasy. Konstrukcja ta stosowana jest w przypadku, gdy użycie klasy jest ograniczone i tworzenie instancji (obiektów) występuje wyłącznie w obrębie jednej klasy.
W przypadku, gdy utworzona klasa znajdzie wykorzystanie tylko w jednej, innej klasie, preferowane jest zdefiniowanie jej jako klasę wewnętrzną (możliwe jest ukrycie jej implementacji). Definicja klasy wewnętrznej musi znaleźć się w ciele innej klasy.

```
class Testowa {
// ciało klasy głównej (składowe klasy)

class KlasaWewnetrzna {
// ciało klasy wewnętrznej (składowe klasy)
}
}
```

Klasa wewnętrzna posiada dostęp do wszystkich składowych klasy w której została zdefiniowana. Nie ma również żadnych ograniczeń co do wykorzystania dziedziczenia czy też implementacji interfejsów. Szczególnym przypadkiem klas wewnętrznych są **klasy anonimowe** (nie posiadające nazwy).