PROGRAMOWANIE WSPÓŁBIEŻNE I RÓWNOLEGŁE

1. Programowanie obiektowe w języku Java

Java jest językiem, w którym realizowana jest idea programowania zorientowanego obiektowo (ang. *Object Oriented Programming*). Tworzenie aplikacji odbywa się z wykorzystaniem odpowiednich, współpracujących ze sobą elementów, zwanych obiektami, konstruowanych na podstawie wzorca zwanego klasą. Biblioteki Javy zawierają pokaźny zbiór gotowych do użycia klas, które ułatwiają tworzenie programów, jednakże złożone aplikacje składają się najczęściej z szeregu nowych klas, tworzonych przez programistę.

Klasy

W otaczającej rzeczywistości występuje szereg obiektów tego samego typu (np. zbiór samochodów, drzew, czy studentów). Klasa określana jest jako zbiór stanów oraz zachowań opisujących obiekty należące do tej samej kategorii. Jest to pewnego rodzaju wzorzec, na podstawie którego

tworzone są unikalne wersje obiektu (np. czerwony samochód marki Fiat, wysokie drzewo). W skład każdej klasy wchodzą:

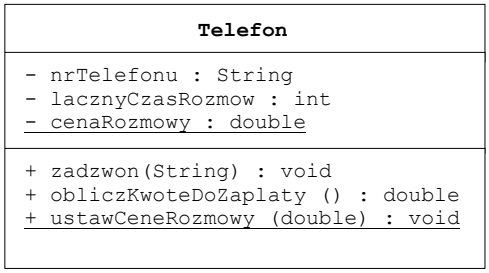
- **pola** (zmienne), zawierające informacje opisujące właściwości (stany) obiektów,
- **metody** (podprogramy), modelujące zachowania obiektów.

Obiekty

Na podstawie zdefiniowanej klasy tworzone są obiekty różniące się właściwościami. Każdy obiekt posiada własny zbiór pól, których wartości określają jego indywidualność. Na podstawie klasy możliwe jest zatem utworzenie dowolnej liczby obiektów (instancji klasy), należących do tej samej kategorii, zróżnicowanych pod względem właściwości.

Tworzenie klas

Proces definiowania klasy polega na określeniu jej składowych, tj. pól oraz metod, reprezentujących stany i zachowania obiektów, które będą tworzone w oparciu o tę klasę. Poniższy przykład ilustruje klasę `Telefon` stanowiącą wzorzec dla zbioru opisywanych telefonów. Każdy telefon posiada pewne unikalne cechy (np. numer telefonu, łączny czas rozmów) oraz zachowania, tj. czynności, które może wykonać (np. zadzwoń).



Definicja klasy w języku Java realizowana jest za pomocą słowa kluczowego `class`:

```
class Telefon {

  // ciało klasy (składowe klasy)

}
```

Każda tworzona klasa może zawierać zbiór pól oraz metod stanowiących składowe klasy.

Pola klasy

Pola to zmienne deklarowane wewnątrz klasy. W przypadku braku jawnej inicjalizacji zmiennej otrzymuje ona wartość domyślną[1]. Zwyczajowo deklaracja pól klasy występuje przed deklaracją metod. Poniższy kod programu zawiera definicję klasy `Telefon` wraz z wchodzącymi w jej skład polami.

```
class Telefon {

  // deklaracja pól klasy
  private String numerTelefonu;
```

[1] W przypadku braku jawnej inicjalizacji pól klasy, przyjmują one wartości domyślne (pola numeryczne wartości zerowe, pola logiczne wartość `false`, natomiast pola klasowe wartość `null`).

```
  private int lacznyCzasRozmow;
  private static double cenaRozmowy = 0.48; // zł/min.
}
```

Pola statyczne (oznaczone w kodzie modyfikatorem `static`), w przeciwieństwie do pól niestatycznych, są wspólne dla wszystkich obiektów danej klasy[2]. Dostęp do nich jest możliwy bez konieczności tworzenia obiektu klasy. W powyższym przykładzie pole `cenaRozmowy` dotyczy właściwości odnoszącej się do wszystkich obiektów klasy `Telefon`. Dostęp do pól klasy jest możliwy zgodnie z rodzajem użytego w deklaracji modyfikatora dostępu.

Metody klasy

Metody deklarowane w klasie stanowią odpowiednik zachowań konkretnych obiektów. Sposób deklaracji metod został omówiony w poprzednim module. Warto przypomnieć o metodach statycznych (klasowych), do których dostęp możliwy jest bez konieczności tworzenia obiektu danej klasy. Metody te nie mogą zatem odwoływać się do niestatycznych (instancyjnych) składowych klasy (pól i metod) powiązanych z konkretnym obiektem. Poniższy kod programu zawiera definicję klasy Telefon wraz z jej składowymi – polami oraz metodami.

```
class Telefon {

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

  public double obliczKwoteDoZaplaty() {
    return cenaRozmowy * (lacznyCzasRozmow / 60);
  }

  public static void ustawCeneRozmowy(double nowaCena){
    cenaRozmowy = nowaCena;
  }

  public void zadzwon(String nrTelefonu) {
    System.out.println("Dzwonię do: " + nrTelefonu);
    System.out.println("Dryń, dryń...");
    System.out.println("Rozmowa w toku...");
    int czasRozmowy = (int) (Math.random()*3600);
    lacznyCzasRozmow += czasRozmowy;
    System.out.println("Rozmowa zakończona. ");
    System.out.printf("Czas rozmowy: %d min. %d sek.", czasRozmowy/60,
                              czasRozmowy%60);
  }
}
```

Konstruktor

Konstruktor stanowi specyficzną metodę, która wywoływana zostaje w momencie tworzenia obiektu[3]. Jego nazwa musi być identyczna z nazwą klasy. Konstruktor nie może również zwracać żadnej wartości[4]. W skład konstruktora wchodzą instrukcje, które zostaną wykonane w trakcie tworzenia obiektu. Szczególnym przypadkiem jest tu nadanie wartości początkowej polom

[2] Pola statyczne nazywane są dość często polami klasowymi.

[3] Składnia tworzenia obiektu zawiera operator `new` oraz nazwę konstruktora, który ma zostać wywołany.

[4] Przed nazwą konstruktora nie może pojawić się również słowo kluczowe `void`.

obiektu, co może zostać zrealizowane albo w momencie deklaracji pola, albo też poprzez przypisanie wartości w konstruktorze obiektu.

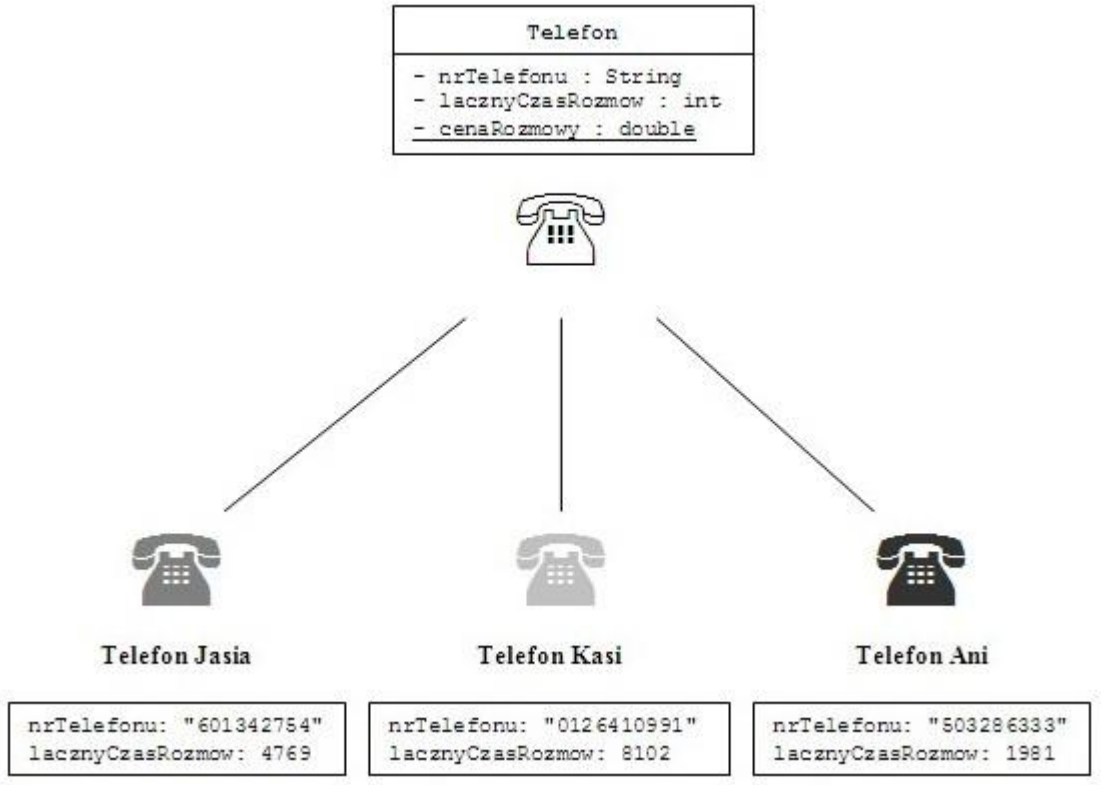
Poniższy kod programu zawiera deklarację konstruktora `Telefon(String)` dla klasy Telefon. Jego składową jest instrukcja przypisująca wartość początkową dla pola `numerTelefonu`.

```
public Telefon (String numer) {
  numerTelefonu = numer;
}
```

W przypadku braku jawnie zadeklarowanego konstruktora Java automatycznie wywołuje konstruktor domyślny[5]. Możliwe jest przeciążanie konstruktorów, podobnie jak innych metod zadeklarowanych w klasie.

Tworzenie obiektów

Definicja klasy udostępnia wzorzec, na podstawie którego tworzone są obiekty występujące w programie. Ich interakcja realizowana poprzez wywoływanie metod stanowi o treści tworzonej aplikacji. Poniższy rysunek przedstawia schemat tworzenia obiektów na podstawie klasy `Telefon`.



Tworzenie nowych obiektów w Javie polega na użyciu operatora `new`, po którym występuje nazwa odpowiedniego konstruktora. Poniższy kod zawiera przykład utworzenia obiektu klasy `Telefon`:

```
Telefon telefonKasi = new Telefon("0126410991");
```

Operator `new` tworzy na podstawie wzorca (klasy) nowy obiekt oraz umieszcza go w pamięci operacyjnej. Wartością zwracaną podczas tworzenia obiektu jest referencja do miejsca, w którym obiekt został utworzony. Wartość ta przypisana zostaje do zmiennej typu obiektowego, co pozwala na odwoływanie się do składowych obiektu:

[5] Konstruktor domyślny posiada pustą liczbę parametrów i jest wywoływany wyłącznie wtedy, gdy klasa nie posiada żadnego jawnie zadeklarowanego konstruktora.

```
public class WykorzystanieObiektow {
  public static void main(String args[]) {

    Telefon telefonJasia = new Telefon("601342754");

    // wywołanie metody zadzwon() na rzecz obiektu telefonJasia
    telefonJasia.zadzwon("687934267");
  }
}
```

Metody dostępowe i modyfikujące

Metody dostępowe (ang. *accessor (getter) methods*) są używane do odczytu wartości pól (instancyjnych i statycznych). Nazwa takiej metody zazwyczaj składa się z czasownika „pobierz" (ang. *get*) oraz nazwy pola.

Zadaniem metod modyfikujących (ang. *mutator (setter) methods*) jest nadanie bądź też zmiana wartości pól (instancyjnych i statycznych). Nazwa metody zawiera zwykle czasownik „ustaw" (ang. *set*) oraz nazwę pola. Metody modyfikujące z reguły nie zwracają żadnej wartości.

```
class Zeszyt{
  private int liczbaKartek;

  // metoda dostępowa
  public int pobierzLiczbeKartek() {
    return liczbaKartek;
  }

  // metoda modyfikująca
  public void ustawLiczbeKartek(int liczba) {
    liczbaKartek = liczba;
  }
}
```

Słowo kluczowe this

Słowo `this` oznacza referencję do bieżącego obiektu. Ułatwia to dostęp do jego składowych, jak również umożliwia wywołanie odpowiednich konstruktorów. Często `this` jest wykorzystywane, gdy nazwa parametru metody jest identyczna z nazwą pola[6], umożliwiając rozróżnienie tych identyfikatorów.

```
class Student {

  private String nazwisko;

  public void ustawNazwisko(String nazwisko) {
    // inicjalizacja pola "nazwisko" wartością parametru "nazwisko"
    this.nazwisko = nazwisko;
  }
}
```

Wywołanie konstruktora z ciała innego konstruktora jest możliwe poprzez użycie słowa kluczowego `this` wraz z ewentualną listą parametrów[7].

```
class Monitor {

  private String nazwa;
  private static int liczbaMonitorów;
```

[6] Występuje wtedy tzw. przesłanianie zmiennych.

[7] Wywołanie konstruktora przy użyciu `this` musi wystąpić na początku bloku instrukcji..

```
  public Monitor(){
    liczbaMonitorów++;
  }

  public Monitor(String nazwa) {
    this(); // wywołanie konstruktora bezparametrowego
    this.nazwa = nazwa; // inicjalizacja pola wartością parametru
  }
}
```

Określanie widoczności składowych klasy

Prawidłowe określenie widoczności poszczególnych składowych klasy umożliwia ukrycie implementacji. Zgodnie z zasadami programowania zorientowanego obiektowo dostęp do pól powinien być realizowany wyłącznie za pomocą metod dostępowych i modyfikujących (tworzących tzw. interfejs obiektu). Proces ukrywania składowych klasy nazywany jest enkapsulacją[8] i odbywa się poprzez użycie odpowiedniego modyfikatora dostępu (zazwyczaj `private`).

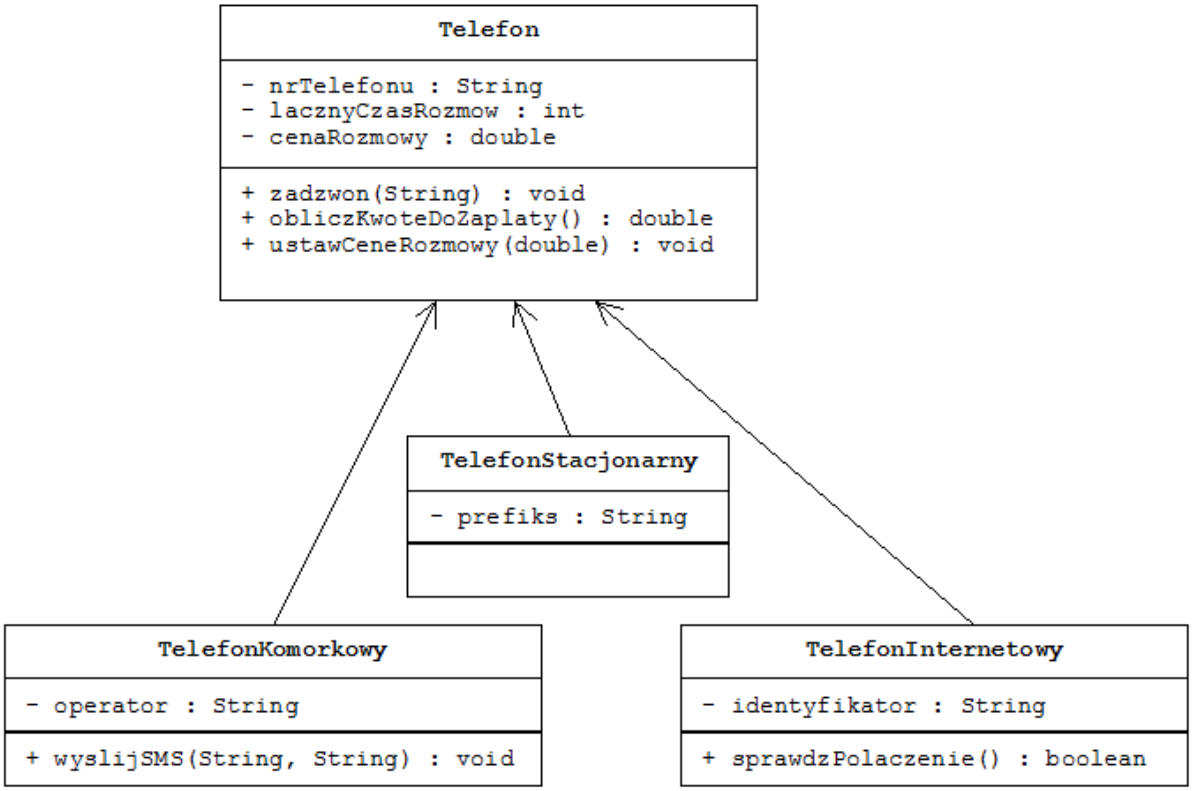
```
class Osoba{

  // pola prywatne – dostęp możliwy tylko za pośrednictwem odpowiednich metod
  private String imie;
  private String nazwisko;
  private double pensja;

  // konstruktor
  public Osoba(String imie, String nazwisko, double pensja) {

    this.imie = imie;
    this.nazwisko = nazwisko;
    this.pensja = pensja;
  }

  // publiczna metoda umożliwiająca zmianę pensji danej osoby
  public void ustawPensje(double nowaPensja) {
    pensja = nowaPensja;
  }
}
```

Dziedziczenie

W celu określenia, iż jedno pojęcie jest uszczegółowieniem (lub uogólnieniem) innego, stosowane jest dziedziczenie. Wyraża ono relację „jest" i jest wykorzystywane wszędzie tam, gdzie należy wskazać, iż dwa pojęcia są do siebie podobne. Realizacja dziedziczenia polega na tym, iż nowo tworzona klasa przejmuje wszelkie cechy i zachowania z klas już istniejących, dodając bądź też modyfikując je, aby były one bardziej wyspecjalizowane. Dzięki temu powstaje nowa klasa określana terminem „podklasa" (ang. *subclass*) lub „klasa pochodna", natomiast klasa, z której dokonano dziedziczenia to „nadklasa" (ang. *superclass*) [9] lub „klasa bazowa".

[8] Enkapsulacja często nazywana jest również hermetyzacją danych.

[9] Używane są również pojęcia dziecka (ang. *child*) i rodzica (ang. *parent*) dla określania nazwy podklasy oraz klasy bazowej.



Dziedziczenie w języku Java realizowane jest za pomocą słowa kluczowego `extends`[10] po którym występuje nazwa klasy, z której dziedziczone są jej składowe:

```
class Student extends Osoba {
  // ciało klasy (składowe klasy)
}
```

Klasa `Student` rozszerza (dziedziczy) klasę `Osoba`. W tym przypadku `Osoba` jest klasą bazową, natomiast `Student` klasą pochodną, która dziedziczy wszystkie dostępne pola i metody klasy bazowej[11]. W klasie pochodnej możliwe jest:

- deklarowanie nowych pól, które nie występują w klasie bazowej oraz tworzenie nowych pól o takiej samej nazwie, ukrywając zarazem pola oryginalne z klasy bazowej[12],
- deklarowanie metod, które nie występują w klasie bazowej oraz tworzenie nowych metod o takiej samej sygnaturze[13].

Rozszerzając możliwości przykładowej klasy `Telefon` można na jej podstawie utworzyć nowe klasy: `TelefonKomorkowy`, `TelefonStacjonarny` oraz `TelefonInternetowy`. Klasy te posiadają nowe cechy (np. operator, prefiks), jak i zachowania (np. wyślij SMS). Każda z klas posiada również własną implementację metody `zadzwon(String)`. Tworząc nowe obiekty można posługiwać się również typem klasy bazowej (zarówno telefon stacjonarny jak i komórkowy jest przecież telefonem[14]).

```
class TelefonKomorkowy extends Telefon {
```

[10] Dziedziczenie w Javie jest jednokrotne (każda klasa dziedziczy tylko z jednej klasy). Klasy, które jawnie nie posiadają zadeklarowanego dziedziczenia (brak słowa kluczowego extends w nagłówku) domyślnie dziedziczą z klasy Object.
[11] Konstruktory nie są dziedziczone.
[12] Tworzenie w klasie pochodnej pól o takiej samej nazwie jak w klasie bazowej nie jest zalecane.
[13] Tworzenie w klasie pochodnej metod o takiej samej sygnaturze jak metody w klasie bazowej nazywane jest przesłanianiem metod (ang. override).
[14] Taki sposób tworzenia obiektów nazywany jest rzutowaniem w górę (ang. upcasting).

```
  // nowe pole
  String operator;

  // konstruktor
  public TelefonKomorkowy (String numer, String operator) {
    super(numer); // wywołanie konstruktora klasy bazowej
    this.operator = operator;
  }

  // nowa metoda
  public void wyslijSMS(String nrTelefonu, String tresc) {
    System.out.println ("Wysylam SMS'a o tresci: " + tresc);
    System.out.println ("pod numer: " + nrTelefonu);
  }

  // nowa implementacja metody zadzwon (metoda przesłonięta)
  public void zadzwon(String nrTelefonu) {
    System.out.println ("Dzwonie z komórki do: " + nrTelefonu);
  }
}

class TelefonStacjonarny extends Telefon {

  // nowe pole
  String prefiks;

  // konstruktor
  public TelefonStacjonarny (String numer, String prefiks) {
    super(numer);
    this.prefiks = prefiks;
  }

  // nowa implementacja metody zadzwon (metoda przesłonięta)
  public void zadzwon(String nrTelefonu) {
    System.out.println ("Dzwonie ze stacjonarnego do: " + nrTelefonu);
  }
}

class TelefonInternetowy extends Telefon {

  // nowe pole
  String identyfikator;

  // konstruktor
  public TelefonInternetowy (String numer, String identyfikator) {
    super(numer);
    this.identyfikator = identyfikator;
  }

  // nowa metoda
  public boolean sprawdzPolaczenie() {
    return true;  }

  // nowa implementacja metody zadzwon (metoda przesłonięta)
  public void zadzwon(String nrTelefonu) {
    if(sprawdzPolaczenie())
      System.out.println ("Dzwonie z internetowego do: " + nrTelefonu)
    else
      System.out.println ("Brak polaczenia z internetem");
  }
}

public class Test {

  public static void main(String args[]) {

    Telefon telefonJasia = new Telefon("867312632");
    Telefon komorkaMarka = new TelefonKomorkowy("606345956", "Era");
```

```
    Telefon stacjonarnyKasi = new TelefonStacjonarny("125439876", "1044");
    Telefon internetowyAni = new TelefonInternetowy("146743253", "Anula7");

  }
}
```

Polimorfizm

Zarówno kompozycja, jak i dziedziczenie, które pozwala kojarzyć klasy obiektów w hierarchie klas, należą do fundamentalnych własności podejścia obiektowego. Umożliwiają realizację idei programowania przyrostowego oraz późniejszą szybką modyfikację kodu programu.

Tworząc obiekty na podstawie klas dziedziczących z innych klas, można jako zmienną referencyjną podać typ klasy bazowej[15]. Wywołując następnie odpowiednie metody kompilator Javy decyduje, która z nich ma zostać wykonana (na podstawie typu obiektu, na rzecz którego są one wywołane). Taki sposób wywołania metod nazywany jest polimorfizmem.

Poniższy przykład przedstawia tablicę elementów klasy `Telefon`. Poszczególne elementy tablicy zawierają referencje zarówno do obiektów klasy `Telefon`, jak również do obiektów utworzonych na podstawie klas pochodnych (`TelefonKomorkowy`, `TelefonStacjonarny`).

```
Telefon[] tablicaTelefonow = new Telefon[4];

tablicaTelefonow[0] = new Telefon("634295432");
tablicaTelefonow[1] = new TelefonKomorkowy("504295432", "Orange");
tablicaTelefonow[2] = new TelefonStacjonarny("126493042", "1033");
tablicaTelefonow[3] = new TelefonInternetowy("325649344", "lech23");
```

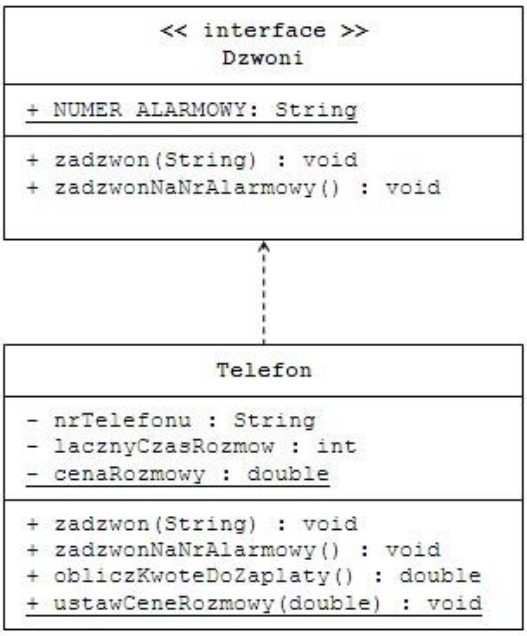
Wywołanie dla każdego obiektu znajdującego się w tablicy `tablicaTelefonow` metody `zadzwon("112")` będzie wywołaniem polimorficznym. W zależności od klasy obiektu wywołana zostanie odpowiednia metoda.

```
for (Telefon tel : tablicaTelefonow)
  tel.zadzwon("112");
```

Interfejsy

Interfejsy w programowaniu obiektowym stanowią zbiór wymagań dotyczących klas, które będą go stosować. Wyrażają sposób opisu funkcjonalności klasy bez określania, w jaki sposób będzie to uzyskane. W skład interfejsów wchodzą metody oraz pola. Deklaracja metod składa się z nagłówków (brak jest ciała metody), natomiast pola interfejsu to wyłącznie stałe statyczne z jawnie określoną wartością. W deklaracjach wszystkich składowych niezbędne jest użycie modyfikatora `public`.

[15] Można użyć również nazwy interfejsu, który dana klasa implementuje.



Klasy wykorzystujące interfejs nie rozszerzają jego możliwości, ale go implementują. Oznacza to, że klasa musi zawierać definicję wszystkich metod (ciała tych metod) zadeklarowanych w interfejsie. Składnia interfejsu zbliżona jest do deklaracji klasy. W miejsce `class` stosowane jest słowo kluczowe `interface`[16].

```
interface Dzwoni {
  public static String NUMER_ALARMOWY = "112";

  public void zadzwon(String);
  public void zadzwonNaNrAlarmowy();
}
```

Implementacja interfejsu polega na umieszczeniu słowa kluczowego `implements` w nagłówku deklaracji klasy, a następnie wymienieniu nazwy interfejsu, który klasa implementuje[17]. Składowymi klasy stają się wszelkie pola i metody występujące w definicji klasy oraz pola i metody określone w deklaracji interfejsu. Poniższy kod programu ilustruje implementację interfejsu `Dzwoni`.

```
class Telefon implements Dzwoni {

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

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

  // deklaracja metod
  public double obliczKwoteDoZaplaty() {
    return cenaRozmowy * (lacznyCzasRozmow / 60);
  }

  public static void ustawCeneRozmowy(double nowaCena){
```

[16] Dopuszczalne jest dziedziczenie interfejsów.
[17] Możliwa jest implementacja dowolnej liczby interfejsów przez pojedynczą klasę co pozwala na realizację tzw. dziedziczenia wielobazowego.

```
    cenaRozmowy = nowaCena;
  }

  // metody wymagane przez interfejs
  public void zadzwon(String nrTelefonu) {
    System.out.println("Dzwonię do: " + nrTelefonu);
    System.out.println("Dryń, dryń...");
    System.out.println("Rozmowa w toku...");
    int czasRozmowy = (int) (Math.random()*3600);
    lacznyCzasRozmow += czasRozmowy;
    System.out.println("Rozmowa zakończona. ");
    System.out.printf("Czas rozmowy: %d min. %d sek.",
                      czasRozmowy/60, czasRozmowy%60);
  }

  public void zadzwonNaNrAlarmowy() {
    System.out.println("Dzwonię do: " + Dzwoni.NUMER_ALARMOWY);
    System.out.println("Dryń, dryń...");
    System.out.println("Centrum pomocy, słucham");
  }
}

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 stosowane, jako typ danych przy deklaracji zmiennej[18]. Jej wartość stanowi odwołanie do obiektu dowolnej klasy, która implementuje interfejs[19].

Zadania

1. Sprawdzanie wydajności aplikacji wiąże się z pomiarem czasu jej wykonania. Zaprojektuj klasę `Stoper`. Jej głównym zadaniem będzie pomiar czasu pomiędzy uruchomieniem stopera, a jego zatrzymaniem. Zastanów się, jakie pola i metody będą potrzebne do realizacji tego zadania. Wykorzystaj metodę `currentTimeMillis()` z klasy `System`.

2. W zaawansowanych programach często istnieje konieczność dokonania zliczania liczby wykonywanych czynności. Zdefiniuj klasę `Licznik` z jednym polem prywatnym `ilosc`. Dopisz metodę dostępową zwracającą wartość pola `ilosc` oraz modyfikującą, której zadaniem będzie inkrementacja pola `ilosc`. Następnie napisz program, który wykorzysta utworzoną klasę do zliczania liczby znaków spacji w podanym przez użytkownika zdaniu.

3. Znajdź obiekt występujący w rzeczywistości i stwórz dla niego odpowiednią implementację w Javie (klasa plus przykładowe obiekty).

4. Dostępna jest lista studentów, którą należy uporządkować według numeru albumu. Napisz program, który zrealizuje opisaną operację. Utwórz klasę `Student` zawierającą imię, nazwisko oraz numer albumu studenta. Klasa ta powinna również implementować interfejs `Comparable` (zapoznaj się z dokumentacją). Do sortowania tablicy zawierającej studentów użyj metody `sort()` z klasy `java.util.Arrays`. Wyświetl wyniki na konsoli. Poniżej przedstawiona została przykładowa lista studentów

[18] Jest to tak zwane rzutowanie rozszerzające.

[19] Odwołania do metod, które nie występują w interfejsie możliwe są za pomocą odpowiedniego rzutowania (tzw. rzutowanie zawężające)

```
Student[] lista = new Student[4];

lista[0] = new Student("Jan", "Kowalski", 432187);
lista[1] = new Student("Adam", "Nowak", 332132);
lista[2] = new Student("Joanna", "Wyszek", 632165);
lista[3] = new Student("Ania", "Nowak", 321419);
```

oraz uzyskane wyniki.

```
Student: Ania Nowak     nr albumu: 321419
Student: Adam Nowak     nr albumu: 332132
Student: Jan Kowalski   nr albumu: 432187
Student: Joanna Wyszek  nr albumu: 632165
```

5. Zmodyfikuj kod programu porządkującego studentów wg numeru albumu w taki sposób, aby sortowanie odbywało się jednocześnie wg nazwiska, imienia, oraz numeru albumu. Przykładowe rezultaty odnoszące się do danych zawartych w poprzednim zadaniu zostały przedstawione poniżej.

```
Student: Jan Kowalski   nr albumu: 432187
Student: Adam Nowak     nr albumu: 332132
Student: Ania Nowak     nr albumu: 321419
Student: Joanna Wyszek  nr albumu: 632165
```