Rozdział 4

KLASY, OBIEKTY, METODY

Java jest językiem w pełni zorientowanym obiektowo. Wszystkie elementy opisujące dane, za wyjątkiem zmiennych prostych są obiektami. Sam program też jest obiektem pewnej klasy.

Aby utworzyć obiekt należy najpierw zdefiniować klasę, która jest wzorem do tworzenia obiektów tej klasy.

Klasa definiuje zarówno dane jak i algorytmy służące do ich przetwarzania. Dane są zapisywane w klasie w postaci pól (zmiennych lub stałych składowych), a algorytmy w postaci metod. Metodą jest wydzielony fragment programu, na ogół zawierający parametry, który może operować na ściśle określonych danych.

Aby utworzyć klasę, należy użyć słowa kluczowego **class** wykorzystując poniższy schemat (tekst następujący w wierszu po dwóch znakach // jest komentarzem):

```
class nazwaKlasy {

   // deklaracja pól
   typ zmiennaPierwsza;
   typ zmiennaDruga;
   ...

   // deklaracja metod
   typ metodaPierwsza(parametry)
```

```
   {
      // ciało metody
   }

   typ metodaDruga(parametry)
   {
      // ciało metody
   }
   ...
}
```

Dla przykładu rozważmy klasę pozwalającą tworzyć obiekty zawierające informacje o paczkach, które na przykład trzeba przewieźć ciężarówką. Paczek jest więcej niż może pomieścić ciężarówka i chodzi o to, by załadować na ciężarówkę jak najwięcej paczek tak, aby suma wartości tych paczek była maksymalna. Dla każdej paczki musimy zatem dysponować następującymi danymi: długość podstawy, szerokość podstawy, wysokość i oczywiście wartość paczki.

Pierwsza wersja klasy Paczka (bez projektowania metod) może wyglądać następująco (typ **int** jest typem całkowitym omówionym w rozdziale 5):

```
class Paczka {

   int dlugosc;
   int szerokosc;
   int wysokosc;
   int wartosc;
}
```

Obiekt klasy Paczka możemy utworzyć w dwóch etapach. Najpierw trzeba zadeklarować referencję do obiektu klasy Paczka:

```
Paczka telewizor;
```

Referencja czyli inaczej odnośnik identyfikuje dany obiekt. Możemy sobie wyobrazić, że referencja podaje między innymi adres w pamięci, gdzie jest umieszczony obiekt, jego wielkość itp. Po wykonaniu powyższej instrukcji zmienna `telewizor` posiada wartość `null`, czyli nie zawiera referencji do żadnego obiektu.

Aby utworzyć nowy obiekt, należy wykorzystać operator **new**:

```
telewizor = new Paczka();
```

Operator **new** tworzy nowy obiekt klasy Paczka, a referencja do niego jest przypisywana zmiennej `telewizor` (przy pomocy operatora =). Od tego momentu zmienna `telewizor` nie posiada już wartości `null`, lecz odnosi się do pewnego obiektu klasy Paczka.

Obie powyższe instrukcje można połączyć i utworzyć obiekt klasy Paczka w jednym etapie:

```
Paczka telewizor = new Paczka();
```

Każdy obiekt posiada własną kopię pól zadeklarowanych w klasie. Dostęp do tych pól można uzyskać przy pomocy operatora ".". (kropka). Ustawienie wartości wszystkich pól dla obiektu, do którego referencją jest zmienna `telewizor` wygląda następująco:

```
telewizor.dlugosc = 150;
telewizor.szerokosc = 80;
telewizor.wysokosc = 120;
telewizor.wartosc = 6000;
```

Obiektów danej klasy można utworzyć dowolnie dużo. Dla przykładu utworzymy jeszcze jeden obiekt klasy Paczka:

```
Paczka pralka = new Paczka();
```

Wartości pól obiektu, do którego referencją jest zmienna `pralka`, ustawiamy analogicznie jak poprzednio:

```
pralka.dlugosc = 70;
pralka.szerokosc = 50;
pralka.wysokosc = 130;
pralka.wartosc = 5000;
```

Jeszcze raz podkreślmy, że każdy obiekt klasy Paczka ma swoją własną kopię pól zdefiniowanych w klasie. Ilustruje to rys. 4.1.

Obliczenie objętości paczki - obiektu, do którego referencją jest zmienna `telewizor` można dokonać przy pomocy instrukcji:

```
obj = telewizor.dlugosc * telewizor.szerokosc
* telewizor.wysokosc;
```

Rysunek 4.1: Obiekty klasy Paczka, do których referencjami są `telewizor` i `pralka`

Każde pole i metoda zdefiniowane w klasie może mieć wyszczególniony specyfikator dostępu określany przez słowa: **public**, **private** oraz **protected**. Umieszczenie słowa **public** oznacza, że dany składnik jest dostępny w całym programie. Słowo **private** oznacza, że dany składnik może być wykorzystywany tylko przez metody danej klasy. Specyfikator dostępu określony przez słowo **protected** jest omówiony w rozdziale 13 i na razie nie będziemy się nim zajmować.

Istnieje jeszcze specyfikator domyślny występujący wtedy, gdy nie używamy żadnego słowa (pola zdefiniowane w dotychczasowej wersji klasy Paczka miały właśnie domyślny specyfikator dostępu). Domyślny specyfikator ma znaczenie takie samo jak specyfikator **public** dla klas zdefiniowanych w tym samym pakiecie. Ponieważ pakietami zajmiemy się dopiero w rozdziale 16, na razie przyjmiemy, że specyfikator domyślny jest równoważny specyfikatorowi **public**.

Zmieńmy teraz definicję klasy Paczka w sposób następujący:

```
class Paczka {

   private int dlugosc;
   private int szerokosc;
   private int wysokosc;
   private int wartosc;
}
```

W tym przypadku nie ma już możliwości wykorzystywania w programie instrukcji rodzaju:

```
   telewizor.dlugosc = 150;
```

czy też

```
obj = telewizor.dlugosc * telewizor.szerokosc
* telewizor.wysokosc;
```

Pola ***dlugosc***, ***szerokosc***, ***wysokosc*** i ***wartosc*** są niedostępne na zewnątrz klasy. I tak w większości przypadków powinno być. Dostęp do tych pól czy też wykonywanie na nich operacji powinno odbywać się przy pomocy metod.

Ogólny schemat dla utworzenia metody jest następujący:

```
typ nazwa(parametry)
{
   // ciało metody
}
```

Dla przykładu utwórzmy teraz metodę pozwalającą na obliczenie objętości paczki.

```
public int ObliczObj()
{
   return dlugosc * szerokosc * wysokosc;
}
```

Metoda ta powinna być umieszczona wewnątrz definicji klasy Paczka:

```
class Paczka {

   private int dlugosc;
   private int szerokosc;
   private int wysokosc;
   private int wartosc;

   public int ObliczObj()
   {
      return dlugosc * szerokosc * wysokosc;
   }
}
```

Metoda ObliczObj() nie zawiera żadnych parametrów i działa na polach zdefiniowanych w klasie Paczka. Mimo że są to pola ze specyfikatorem

private metoda obliczObj() ma oczywiście do nich dostęp, ponieważ sama jest składnikiem klasy Paczka. Zwróćmy uwagę na słowo kluczowe **return** oznaczające powrót do instrukcji, w której dana metoda została wywołana wraz z podaniem obliczonej wartości do nazwy funkcji.

Obecnie zajmiemy się szczególną metodę zwaną konstruktorem. Konstruktor jest wywoływany zawsze w momencie tworzenia obiektu. Jeżeli konstruktor nie został zdefiniowany, to wtedy jest wywoływany konstruktor domyślny - bezparametrowy przypisujący wszystkim polom wartość 0. Jeżeli w klasie jest zdefiniowany konstruktor z parametrami, to wtedy konstruktor domyślny nie jest tworzony automatycznie. W razie konieczności wykorzystywania konstruktora bez parametrów trzeba go jawnie zdefiniować.

Konstruktor ma zawsze nazwę taką samą jak nazwa klasy i w jego definicji nie wolno podać typu. Na przykład, zaprojektujmy konstruktor dla klasy Paczka:

```
public Paczka(int aDl,int aSzer,int aWys,int aWart)
{
   dlugosc = aDl;
   szerokosc = aSzer;
   wysokosc = aWys;
   wartosc = aWart;
}
```

Konstruktor ten ma cztery parametry, nazywane często parametrami formalnymi, których wartości służą do przypisania wartości polom obiektu.

Tak jak dla innych metod definicję konstruktora umieszczamy wewnątrz definicji klasy:

```
class Paczka {

   private int dlugosc;
   private int szerokosc;
   private int wysokosc;
   private int wartosc;

   public Paczka(int aDl,int aSzer,int aWys,int aWart)
   {
      dlugosc = aDl;
```

```
      szerokosc = aSzer;
      wysokosc = aWys;
      wartosc = aWart;
   }

   public int ObliczObj()
   {
      return dlugosc * szerokosc * wysokosc;
   }
}
```

Obiekty klasy Paczka możemy teraz tworzyć przy pomocy instrukcji:

```
Paczka telewizor = new Paczka(150,80,120,6000);
```

Utworzony obiekt klasy Paczka, do którego referencją jest zmienna `telewizor`, ma swój zestaw pól o przypisanych wartościach `150,80,120,6000`. Zwróćmy uwagę, że parametry podane w momencie wywołania metody - w tym przypadku konstruktora, są nazywane parametrami aktualnymi. Trzeba zapamiętać, że w języku Java parametry są przekazywane zawsze przez wartość, co oznacza, że wartość parametru aktualnego jest przekazywana do parametru formalnego. Oznacza to, że nie ma możliwości zmiany parametru aktualnego poprzez wykonanie metody. Do tego zagadnienia powrócimy jeszcze w następnych rozdziałach, a teraz zastanówmy się, jak obliczyć objętość paczki - obiektu, do którego referencję zawiera zmienna `telewizor`. Otóż wystarczy w tym celu wywołać metodę obliczObj() dla odpowiedniego obiektu klasy Paczka, wykorzystując operator kropki. Ilustruje to poniższa instrukcja:

```
int obj1 = telewizor.ObliczObj();
```

Metody z danej klasy czyli w naszym przypadku klasy Paczka mogą być wywoływane wyłącznie dla obiektów tej klasy. A zatem na przykład, jeżeli obiekt o nazwie ***inny*** jest obiektem innej klasy niż Paczka, to poniższa instrukcja spowoduje błąd kompilacji:

```
int obj1 = innny.ObliczObj();    // BŁĄD KOMPILACJI
```

Zasygnalizujmy jeszcze skąd metoda obliczObj() wie dla jakiego obiektu, czyli na jakich wartościach, ma wykonać obliczenia. Jak wiadomo, obiektów klasy Paczka może być dowolnie dużo i metoda obliczObj()

może obliczyć objętość każdego z tych obiektów. W momencie wywołania metody obliczObj() jest do niej przekazywana referencja do obiektu, na rzecz którego nastąpiło wywołanie. W naszym przypadku referencją tą jest `telewizor` i dla tego obiektu nastąpią obliczenia. Referencję tę można do metody przekazać w sposób jawny przy pomocy słowa kluczowego **this**, co wyjątkowo może być przydatne. Zostanie to omówione w dalszej części książki w rozdziale 11.

Przeanalizujmy teraz ostatnią wersję definicji klasy Paczka. Ponieważ pola klasy są deklarowane ze specyfikatorem **private**, nie ma możliwości wykonywania na nich operacji na zewnątrz klasy. I bardzo dobrze. Do nadania wartości początkowych polom obiektu klasy Paczka służy konstruktor, a do obliczenia objętości metoda obliczObj(). Może być jednak potrzebne wyprowadzenie rozmiarów obiektu - paczki czy też jej wartości. W tym celu można utworzyć metody deklarowane ze specyfikatorem **public**, co oznacza, że można je wykonać w całym programie. Jedna z tych metod ma postać:

```
public int PodajDlugosc()
{
   return dlugosc;
}
```

Zadaniem metody podajDlugosc() jest tylko zwrócenie wartości odpowiedniego pola klasy.

Można się teraz zastanowić, czy warto było pola klasy deklarować ze specyfikatorem **private**. Czy nie lepiej po prostu zezwolić na dostęp do nich w całym programie. Otóż na pewno warto. Dzięki temu przypisanie wartości tym polom odbywa się wyłącznie w konstruktorze i przy szukaniu błędu w programie nie musimy sprawdzać, czy gdzieś w programie nie nastąpiła przypadkowa zmiana wartości któregoś z pól. Ponadto przykładowa metoda podajDlugosc() jest najprostsza z możliwych i łatwo można dokonać jej zmiany tak, by zwracała zmodyfikowaną wartość pola (na przykład można przeliczyć centymetry na metry).

Ostateczna wersja definicji klasy Paczka jest zatem następująca:

```
class Paczka {

   private int dlugosc;
   private int szerokosc;
```

```
private int wysokosc;
private int wartosc;

public Paczka(int aDl,int aSzer,int aWys,int aWart)
{
   dlugosc = aDl;
   szerokosc = aSzer;
   wysokosc = aWys;
   wartosc = aWart;
}

public int PodajDlugosc()
{
   return dlugosc;
}

public int PodajSzerokosc()
{
   return szerokosc;
}

public int PodajWysokosc()
{
   return wysokosc;
}

public int PodajWartosc()
{
   return wartosc;
}

public int ObliczObj()
{
   return dlugosc * szerokosc * wysokosc;
}

public int ObliczPodst()
{
   return dlugosc * szerokosc;
```

```
   }
}
```

Zauważmy, że w definicji klasy Paczka dodaliśmy jeszcze metodę oblicz-Podst() pozwalającą obliczyć pole powierzchni podstawy paczki, co może się przydać przy upakowywaniu paczek.

Po tym wstępie możemy wreszcie podać jak napisać pierwszy program w języku Java. Program jest również obiektem pewnej klasy, która musi być zdefiniowana ze słowem **public**. Koniecznie trzeba zapamiętać, że nazwa pliku, w którym jest umieszczona ta klasa, musi być taka sama jak nazwa klasy, uwzględniając również duże litery. Wykonanie programu rozpoczyna się od metody o nazwie main() typu **void** (co oznacza, że nie zwraca ona żadnej wartości) zadeklarowanej ze specyfikatorami **public** i **static**.

Specyfikator **static** zostanie dokładnie omówiony w rozdziale 11, a teraz musi nam wystarczyć informacja, że składniki zadeklarowane ze słowem **static** są związane z klasą, a nie z poszczególnymi obiektami klasy. Jeżeli składnik jest zadeklarowany ze słowem **static**, to możemy mieć do niego dostęp bez utworzenia żadnego obiektu tej klasy (podajemy nazwę klasy i po kropce nazwę składnika). Wszystkie pola statyczne mają taką samą wartość dla wszystkich utworzonych obiektów danej klasy. Ponadto metody statyczne mogą wykorzystywać wyłącznie składniki statyczne (pola i metody).

Metoda main() posiada parametry, które są tablicą typu **String**. Wykorzystanie tych parametrów omówimy bliżej w rozdziale 9.

Schemat, według którego możemy utworzyć program, może być następujący:

```
public
class Pierwszy {

   public static void main(String[] args)
   {
      new Pierwszy(); // utworzenie obiektu klasy Pierwszy
   }

   public Pierwszy()  // konstruktor dla klasy Pierwszy
   {
      // instrukcje
```

```
   }
}
```

W metodzie main() tworzymy wyłącznie obiekt klasy Pierwszy. Obiekt ten jest naszym programem. W momencie tworzenia obiektu zawsze jest wywoływany konstruktor, który w tym przypadku istnieje i oczywiście ma nazwę Pierwszy(). Referencji do obiektu klasy Pierwszy nie zapamiętujemy w żadnej zmiennej, po prostu nie jest to do niczego potrzebne. Natomiast wszystkie niezbędne instrukcje zapisujemy w konstruktorze.

A oto nasz pierwszy program skonstruowany zgodnie z powyższym schematem:

Program 4.1

```
public
class Pierwszy {

   public static void main(String[] args)
   {
      new Pierwszy();
   }

   public Pierwszy()
   {
      Paczka telewizor = new Paczka(150,80,120,6000);
      int objetosc1 = telewizor.ObliczObj();

      System.out.println("Objętość paczki o rozmiarach: " +
      '\n' + telewizor.PodajDlugosc()+ " , "+
      telewizor.PodajSzerokosc() + " ,  " +
      telewizor.PodajWysokosc() + '\n' + "wynosi: " + '\n' +
      objetosc1);
   }
}
```

Zadaniem tego prostego programu jest utworzenie obiektu klasy Paczka, obliczenie objętości tego obiektu i wyprowadzenie niezbędnej informacji. Zwróćmy uwagę, że do wyprowadzenia informacji można wykorzystać instrukcję

```
   System.out.println(par);
```

gdzie parametr `par` może być typu **String** (typ **int** jest automatycznie transformowany do typu **String**). Więcej informacji o klasie String zawiera następny rozdział. Teraz zauważmy tylko, że operator + służy do łączenia obiektów klasy String, a znak '$\backslash n$' umożliwia wyprowadzenie znaku nowej linii.

Zauważmy jeszcze, że program w języku Java można zaprojektować, wykorzystując tylko metodę main() bez konstruktora klasy. Ilustruje to poniższy schemat:

```
public
class Drugi {

   public static void main(String[] args)
   {
      // instrukcje
   }
}
```

Program 4.2, skonstruowany zgodnie z powyższym schematem, wykonuje analogiczne operacje jak program 4.1:

Program 4.2

```
public
class Drugi {

   public static void main(String[] args)
   {
      Paczka telewizor = new Paczka(150,80,120,6000);
      int objetosc1 = telewizor.obliczObj();

      System.out.println("Objętość paczki o rozmiarach: ");
      System.out.println(telewizor.PodajDlugosc()+ " , "+
      telewizor.PodajSzerokosc() + " ,  " +
      telewizor.PodajWysokosc() + '\n' + "wynosi: ");
      System.out.println(objetosc1);
   }
}
```

Rozwiązanie pierwsze będziemy stosować częściej, ponieważ po pierwsze jasno stwierdza, że program jest też obiektem, a po drugie umożliwia wywoływanie w konstruktorze klasy metod, które nie są statyczne. W metodzie main(), która jest statyczna, można wywoływać tylko metody statyczne. Aby wykorzystać metodę, która nie jest statyczna, należy utworzyć obiekt danej klasy i dopiero dla tego obiektu wywołać metodę. Zostanie to dokładniej zilustrowane w rozdziale 9. Natomiast rozwiązanie drugie można zastosować do przetestowania klasy, którą będziemy później wykorzystywać w innych aplikacjach. Niezbędne instrukcje testujące zamieszczamy wtedy w metodzie main().

Więcej przykładów ilustrujących pracę z metodami i przekazywanie parametrów zawiera rozdział 9, a obecnie zwrócimy jeszcze uwagę na możliwość przypisywania referencji. Wyobraźmy sobie, że mamy zadeklarowane trzy zmienne referencyjne do obiektów klasy Paczka: `ref1`, `ref2`, `ref3`. Można tego dokonać przy pomocy instrukcji:

```
Paczka ref1, ref2, ref3;
```

Jak już wiemy, w tej chwili zmienne te nie odnoszą się do żadnego obiektu, a ich wartość jest `null`.

Utwórzmy teraz dwa obiekty klasy Paczka, do których referencję będą zawierały zmienne `ref1` i `ref2`:

```
ref1 = new Paczka(50,50,10,100);
ref2 = new Paczka(100,100,40,500);
```

Zmienna referencyjna `ref3` w dalszym ciągu posiada wartość `null`. W języku Java można dokonać następującego przypisania:

```
ref3 = ref1;
```

Oznacza to, że zmienna `ref3` zawiera referencję do tego samego obiektu, do którego odnosiła się zmienna `ref1`.

Zmiennej `ref3` można następnie przypisać wartość innej zmiennej referencyjnej, co ilustruje poniższa instrukcja.

```
ref3 = ref2;
```

Program 4.3 ilustruje możliwości stwarzane przez przypisywanie referencji.

Program 4.3

```
public
class Trzeci {

   public static void main(String[] args)
   {
      new Trzeci();
   }

   public Trzeci()
   {
      Paczka ref1, ref2, ref3;

      // utworzenie dwóch obiektów klasy Paczka
      ref1 = new Paczka(50,50,10,100);
      ref2 = new Paczka(100,100,40,500);
      ref3 = ref1;

      System.out.println("Objętość paczki o rozmiarach: " +
      ref3.PodajDlugosc()+ " , "+
      ref3.PodajSzerokosc() + " ,  " +
      ref3.PodajWysokosc() + '\n' + "wynosi: " +
      ref3.ObliczObj());
      ref3 = ref2;

      System.out.println("Objętość paczki o rozmiarach: " +
      ref3.PodajDlugosc()+ " , "+
      ref3.PodajSzerokosc() + " ,  " +
      ref3.PodajWysokosc() + '\n' + "wynosi: " +
      ref3.ObliczObj());
   }
}
```

I ostatnia już informacja w tym rozdziale. W języku Java nie ma konieczności dbania o usuwanie obiektów już niepotrzebnych. System sam usunie wszystkie obiekty, do których nie istnieje żadna referencja.