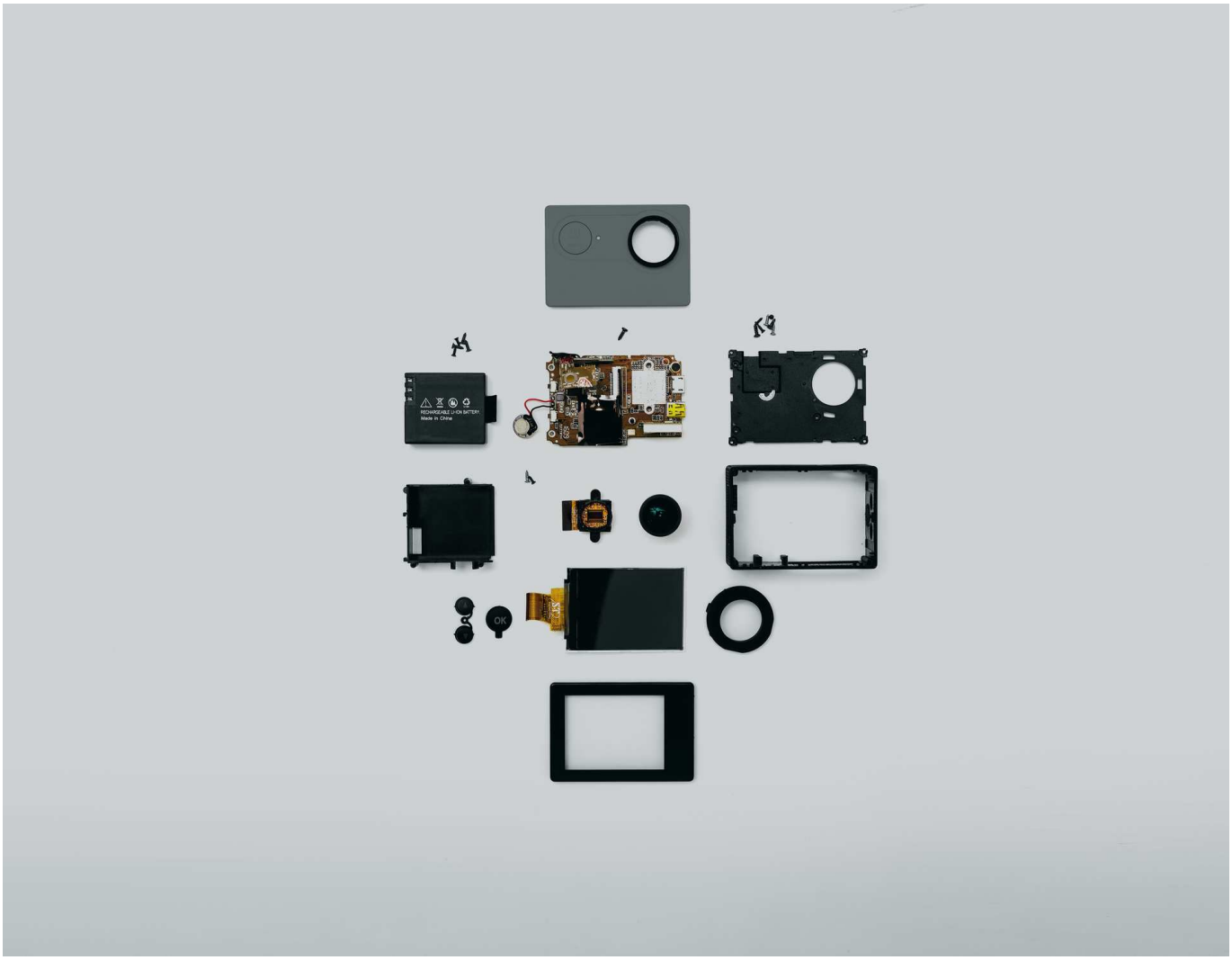

Wstęp do programowania obiektowego w języku Java

Źródło: Alexander Andrews, domena publiczna.

Programowanie obiektowe to sposób programowania, w którym dane oraz funkcje są traktowane jako obiekty komunikujące się między sobą. W podejściu tym tworzymy wzorce danych (klasy) oraz ich instancje (obiekty). W definicji klas zawieramy również funkcje (metody) związane z danymi, które opisują klasy. Programowanie obiektowe ma za zadanie ułatwić pisanie programów – sprawić, by kod był bardziej czytelny. Usprawnia ono również edycję kodu, ponieważ nie trzeba danego błędu poprawiać wiele razy, wystarczy tylko w jednym miejscu.

Podstawowe informacje na temat omawianego zagadnienia znajdziesz w e-materiale Wstęp do programowania obiektowego.

W tym e-materiale poznamy specyfikę programowania zorientowanego obiektowo w języku Java.

Jeśli chcesz dowiedzieć się, jak to zagadnienie wygląda w innych językach programowania, sięgnij do e-materiałów:

- Wstęp do programowania obiektowego w języku C++,
- Wstęp do programowania obiektowego w języku Python.

> **Twoje cele**

- Zweryfikujesz wiedzę o tym, czym są klasy i obiekty oraz na czym polega dziedziczenie.
- Zdefiniujesz klasy w języku Java, używając w nich metod oraz atrybutów.
- Przeanalizujesz strukturę programu zawierającego zdefiniowane klasy w języku Java.
- Zaplanujesz konstrukcję programu w języku Java pod kątem obiektowości.

Film samouczek

Problem 1

Zaimplementuj klasę reprezentującą punkt w kartezjańskim układzie współrzędnych. Klasa powinna posiadać metodę, za pomocą której możliwe będzie obliczenie i wypisanie na standardowe wyjście odległości punktu od początku układu współrzędnych.

Specyfikacja problemu:

Dane:

- x — liczba całkowita; współrzędna x punktu w układzie współrzędnych;
- y — liczba całkowita; współrzędna y punktu w układzie współrzędnych;

Wynik:

- d — liczba rzeczywista; odległość punktu (x, y) od początku układu współrzędnych, z dokładnością do dwóch miejsc po przecinku

```
1
```

```
1
```

Polecenie 1

Porównaj swoje rozwiązanie z programem przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/RIk8xkldzgoxl

Lekcja o tematyce elementów programowania obiektowego w języku Java.

Przeczytaj

Sekcja „Film samouczek” wprowadza w temat programowania obiektowego w języku Java. Materiał filmowy wyjaśnia pojęcia klasy oraz dziedziczenia. W tej sekcji utrwalimy przyswojone informacje.

Zgodnie z założeniem programowania obiektowego klasa jest modelem, który określa typ oraz zachowanie pewnego zbioru danych – obiektu. Obiekt zwany jest instancją klasy. W uproszczeniu można powiedzieć, że klasa jest typem, a obiekt zmienną.

W przypadku typów prostych – `int`, `double` , `boolean` – nie martwimy się ich implementacją, ponieważ zostały nam dostarczone wraz ze środowiskiem programistycznym. W programowaniu obiektowym mamy możliwość samodzielnego tworzenia typów, dzięki czemu idealnie wpasowują się one w nasze potrzeby.

Definicja klasy może zawierać zarówno pola, które stanowią stan obiektu, jak i funkcje (zwane metodami), które określają zachowanie danego obiektu.

Struktura programu obiektowego w języku Java

Program w języku Java rozpoczynamy od definicji klasy. Może ona przyjmować różne nazwy. Niektórzy klasę główną nazywają tak jak swój program, inni wolą nazwać tę klasę: `Main`. Dobrą praktyką jest rozpoczynanie nazwy klasy wielką literą. Wewnątrz niej wpisujemy definicję publicznej statycznej funkcji typu `void` o nazwie `main`. Typowy program w języku Java zawsze przyjmuje podaną strukturę.

```
1 public class Main {
2     public static void main(String[] args) {
3         // w tym miejscu zaczynamy pisać nasz program
4     }
5 }
```

Programując w ten sposób, masz już podstawy programowania obiektowego. Wiesz, co oznaczają poszczególne słowa w tej strukturze?

Zacznijmy od linijki nr 1. Znajdują się w niej dwa słowa kluczowe: `public` oraz `class`. W połączeniu z nazwą `Main` stanowią one definicję klasy publicznej. W języku Java określamy w ten sposób klasę, którą można użyć w dowolnym miejscu programu. Związane jest z nią pojęcie hermetyzacji, które zostanie omówione w kolejnych e-materiałach. W tym momencie możemy przyjąć, że słowo to jest zbędne – jego usunięcie

nie zmieni działania programu. Java jest językiem ściśle obiektowym, w którym nawet funkcja rozpoczynająca wykonywanie programu musi być zadeklarowana w klasie.

Przechodzimy do linijki nr 2. Widzimy bardzo charakterystyczną dla języka Java definicję funkcji `main` – metody, która uruchamiana jest w trakcie startu programu. W momencie kompilacji we wszystkich plikach programu szukana jest właśnie taka definicja. Nie jest określone, w jakiej klasie powinna być zadeklarowana, jednak jest niezbędna do uruchomienia programu.

Wspomniane wcześniej słowo kluczowe `public` oznacza, że dostęp do metody `main` jest powszechny – można ją wywołać w dowolnym miejscu programu, w którym posiadamy dostęp do jej klasy. W tym wypadku również klasa ma ustawiony powszechny dostęp, zatem wywołanie funkcji `main` byłoby możliwe w dowolnym momencie działania programu. Należy jednak pamiętać, że program nie powinien wywoływać sam siebie bez wyraźnej przyczyny, zatem nie należy na przykład wywoływać funkcji `main` z poziomu funkcji `main`.

Słowo kluczowe `static` oznacza, że funkcja `main` jest statyczna. W przypadku funkcji głównej programu nie należy przejmować się jej statycznością. Jak wspomnieliśmy wcześniej, najprawdopodobniej nie będziemy chcieli jej wywoływać. Nie zmienia to faktu, że słowo `static` jest niezbędne do uruchomienia programu.

Słowo kluczowe `void` oznacza, że metoda `main` jest tak naprawdę procedurą – nie zwraca żadnej wartości. W wielu językach uznano, że funkcja główna programu powinna zwracać status, tj. informację czy wszystkie operacje zakończyły się poprawnie, jako typ `integer`. Jednak nie dotyczy to języka Java, w którym nie musimy zwracać statusu programu.

Ostatnią charakterystyczną cechą linijki nr 2 są przyjmowane parametry. Widzimy, że metoda przyjmuje tablicę łańcuchów znaków – `String`. Są to parametry wywołania programu, które mogą pochodzić z systemu operacyjnego lub jego wywołania – np. z poziomu konsoli. Jedynym, co możemy zmienić w drugiej linijce, aby nadal pozostała ona definicją funkcji głównej programu, jest właśnie nazwa tych parametrów.

Dziedziczenie w języku Java

Dziedziczenie to mechanizm współdzielenia pól oraz metod. W programowaniu obiektowym typy mogą pochodzić od innych typów. Klasy pochodne mogą zarówno dziedziczyć, jak i nadpisywać metody klasy, od której pochodzą. Można wyróżnić m.in. klasy właściwe (czyli takie, które mogą być instancjonowane) oraz klasy abstrakcyjne, które stanowią bazę dla klas pochodnych i nie można utworzyć ich instancji. Instancjonowaniem nazywamy tworzenie obiektów danej klasy. Zasada dziedziczenia stanowi zatem również wzór projektowy, dzięki któremu wspólna praca wielu programistów nad tym samym kodem jest łatwiejsza.

Klasa abstrakcyjna w języku Java może zostać utworzona przez słowo kluczowe `abstract`.

```
abstract class Owoc {
    // w tym miejscu możemy deklarować pola oraz metody

    // przykładowe pole w klasie Owoc
    double masa;

    // przykładowa metoda w klasie Owoc
    void ugryz() {
        // ugryzienie powoduje zmniejszenie masy owocu
        masa -= 0.01;
    }
}
```

Klasa `Owoc` została zaimplementowana jako abstrakcyjna, ponieważ definiuje ona tylko ogólną kategorię, jaką są owoce. Definicja ta powinna zostać jednak doprecyzowana, żeby mogła faktycznie zaistnieć. Gdybyśmy pozwolili na utworzenie instancji klasy `Owoc`, nie wiedzielibyśmy, jaki to owoc. Zaimplementujmy zatem klasę pochodną, która powie nam więcej o wybranym owocu. W języku Java klasę pochodną możemy utworzyć poprzez następującą konstrukcję:

```
class Jablko extends Owoc {
    // w tym miejscu możemy deklarować pola oraz metody
    // klasy pochodnej

    // przykładowe pole w klasie Jablko, które nie występuje
    // w klasie pierwotnej
    int liczbaPestek;

    // przykładowa metoda w klasie Jablko, która nie występuje
    // w klasie pierwotnej
    void zabierzPestke() {
        liczbaPestek--;
    }
}
```

Zauważ, że w przypadku klasy `Jablko` nie mamy już do czynienia z klasą abstrakcyjną, zatem możemy utworzyć jej instancję. Sama definicja klasy `Jablko` nie może istnieć samodzielnie, ponieważ odnosi się do definicji klasy abstrakcyjnej `Owoc`. Zwróć uwagę, że

w klasie pochodnej nie musimy zawierać informacji o polach oraz metodach, które już występują w klasie bazowej.

Aby zaobserwować mechanizm dziedziczenia oraz potwierdzić fakt, że w klasie `Jablko` istnieją pola oraz metody z klasy bazowej `Owoc`, a także pola i metody dodane w definicji klasy pochodnej, możemy napisać prosty program wykorzystujący utworzone klasy.

```
abstract class Owoc {
    // w tym miejscu możemy deklarować pola oraz metody

    // przykładowe pole w klasie Owoc
    double masa;

    // przykładowa metoda w klasie Owoc
    void ugryz() {
        // ugryzienie powoduje zmniejszenie masy owocu
        masa -= 0.01;
    }
}

class Jablko extends Owoc {
    // w tym miejscu możemy deklarować pola oraz metody
    // klasy pochodnej

    // przykładowe pole w klasie Jablko, które nie występuje
    // w klasie pierwotnej
    int liczbaPestek;

    // przykładowa metoda w klasie Jablko, która nie występuje
    // w klasie pierwotnej
    void zabierzPestke() {
        liczbaPestek--;
    }
}

public class Main {
    public static void main(String[] args) {
        // nowa instancja klasy Jablko
        Jablko jablko = new Jablko();

        // ustawienie wartości masa oraz liczbaPestek
```

```
        // instancji klasy Jablko
        jablko.masa = 0.3;
        jablko.liczbaPestek = 3;

        // wywołanie metod klasy Jablko
        jablko.ugryz();
        jablko.zabierzPestke();

        // poniższa linijka powinna wypisać: 0.29 2
        System.out.println(jablko.masa + " " + jablko.liczbaPeste
    }
}
```

Konstruktory

W omawianym dotychczas programie, jeśli chcemy przypisać wartość atrybutów obiektów klasy `Jablko`, odwołujemy się bezpośrednio do atrybutów obiektu po znaku kropki. Istnieje jednak specjalny rodzaj funkcji, który pozwala na przypisanie wartości atrybutom obiektów podczas ich tworzenia. Nazywamy go konstruktorem. Prześledźmy definicję konstruktora na przykładzie klasy `Jablko`:

```
class Jablko extends Owoc {
    // w tym miejscu możemy deklarować pola oraz metody
    // klasy pochodnej

    // przykładowe pole w klasie Jablko, które nie występuje
    // w klasie pierwotnej
    int liczbaPestek;

    // konstruktor
    Jablko() {
        // funkcja ta zostanie wywołana przy tworzeniu obiektu
        // klasy Jablko
        // w tym miejscu możemy przypisywać wartości atrybutom
        // oraz wywoływać metody
    }

    // przykładowa metoda w klasie Jablko, która nie występuje
    // w klasie pierwotnej
    void zabierzPestke() {
        liczbaPestek--;
```

```
    }
}
```

W definicji konstruktora możemy zawrzeć także parametry przekazywane tak jak w przypadku zwykłej funkcji:

```
class Jablko extends Owoc {
    // w tym miejscu możemy deklarować pola oraz metody
    // klasy pochodnej

    // przykładowe pole w klasie Jablko, które nie występuje
    // w klasie pierwotnej
    int liczbaPestek;

    // konstruktor
    Jablko(double masa, int liczbaPestek) {
        // funkcja ta zostanie wywołana przy tworzeniu obiektu
        // klasy Jablko
        // w tym miejscu możemy przypisywać wartości atrybutom
        // oraz wywoływać metody
    }

    // przykładowa metoda w klasie Jablko, która nie występuje
    // w klasie pierwotnej
    void zabierzPestke() {
        liczbaPestek--;
    }
}
```

Przypisujemy przekazane do konstruktora wartości do atrybutów. Jeżeli przesłoniliśmy nazwę atrybutu w konstruktorze (lub innej metodzie), dostęp do atrybutu gwarantuje nam konstrukcja `this`.

```
class Jablko extends Owoc {
    // w tym miejscu możemy deklarować pola oraz metody
    // klasy pochodnej

    // przykładowe pole w klasie Jablko, które nie występuje
    // w klasie pierwotnej
    int liczbaPestek;
```

```
8
9       // konstruktor
10      Jablko(double masa, int liczbaPestek) {
11          // funkcja ta zostanie wywołana przy tworzeniu obiektu
12          // klasy Jablko
13          // w tym miejscu możemy przypisywać wartości atrybutom
14          // oraz wywoływać metody
15
16          // przypisanie wartości atrybutu z klasy bazowej
17          this.masa = masa;
18          // przypisanie wartości atrybutu z klasy pochodnej
19          this.liczbaPestek = liczbaPestek;
20      }
21
22      // przykładowa metoda w klasie Jablko, która nie występuje
23      // w klasie pierwotnej
24      void zabierzPestke() {
25          liczbaPestek--;
26      }
27  }
```

Tak zdefiniowaną klasę możemy instancjonować w następujący sposób:

```
1  Jablko jablko = Jablko(0.3, 3);
```

Analizowany program,wzbogacony o wykorzystanie konstruktora, prezentuje się następująco:

```
1  abstract class Owoc {
2      // w tym miejscu możemy deklarować pola oraz metody
3
4      // przykładowe pole w klasie Owoc
5      double masa;
6
7      // przykładowa metoda w klasie Owoc
8      void ugryz() {
9          // ugryzienie powoduje zmniejszenie masy owocu
10         masa -= 0.01;
11     }
12 }
```

```
13
14  class Jablko extends Owoc {
15      // w tym miejscu możemy deklarować pola oraz metody
16      // klasy pochodnej
17
18      // przykładowe pole w klasie Jablko, które nie występuje
19      // w klasie pierwotnej
20      int liczbaPestek;
21
22      // konstruktor
23      Jablko(double masa, int liczbaPestek) {
24          // funkcja ta zostanie wywołana przy tworzeniu obiektu
25          // klasy Jablko
26          // w tym miejscu możemy przypisywać wartości atrybutom
27          // oraz wywoływać metody
28
29          // przypisanie wartości atrybutu z klasy bazowej
30          this.masa = masa;
31          // przypisanie wartości atrybutu z klasy pochodnej
32          this.liczbaPestek = liczbaPestek;
33      }
34
35      // przykładowa metoda w klasie Jablko, która nie występuje
36      // w klasie pierwotnej
37      void zabierzPestke() {
38          liczbaPestek--;
39      }
40  }
41
42  public class Main {
43      public static void main(String[] args) {
44          // nowa instancja klasy Jablko z ustawieniem atrybutów
45          // masa = 0.3 oraz liczbaPestek = 3
46          Jablko jablko = new Jablko(0.3, 3);
47
48          // wywołanie metod klasy Jablko
49          jablko.ugryz();
50          jablko.zabierzPestke();
51
52          // poniższa linijka powinna wypisać: 0.29 2
53          System.out.println(jablko.masa + " " + jablko.liczbaPeste
54      }
```

```
55 }
```

Słownik

> **hermetyzacja**
>
> paradygmat programowania obiektowego, który pozwala na zarządzanie dostępem do atrybutów oraz metod w klasie
> **instancja klasy**
>
> niezależna od innych instancji realizacja modelu danej klasy; stanowi utworzony w pamięci RAM element danych, podczas gdy klasa jest definicją, zgodnie z którą instancja jest tworzona
> **metoda statyczna**
>
> metoda wspólna dla wszystkich instancji danej klasy, może być wywołana również wówczas, gdy nie istnieje żadna instancja danej klasy; metody statyczne nie mogą modyfikować żadnych pól niestatycznych klasy

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program, w którym utworzysz klasę o nazwie `Publikacja`, zawierającą następujące elementy: `tytul`, `autor` oraz `rokWydania` (ich wartości są ustawiane w konstruktorze).

Następnie zaimplementuj w programie klasę `Ksiazka`, która będzie dziedziczyła po klasie `Publikacja`. Nadaj jej również właściwości: `liczbaStron` oraz `numerIsbn`. Klasa `Ksiazka` powinna wykorzystywać konstruktor z odziedziczonej klasy we własnym konstruktorze oraz posiadać metodę `wypiszDane()`, wypisującą wartości wszystkich pól dostępnych w tej klasie. Dodatkowa metoda `ileLatOdWydania()` niech zwraca liczbę całkowitą określającą, ile lat minęło od wydania książki.

Swoje rozwiązanie przetestuj dla obiektu typu `Ksiazka` z następującymi wartościami:
`Alicja w Krainie Czarów` (tytuł), `Lewis Carroll` (autor), `1865` (rok wydania), `241` (liczba stron), `9788377797662` (numer ISBN).

Specyfikacja problemu:

Dane:

Publiczne pola klasy `Publikacja`:

- `tytul` – łańcuch znaków
- `autor` – łańcuch znaków
- `rokWydania` – liczba całkowita

Publiczne pola klasy `Ksiazka`:

- `liczbaStron` – liczba naturalna z przedziału [1, 5000]
- `numerIsbn` – łańcuch znaków

Wynik:

Na standardowym wyjściu program wypisuje wartości wprowadzone podczas tworzenia obiektu zgodnie ze schematem:
`tytul autor rokWydania liczbaStron numerIsbn`.

Twoje zadania

1. Program wypisuje oddzielone spacjami wszystkie dane obiektu `Ksiazka`, np.: `Alicja w Krainie Czarów Lewis Carroll 1865 241 9788377797662`.

Ćwiczenie 2

Napisz program, w którym utworzysz klasę o nazwie `Pojazd`, zawierającą następujące elementy: `marka`, `model` oraz `rokProdukcji` (ich wartości są ustawiane w konstruktorze). Dodatkowo klasa powinna posiadać metodę `wypiszDane()`, wypisującą wartości wszystkich swoich pól.

Następnie stwórz w programie klasę `Samochod`, która będzie dziedziczyła z klasy `Pojazd`. Zaimplementuj jej również pola: `liczbaDrzwi` oraz `rodzajPaliwa`. Klasa `Samochod` powinna wykorzystywać konstruktor z odziedziczonej klasy we własnym konstruktorze oraz przesłaniać metodę `wypiszDane()` wypisującą wartości wszystkich pól (ma wykorzystywać implementację tej metody z klasy dziedziczonej). Stwórz metodę `kosztUtrzymania()`, której zadaniem jest zwrócenie łańcucha znaków mówiącego, jak drogi w utrzymaniu jest samochód. Zaproponuj własną implementację tej metody. Pod uwagę weź dwa parametry: `rodzajPaliwa` i `liczbaDrzwi`.

Swoje rozwiązanie przetestuj dla obiektu typu `Samochod` z następującymi wartościami: `Małe autko` (marka), `Compact+` (model), `2019` (rok produkcji), `4` (liczba drzwi), `Elektryk` (rodzaj paliwa).

Specyfikacja problemu:

Dane:

Publiczne pola klasy `Pojazd`:

- `marka` – łańcuch znaków
- `model` – łańcuch znaków
- `rokProdukcji` – liczba całkowita

Publiczne pola klasy `Samochod`:

- `liczbaDrzwi` – liczba naturalna z przedziału [2, 5]
- `rodzajPaliwa` – łańcuch znaków

Wynik:

Na standardowym wyjściu program wypisuje wartości wprowadzone podczas tworzenia obiektu zgodnie ze schematem: `marka model rokProdukcji liczbaDrzwi rodzajPaliwa`.

Twoje zadania

1. Program wypisuje oddzielone spacjami wszystkie dane obiektu `Samochod`, np.: `Małe autko Compact+ 2019 4 Elektryk`.

Ćwiczenie 3

Napisz program, w którym utworzysz klasę o nazwie `Urzadzenie`, zawierającą następujące elementy: `marka`, `model` oraz `rokProdukcji` (ich wartości są ustawiane w konstruktorze). Dodatkowo klasa powinna posiadać metodę `wypiszDane()` wypisującą wartości wszystkich swoich pól.

Następnie stwórz w programie klasę `Telefon`, która będzie dziedziczyła publicznie z klasy `Urzadzenie`. Zaimplementuj jej również pola: `iloscWbudowanejPamieciDyskowej` (wyrażona w GB) oraz `ksiazkaTelefoniczna`. Klasa `Telefon` powinna wykorzystywać konstruktor z odziedziczonej klasy we własnym konstruktorze oraz przesłaniać metodę `wypiszDane()` wypisującą wartości wszystkich właściwości (ma wykorzystywać implementację tej metody z klasy dziedziczonej). Dodaj do klasy `Telefon` metodę `dodajKontakt(nazwaKontaktu)`. Metoda powinna dodawać do pola `ksiazkaTelefoniczna` nowy kontakt.

Na koniec stwórz w programie klasę `Smartfon`, która będzie dziedziczyła publicznie z klasy `Telefon`. Niech posiada ona również swoje właściwości: `systemOperacyjny` oraz `iloscPamieciRAM`. Klasa `Smartfon` powinna wykorzystywać konstruktor z odziedziczonej klasy we własnym konstruktorze oraz przesłaniać metodę `wypiszDane()`, wypisującą wartości wszystkich pól (ma wykorzystywać implementację tej metody z klasy dziedziczonej).

Swoje rozwiązanie przetestuj dla obiektu typu `Smartfon` z następującymi wartościami: `PhonX` (marka), `PhonX Big` (model), `2021` (rok produkcji), `256` (wbudowana pamięć dyskowa), `SuperOS` (system operacyjny), `16` (pamięć RAM).

Specyfikacja problemu:

Dane:

Publiczne pola klasy `Urzadzenie`:

- `marka` – łańcuch znaków
- `model` – łańcuch znaków
- `rokProdukcji` – liczba całkowita

Publiczne pola klasy `Telefon`:

- `iloscWbudowanejPamieciDyskowej` – liczba naturalna z przedziału [2, 512]
- `ksiazkaTelefoniczna[]` – dziesięcioelementowa tablica łańcuchów znaków

Publiczne pola klasy `Smartfon`:

- `systemOperacyjny` – łańcuch znaków
- `iloscPamieciRAM` – liczba naturalna z przedziału [4, 32]

Wynik:

Na standardowym wyjściu program wypisuje wartości wprowadzone podczas tworzenia obiektu zgodnie ze schematem: `marka model rokProdukcji iloscWbudowanejPamieciDyskowej systemOperacyjny iloscPamieciRAM`.

Twoje zadania

1. Program wypisuje oddzielone spacjami wszystkie dane obiektu `Smartfon`, np.: `PhonX PhonX Big 2021 256 SuperOS 16`.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Wstęp do programowania obiektowego w języku Java

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Zweryfikujesz wiedzę o tym, czym są klasy i obiekty oraz na czym polega dziedziczenie.
- Zdefiniujesz klasy w języku Java, używając w nich metod oraz atrybutów.
- Przeanalizujesz strukturę programu zawierającego zdefiniowane klasy w języku Java.
- Zaplanujesz konstrukcję programu w języku Java pod kątem obiektowości.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji).

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Wstęp do programowania obiektowego w języku Java". Uczniowie mają rozwiązać problem 1 z sekcji „Film samouczek" i porównać swój program z przedstawionym w filmie.

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel prosi wybraną osobę o zaprezentowanie rozwiązania problemu 1 z sekcji „Film samouczek". Pozostali uczniowie dzielą się swoimi spostrzeżeniami. W razie potrzeby nauczyciel wyjaśnia wątpliwości i odpowiada na pytania uczniów.
2. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Uczniowie przystępują do cichego czytania tekstu. W kolejnym kroku chętne osoby podsumowują najważniejsze informacje zawarte w tej sekcji.
3. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że będą rozwiązywać ćwiczenie nr 1 z sekcji „Sprawdź się". Każdy z uczniów robi to samodzielnie. Po ustalonym czasie następuje porównanie napisanych kodów podczas wspólnego omówienia rozwiązań.
4. Liga zadaniowa – uczniowie pracując w parach, wykonują ćwiczenie nr 2 z sekcji „Sprawdź się", a następnie dzielą się swoimi wynikami przez porównywanie napisanego kodu z inną grupą, która również zakończyła zadanie.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Wybrany uczeń podsumowuje zajęcia z programowania w Javie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 3 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).

- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj” można wykorzystać na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.