Operacje wejścia i wyjścia w języku Java

Źródło: Christina Morillo, domena publiczna.

> W tym e-materiale powtarzamy wiadomości ze szkoły podstawowej.

Jak komunikować się z napisanym programem? Służą do tego poznane już przez nas operacje wejścia i wyjścia. Pozwalają one na dwustronną komunikację użytkownika z programem. Obejmują wszystkie czynności, od odczytywania przycisków z klawiatury do wyświetlania filmów na ekranie.

W tym e-materiale zapoznamy się z operacjami wejścia i wyjścia w języku Java.

Charakterystykę operacji wejścia i wyjścia w innych językach programowania znajdziesz w e-materiałach:

- Operacje wejścia i wyjścia w języku C++,
- Operacje wejścia i wyjścia w języku Python.

> **Twoje cele**
>
> - Wyjaśnisz, czym są operacje wejścia i wyjścia w języku Java.
> - Stworzysz obiekty typu `Scanner` i wywołasz na nich odpowiednie metody.
> - Napiszesz własny program do obliczenia dziennego zapotrzebowania kalorycznego.

Przeczytaj

Jak komunikuje się program komputerowy

Naszym zadaniem jest napisanie programu spełniającego funkcję kalkulatora. W jaki sposób będziemy wprowadzać do niego dane? Mamy dostęp do kodu źródłowego i to bezpośrednio w nim możemy edytować wartości zmiennych. Zauważmy, że program napisany w ten sposób, choć może być dobrym ćwiczeniem programistycznym, jest po prostu nieużyteczny. Każdorazowa zmiana danych, na których operujemy w programie, wymagałaby edycji kodu źródłowego, a poza tym nikt inny nie mógłby z niego korzystać.

Użytkownik może wprowadzać informacje do programu np. za pomocą klawiatury.

Nawet najprostszy program wyświetlający napis „Hello World" korzysta z operacji wyjściowej `System.out`.

Obie wymienione operacje: pobierania i wypisywania danych, są wykonywane z użyciem strumieni.

Strumienie wejścia/wyjścia, czyli I/O

Programy napisane w języku Java komunikują się z nami za pomocą strumieni I/O (Input/Output). Wyróżniamy następujące strumienie:

- `System.in` – wejścia,
- `System.out` – wyjścia,
- `System.err` – szczególny rodzaj strumienia wyjścia służący do obsługi wszelkiego rodzaju wyjątków – błędów w programie.

> **Ciekawostka**
>
> Jeżeli korzystasz ze środowiska programistycznego Eclipse, błędy, które wypisują się w konsoli IDE, są komunikowane za pomocą strumienia `System.err` i czerwonej czcionki.

Strumienie są najczęściej podłączone do źródła, takiego jak np. baza danych, plik itp. My posłużymy się danymi wprowadzanymi przez użytkownika z klawiatury.

Obiekt `Scanner`

Stwórzmy zatem najprostszy strumień wejścia za pomocą obiektu klasy `Scanner`, czyli klasy potrzebnej do odczytania danych wprowadzanych przez użytkownika.

W tym celu stworzymy klasę `UserInput` z metodą główną `main()`.

Aby utworzyć obiekt typu `Scanner`, musimy stworzyć instancję, czyli egzemplarz klasy `Scanner` wewnątrz naszej metody `main()`. W konstruktorze klasy `Scanner` znajduje się `System.in`. Jest to strumień wejścia, na którym będziemy operować.

Robimy to w następujący sposób:

```
Scanner sc = new Scanner(System.in);
```

Jest to ogólny schemat tworzenia obiektów w języku Java.

```
NazwaKlasy nazwaObiektu = new NazwaKlasy();
```

Wpisujemy najpierw nazwę klasy, której instancję (obiekt) chcemy utworzyć, potem nazwę obiektu (nadaną przez nas), następnie operator przypisania =, słowo kluczowe `new` (służące do tworzenia obiektów) i ponownie nazwę klasy.

Jeśli obiektowi nadamy np. nazwę `scanner`, całe wywołanie będzie wyglądać następująco:

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
}
```

Należy jednak pamiętać, że aby korzystać z klasy `Scanner`, musimy zaimportować odpowiedni pakiet języka Java, który zawiera klasę `Scanner`:

```
import java.util.Scanner;

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
}
```

> **Ciekawostka**
>
> Jeżeli korzystasz ze środowiska programistycznego Eclipse i użyjesz klasy `Scanner` bez importowania odpowiedniego pakietu, pojawi się błąd w postaci małej litery x na czerwonym tle w linijce, gdzie została użyta klasa `Scanner`. Wtedy wystarczy kliknąć w ikonę żaróweczki obok litery x, a Eclipse podpowie, jak rozwiązać ten problem.

> **Ważne!**
>
> Należy zwracać uwagę na stosowanie małych i wielkich liter w kodzie programu.

> **Dla zainteresowanych**
>
> Strumienie są wbudowane w SDK języka Java i dostępne bez konieczności importu. Zauważmy, że użycie instrukcji `System.in` nie spowodowało przerwania procesu kompilacji. Jednak wiele pakietów dostępnych w języku Java wymaga dodatkowego zaimportowania. Nie importujemy za każdym razem wszystkich, ponieważ znacząco spowolniłoby to pracę IDE, gdyby przy uruchomieniu najprostszej klasy program musiał przechodzić przez niezliczone pakiety i biblioteki dostępne w języku Java. W omawianej klasie zaimportowaliśmy jedynie klasę `Scanner` z obszernego pakietu `java.util`. Gdybyśmy jednak chcieli zaimportować cały pakiet `util`, zamiast:
>
> ```
> 1 import java.util.Scanner;
> ```
>
> musielibyśmy wpisać:
>
> ```
> 1 import java.util.*;
> ```

Stworzyliśmy obiekt klasy `Scanner`, na którym będzie teraz można wywoływać metody pozwalające na odczyt różnych typów danych.

> **Ważne!**
>
> W języku Java metody można wywoływać tylko na obiektach lub bezpośrednio na klasach. Na typach prymitywnych (takich jak `int` czy `double`) nie da się tego zrobić.

Użyjmy utworzonego wcześniej obiektu klasy `Scanner`. Spróbujmy zapisać do łańcucha znaków `String imie` tekst pobrany od użytkownika. Zrobimy to w następujący sposób:

```
1  import java.util.Scanner;
2
3  public class UserInput {
4
5      public static void main(String []args){
6         Scanner scanner = new Scanner(System.in);
7         String imie = scanner.next();
8
9         System.out.println(imie);
10     }
```

```
}
```

> **Już wiesz**
>
> W języku Java metody na obiektach wywołujemy za pomocą kropki.

Powyższy program pobierze ciąg znaków wprowadzony przez użytkownika, następnie zapisze go w zmiennej `imie`, a za pomocą strumienia wyjściowego `System.out` wypisze na ekran konsoli.

Metoda `next()` odczytuje dane do napotkania pierwszego białego znaku (spacji), a zatem może odczytać jedno słowo.

Do powyższego programu można dopisać linię odpowiedzialną za interakcję z użytkownikiem – za pomocą metody `println()` wypiszemy na standardowym wyjściu informację, co ma zrobić, jakie dane ma wprowadzić itp.:

```
import java.util.Scanner;

public class UserInput {

     public static void main(String []args){
        Scanner scanner = new Scanner(System.in);
        System.out.println("Jak się nazywasz?");
        String imie = scanner.next();

        System.out.println(imie);
     }
}
```

Po uruchomieniu programu na standardowym wyjściu pojawi się:

```
Jak się nazywasz?
```

Następnie program oczekuje na wprowadzanie danych. Po wprowadzeniu:

```
Jak się nazywasz?
Jan Kowalski
```

zostanie wypisane:

```
Jak się nazywasz?
Jan Kowalski
Jan
```

System wydrukował na ekran konsoli jedynie słowo „Jan", ponieważ użyliśmy metody `next()`, która odczytuje ciąg znaków do momentu pojawienia się spacji. Gdybyśmy chcieli odczytać całą linijkę, musielibyśmy użyć metody `nextLine()`.

Metody obiektu Scanner

Klasa `Scanner` oferuje nam szereg metod odczytu danych od użytkownika. W języku Java nie jest to jednak taki łatwy mechanizm. **Pamiętaj, że Java jest językiem o silnym typowaniu.** W związku z tym odczytuje dane jako jakiś **konkretny typ.**

Chcąc pobrać od użytkownika np. dane dotyczące jego wieku, można to zrobić w sposób identyczny jak zaprezentowany wcześniej i zapisać w zmiennej `String`. Jednak wtedy wprowadzona wartość będzie ciągiem znaków, a więc nie będzie można na niej wykonywać żadnych działań matematycznych. W takim przypadku wiek podany przez użytkownika należy zapisać do zmiennej typu `int`. Robimy to w następujący sposób:

```
int imie = scanner.nextInt();
```

> **Dla zainteresowanych**
>
> Obiektu `Scanner` możemy używać wielokrotnie w obrębie klasy, ponieważ w języku Java jest możliwość wielokrotnego używania obiektów. Nie należy tworzyć obiektów, kiedy nie ma takiej potrzeby.

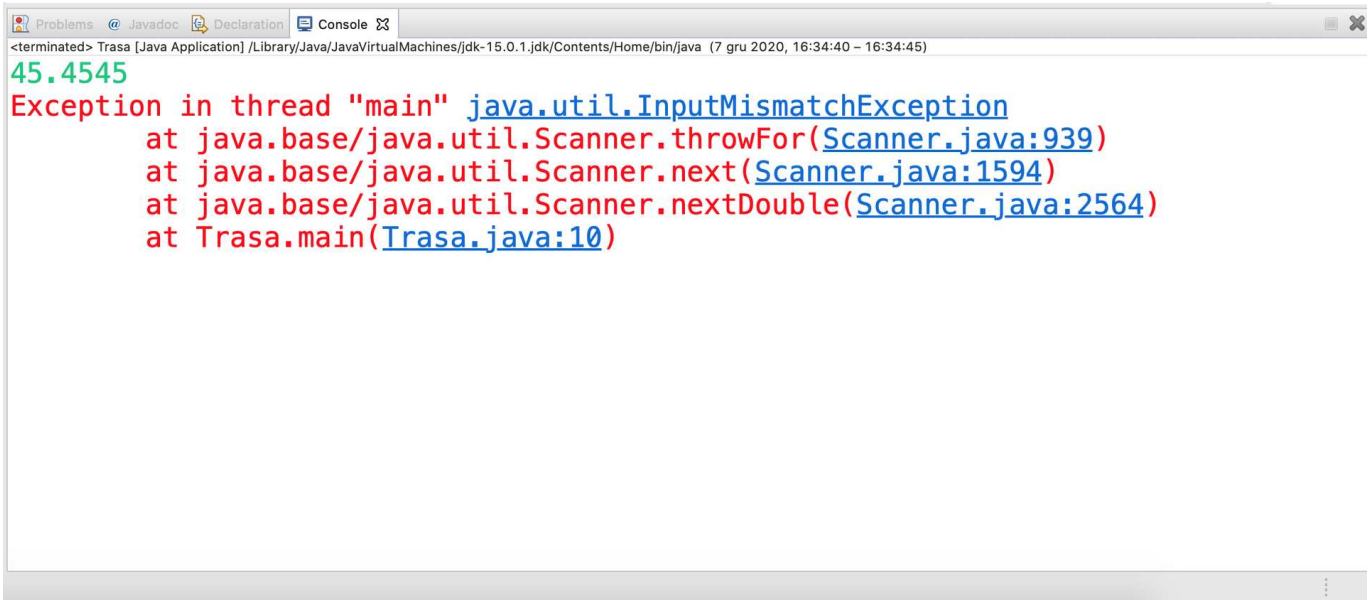
> **Ważne!**
>
> Kiedy tworzymy obiektu typu `Scanner`:
>
> ```
> Scanner scan = new Scanner(System.in);
> ```
>
> Jednocześnie **otwieramy strumień wejścia** za pomocą `System.in` w argumencie.
>
> Dlatego dobrym nawykiem jest używanie metody zamykającej strumień:

```
1 Scanner.close();
```

Wprawdzie może on się zamknąć sam, ale nie wiemy na pewno, czy tak się stanie. Strumienie działają na mechanizmach niskopoziomowych, takich jak jądro systemu, dlatego programując w języku Java, nie do końca mamy nad nimi kontrolę. Przy dostępie jedynie do danych wpisywanych przez użytkownika w konsoli prawdopodobnie nie będzie to problem, jednak w przypadku pracy z plikami czy dużą bazą danych mógłby wystąpić błąd typu Resource Leak, czyli wyciek zasobów.

Metoda useLocale

Może się zdarzyć, że przy wprowadzaniu danych typu `double` (545.43243d) czy `float` (6.75f) pojawi się błąd, który będzie trudny do zinterpretowania:



Metody takie jak `nextInt()` czy `nextFloat()` wymagają wprowadzenia liczb w odpowiednim formacie. W przypadku `nextInt()` klasa `Scanner` oczekuje ciągu cyfr, który zamieniony na liczbę znajdzie się w zakresie danych, które może przechowywać typ `int`. Sytuacja komplikuje się dla metod `nextFloat()` czy `nextDouble()` – w zależności od ustawień regionalnych systemu operacyjnego użytkownika separatorem rozdzielającym część całkowitą liczby od części ułamkowej może być kropka lub przecinek.

Jeśli wpisaliśmy liczbę typu `double` czy `float`, używając kropki, a otrzymaliśmy powyższy błąd, to znaczy, że `Scanner` nie umiał poprawnie zinterpretować liczby, co najprawdopodobniej ma związek z ustawieniami regionalnymi (`locale`). Rozwiązaniem problemu jest wprowadzenie liczby z użyciem przecinka.

Jeżeli jednak chcesz wczytywać liczby w formacie anglojęzycznym, możesz użyć metody `useLocale()` na obiekcie typu `Scanner`, gdzie argumentem w nawiasach będzie obiekt `Locale` oraz nazwa danego języka lub kraju bądź też inna metoda, np.:

```
scanner.useLocale(Locale.ENGLISH);
```

Teraz `Scanner` będzie wczytywał i interpretował dane zgodnie ze standardami języka angielskiego.

Klasę `Locale`, podobnie jak `Scanner`, będziemy musieli zaimportować z pakietu `java.util;`

```
import java.util.Locale;
```

Ćwiczenie 1

Napisz program w języku Java do obliczenia wartości BMR, czyli Basal Metabolic Rate (po polsku PPM – podstawowa przemiana materii) dla kobiet, który na podstawie wprowadzonych danych będzie obliczał podstawowe zapotrzebowanie na kalorie użytkowniczki.

BMR liczymy, wykorzystując następujący wzór:

$$\mathrm{BMR}(\text{in kcal}) = 655 + (9.6 \cdot \text{waga}) + (1.7 \cdot \text{wzrost}) - (4.7 \cdot \text{wiek})$$

Specyfikacja:

Dane:

- `imie` – ciąg znaków; imię użytkowniczki
- `waga` – liczba zmiennoprzecinkowa dodatnia; waga użytkowniczki w kg
- `wzrost` – liczba zmiennoprzecinkowa dodatnia; wzrost użytkowniczki w cm

Wynik:

Program prezentuje podstawowe zapotrzebowanie na kalorie użytkowniczki.

Implementację zacznijmy od stworzenia klasy `CalcBMR` z metodą `main()`.

Zaczynamy od utworzenia obiektu typu `Scanner`. Za pomocą metody `getDefault()` w parametrze metody `useLocale()` wymuszamy domyślne ustawienia regionalne (dla Polski) – aby polski użytkownik, wpisując liczby, mógł intuicyjnie używać przecinka. Nasz kalkulator może wyglądać np. tak:

```
1 import java.util.Scanner;
2 import java.util.Locale;
3
4 public class CalcBMR {
5
6     public static void main(String []args){
7        Scanner scanner = new Scanner(System.in);
```

```
        scanner.useLocale(Locale.getDefault());

        System.out.println("Jak masz na imie?");
        String imie = scanner.next();

        System.out.println("Witaj " + imie + ", podaj swoja wage.
        float waga = scanner.nextFloat();

        System.out.println("Podaj swój wzrost w centymetrach.");
        float wzrost = scanner.nextFloat();

        System.out.println("Podaj swój wiek.");
        int wiek = scanner.nextInt();

        double bmr = 655 + (9.6 * waga) + (1.7 * wzrost) - (4.7 *

        System.out.println("Twoje podstawowe zapotrzebowanie kalo

        scanner.close();
    }
}
```

Przykładowa, interaktywna sesja z użytkowniczką może wyglądać następująco:

```
Jak masz na imie?
Milena
Witaj Milena, podaj swoja wage.
65
Podaj swój wzrost w centymetrach.
174
Podaj swój wiek.
27
Twoje podstawowe zapotrzebowanie kaloryczne wynosi: 1447.89999999
```

Wartość BMR nie jest jednak zbyt czytelna. Zastosujmy więc metodę z biblioteki dostępnej dla języka Java z klasy `java.lang.Math` w celu zaokrąglenia wyniku do pełnej liczby. Będzie to metoda:

```
round();
```

Wystarczy więc dopisać kilka znaków w ostatniej linijce kodu.

```
System.out.println("Twoje podstawowe zapotrzebowanie kaloryczne w
```

Po zmianach przykładowa, interaktywna sesja może wyglądać następująco:

```
Jak masz na imie?
Milena
Witaj Milena, podaj swoja wage.
65
Podaj swój wzrost w centymetrach.
174
Podaj swój wiek.
27
Twoje podstawowe zapotrzebowanie kaloryczne wynosi: 1448 kcal.
```

Słownik

konstruktor

specjalna metoda klasy, której wywołanie nastąpi przy inicjalizacji obiektu danej klasy

słowo kluczowe

słowo oznaczające polecenie, instrukcję, definicję lub deklarację w programie komputerowym; lista słów kluczowych jest zazwyczaj specyficzna dla konkretnego języka programowania; w języku Java przykładowe słowa kluczowe to: `int`, `for`, `static`, `public`, `private`, `class`

IDE

(ang. *Integrated Development Environment*); zintegrowane środowisko programistyczne; najczęściej zawiera edytor kodu źródłowego oraz wbudowany kompilator lub interpreter

SDK

(ang. *Software Development Kit*); zestaw narzędzi dla programistów niezbędny w tworzeniu aplikacji korzystających z funkcjonalności danej biblioteki

Film samouczek

Polecenie 1

Zapoznaj się z prezentacją przedstawiającą implementację algorytmu obliczającego średnią prędkość na zadanej przez użytkownika trasie, a następnie rozwiąż problem 1.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Przeanalizujmy krok po kroku program obliczający średnią prędkość na zadanej przez użytkownika trasie.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Program ma działać w następujący sposób:

1. Użytkownik wprowadza punkt startowy.
2. Użytkownik wprowadza punkt końcowy.
3. Użytkownik wprowadza odległość, jaką przebył.
4. Użytkownik wprowadza czas, w jakim przebył odcinek między punktem startowym a końcowym.
5. Program oblicza średnią prędkość według otrzymanych danych.

6. Program wypisuje dane dotyczące trasy użytkownikowi.

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Zacznijmy od utworzenia klasy `Trasa` i zadeklarowania głównej funkcji programu `main()`

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Następnie utwórzmy obiekt klasy `Scanner`. Za jego pomocą będziemy pobierać kolejne dane od użytkownika.

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

W związku z użyciem klasy `Scanner` należy pamiętać o zaimportowaniu odpowiedniej biblioteki.

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Teraz możemy rozpocząć interakcję z użytkownikiem. Zacznijmy od pobrania imienia użytkownika. Zrobimy to za pomocą jednej z metod klasy `Scanner` – `next()`, a wartość zapiszemy do łańcucha znaków `String imie`.

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Następnie pobierzemy punkt startowy trasy. Ponieważ może on składać się z więcej niż jednego słowa, zrobimy to za pomocą metody `nextLine()`.

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Kolejnym krokiem jest pobranie informacji o punkcie końcowym trasy.

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Następnie użytkownik poda w kilometrach odległość, jaką przebył. Otrzymaną wartość zapiszemy do zmiennej typu `double`, a pobierzemy za pomocą metody `nextDouble()`.

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Ostatnią informacją, jaką pobierzemy od użytkownika, jest czas (w godzinach), w jakim przebył podaną wcześniej odległość.

11

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Zdefiniujmy operację matematyczną, która wyliczy średnią prędkość, w jakiej użytkownik przebył daną trasę. Wynik zostanie zapisany w zmiennej `predkosc`.

12

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Ostatnim krokiem jest wypisanie użytkownikowi informacji o trasie oraz zamknięcie strumienia wejścia. Została dodana

metoda z biblioteki języka Java z klasy `java.lang.Math` w celu zaokrąglenia wyniku do jedności.

13

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PFhUp6DAj

Przykładowe działanie programu:

Problem 1

Napisz program, który pobierze od użytkownika wysokość i szerokość ściany, a następnie policzy, ile potrzeba farby, by pomalować jedną warstwą ścianę.

Zakładamy, że do pomalowania metra kwadratowego ściany potrzeba 13 litrów farby.

Specyfikacja:

Dane:

- `wysokosc`, `szerokosc` – liczby rzeczywiste
- `zuzycieFarby` – liczba naturalna; liczba litrów farby potrzebna do pomalowania metra kwadratowego ściany

Wynik:

Ilość farby potrzebna do pomalowania ściany jedną warstwą.

Działanie programu przetestuj w środowisku zainstalowanym na swoim komputerze, ponieważ tester kodu nie obsługuje plików oraz pobierania danych od użytkownika.

Polecenie 2

Zapoznaj się z filmem, w którym zaprezentowano implementację algorytmu obliczającego BMI użytkownika.

Film dostępny pod adresem https://zpe.gov.pl/a/DSMvYSw0A

Film nawiązujący do treści materiału: Komunikacja programu z użytkownikiem.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wskaż, która metoda poprawnie odczyta imię i nazwisko użytkownika.

- ◯ nextString();
- ◯ hasNext();
- ◯ next();
- ◯ nextLine();

Ćwiczenie 2

Wskaż prawidłowo zainicjowany obiekt Scanner.

- ◯ Scanner scanner = new Scanner();
- ◯ Scanner = new Scanner(System.in);
- ◯ Scanner scanner = new Scanner(System.in);
- ◯ Scanner = new Scanner();

Ćwiczenie 3

Wskaż fragmenty kodu, które pobiorą ciąg znaków od użytkownika i poprawnie zapiszą go do zmiennej.

- [ ]
```
Scanner sc = new Scanner();
int i = sc.nextInt();
```

- [ ]
```
Scanner sc = new Scanner(System.in);
String i = sc.nextLine();
```

- [ ]
```
Scanner sc = new Scanner(System.in);
String i = sc.nextInt();
```

- [ ]
```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

- [ ]
```
Scanner sc = new Scanner(System.in);
String i = sc.next();
```

Ćwiczenie 4

Zapisz fragment kodu odpowiedzialny za dodanie odpowiedniego pakietu w języku Java pozwalającego korzystać z klasy `Scanner`.

Ćwiczenie 5

Wskaż, który z poniższych strumieni jest strumieniem wejścia.

- ◯ `System.out`
- ◯ `System.err`
- ◯ `System.in`

Ćwiczenie 6

Uzupełnij zdanie.

Jeśli strumień wejścia nie zostanie zamknięty, [ ].

strumień nie będzie pobierał ciągów znaków od użytkownika

metody używane na strumieniu nie będą działać

może nastąpić wyciek zasobów

Ćwiczenie 7

Wskaż, która metoda poprawnie odczyta od użytkownika jego wiek.

- ◯ `nextInt()`
- ◯ `nextDouble()`
- ◯ `nextLine()`
- ◯ `next()`

Ćwiczenie 8

Wskaż, co będzie wynikiem działania programu dla wprowadzonych danych: `AAA BBB CCC`.

```
1 Scanner sc = new Scanner(System.in);
2
3 String s = sc.next();
4
5 System.out.println(s);
```

- ◯ `1 A`
- ◯ `1 CCC`
- ◯ `1 AAA`
- ◯ `1 B`
- ◯ `1 AAA BBB`
- ◯ `1 BBB`
- ◯ `1 AAA BBB CCC`
- ◯ `1 C`

Dla nauczyciela

Autor: Maurycy Gast

Przedmiot: Informatyka

Temat: Operacje wejścia i wyjścia w języku Java

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy i rozszerzony

Podstawa programowa:
Zakres podstawowy i rozszerzony
Cele kształcenia – wymagania ogólne
1) Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.
2) 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
Zakres podstawowy
II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych. Uczeń:
1. projektuje i programuje rozwiązania problemów z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, funkcje z parametrami i bez parametrów, testuje poprawność programów dla różnych danych; w szczególności programuje algorytmy z punktu I.2);
Zakres rozszerzony
II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:
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):

- Wyjaśnisz, czym są operacje wejścia i wyjścia w języku Java.
- Stworzysz obiekty typu `Scanner` i wywołasz na nich odpowiednie metody.
- Napiszesz własny program do obliczenia dziennego zapotrzebowania kalorycznego.

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: „Operacje wejścia i wyjścia w języku Java". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.
2. Prowadzący prosi uczniów, aby zgłaszali swoje propozycje pytań do tematu. Jedna osoba może zapisywać je na tablicy. Gdy uczniowie wyczerpią swoje pomysły, a pozostały jakieś ważne kwestie do poruszenia, nauczyciel je dopowiada.

Faza realizacyjna:

1. **Praca z tekstem.** Jeżeli przygotowanie uczniów do lekcji jest niewystarczające, nauczyciel prosi o indywidualne zapoznanie się z treścią zawartą w sekcji „Przeczytaj". Każdy uczestnik zajęć podczas cichego czytania wynotowuje najważniejsze kwestie poruszane w tekście.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie wspólnie zapoznają się z treścią problemu 1. Następnie w parach piszą program, który następnie porównują z rozwiązaniem zaprezentowanym w prezentacji.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1-8 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel zadaje pytania podsumowujące, np.
   – co oznacza termin „słowo kluczowe"?
   – jak nazywamy specjalną metodę klasy, której wywołanie nastąpi przy inicjalizacji obiektu danej klasy?
2. 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.

Praca domowa:

1. Uczniowie proponują alternatywny sposób rozwiązania problemów postawionych w sekcji „Przeczytaj".

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 jako podsumowanie i utrwalenie wiedzy uczniów.