Kreatywne programowanie

Zajęcia nr 4

- *Programowanie obiektowe*
- *pętla while, odczyt i zapis danych*

Piotr Radzikowski
ptr.radzikowski@gmail.com
radzikowski.one.pl

O czym mówiliśmy ostatnio? :)

- typach danych i formatowanu ciągu znaków (tabulacja, nowa linia, itd)
- czym jest **klasa** i dziedziczenie (na przykładzie, człowiek/"**Human**")
- Specjalna metoda **toString**
- hermetyzacji danych (gettery i settery)
- klasa "**Book.java**"

Programowanie obiektowe

Przedstawienia świata rzeczywistego i relacji w nim zachodzących, za pomocą obiektów.

Najważniejszymi cechami programowania obiektowego są:

- **Enkapsulacja/Hermetyzacja**
- **Abstrakcja**
- **Dziedziczenie**
- **Polimorfizm**

Enkapsulacja/hermetyzacja

Enkapsulacja (ang. *encapsulation*), czy inaczej *hermetyzacja* to sposób na ukrycie szczegółów implementacji klasy.

Enkapsulacja to bardzo ważny element programowania obiektowego - pozwala na pełną kontrolę nad zachowaniem i stanem danego obiektu.

Modyfikatory dostępu

Modyfikator public

Słowo kluczowe **public** jest modyfikatorem dostępu, który pozwala na najbardziej swobodny dostęp do elementu, który poprzedza. **public** może być używane przed definicjami klas, pól w klasach, metod czy typów wewnętrznych.

```
public String zmienna;
```

Modyfikatory dostępu

Modyfikator private

Słowo kluczowe **private** jest najbardziej restrykcyjnym modyfikatorem dostępu. Może być stosowane wyłącznie przed elementami klasy, w tym przed klasami wewnętrznymi. Oznacza on tyle, że dany element (klasa, metoda, czy pole) widoczny jest tylko i wyłącznie wewnątrz klasy.

```
private String zmienna;
```

Modyfikatory dostępu

Modyfikator protected

Modyfikator **protected** ma znaczenie w przypadku dziedziczenia. Elementy poprzedzone tym modyfikatorem dostępu są udostępnione dla danej klasy i jej podklas. Dodatkowo elementy oznaczone modyfikatorem **protected** dostępne są dla innych klas w tym samym pakiecie. Modyfikatora protected nie można stosować przed klasami.

```
protected String zmienna;
```

Brak modyfikatora dostępu

Brak modyfikatora dostępu również ma znaczenie. W przypadku gdy pominiemy modyfikator dostępu wówczas dana klasa czy element jest dostępna wyłącznie wewnątrz tego samego pakietu. Jest to podzbiór uprawnień, które nadaje modyfikator **protected**.

```
String zmienna;
```

```
protected String zmienna;
```

Enkapsulacja w praktyce czyli - Gettery i settery

- Co to jest?
- Po co używać?
- Kiedy używać?

Point.java > ...

```java
class Point {
  private double x;
  private double y;

  public Point(double x, double y) {
      this.x = x;
      this.y = y;
  }

  public double getX() { return x; }
  public double getY() { return y; }

  public void setX(double x) { this.x = x; }
  public void setY(double y) { this.y = y; }
}
```

Piszemy klasę “książka”

Book.java > ...

```java
/**
 * Book
 */
public class Book {

    private int pageAmount;
    private String title;
    private String author;

    public Book(String title, String author, int pageAmount) {
      this.title = title;
      this.author = author;
      this.pageAmount = pageAmount;
    }

    @Override
    public String toString() {
      return "------------------------------------ Książka ----------------------------------------\n"
          + "\tTytuł:\t\"" + title + "\"\n"
          + "\tAutor:\t\"" + author + "\"\n"
          + "\tStron:\t\"" + pageAmount + "\"\n"
          + "-------------------------------------------------------------------------------------------\n";
    }
  }

```

Main

```java
/**
 * Main
 */
public class Main {

    Run | Debug
    public static void main(String[] args) {
        Book book = new Book("Tytuł", "Autor", 1);
```

Main - wypisanie na ekran danych

```java
/**
 * Main
 */
public class Main {


  public static void main(String[] args) {
    Book book = new Book("Tytuł", "Autor", 1);
    System.out.println(book);
```

Odczytywanie danych z konsoli

Klasa Scanner

```
import java.util.Scanner;

public class Main {

    Run | Debug
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String title = in.nextLine();
        String author = in.nextLine();
        int page = in.nextInt();
        Book book = new Book(title, author, page);
        System.out.println(book);
```

Przeciążony konstruktor i parsowanie danych

```
public class Book {

  private String author;
  private String title;
  private int pageAmount;

  public Book(String author, String title, int pageAmount) {
    this.author = author;
    this.title = title;
    this.pageAmount = pageAmount;
  }

  public Book(String data) {

  }
```

Parsowanie danych

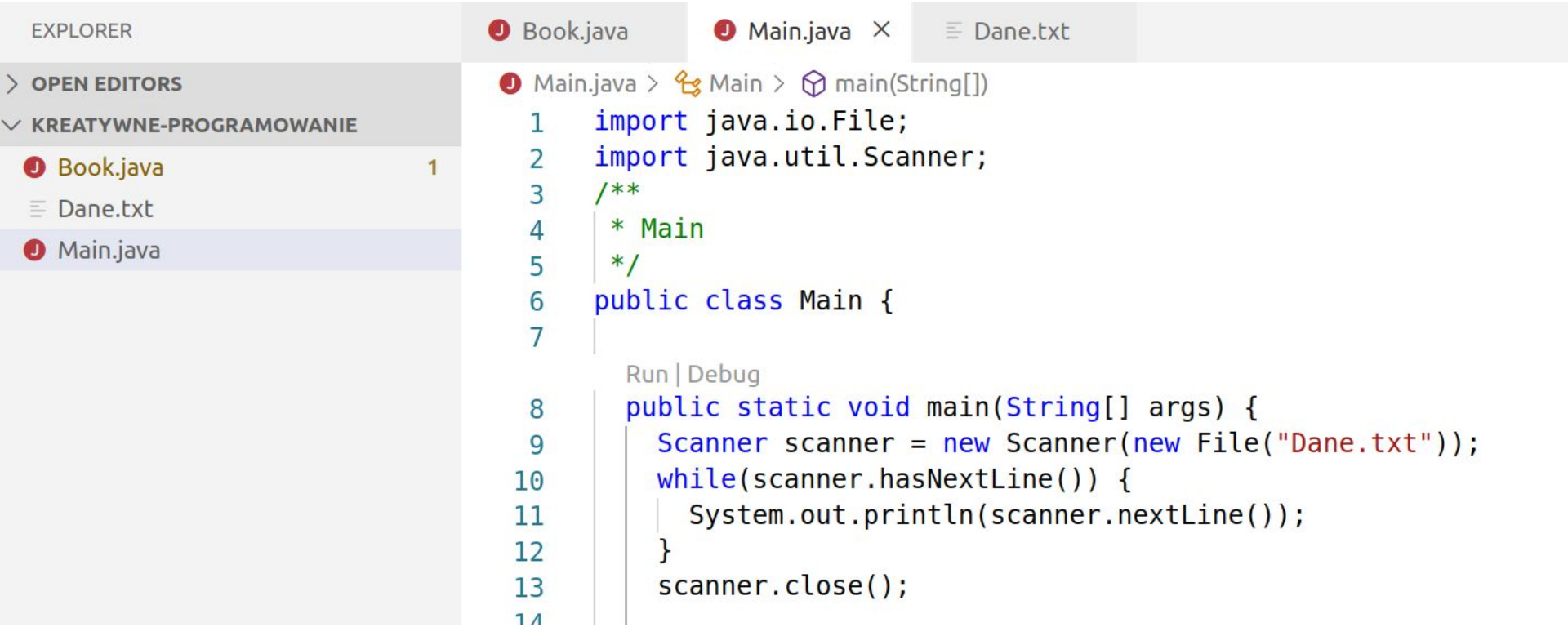
```
public Book(String data) {
  Scanner s = new Scanner(data).useDelimiter(";")
  System.out.println(s);
  this.author = s.next();
  this.title = s.next();                              );
  this.pageAmount = s.nextInt();
  s.close();
}
```

Użycie w klasie main

```
public class Main {

    Run | Debug
    public static void main(String[] args) {
        System.out.println(new Book("abcxc;zzz;12"));
```

Odczyt z pliku Dane.txt



Konstruktor z parsowaniem

```
public class Main {

    Run | Debug
    public static void main(String[] args) {
        System.out.println(new Book("abcxc;zzz;12"));
```

Odczyt książek z pliku

```
public class Main {

    Run | Debug
    public static void main(String[] args) {
        System.out.println(new Book("abcxc;zzz;12"));
        Scanner scanner = new Scanner(new File("Dane.txt"));
        Book book;
        while(scanner.hasNextLine()) {
            book = new Book(scanner.nextLine());
            System.out.println(book);
        }
        scanner.close();
```

Zapis do pliku - Klasa PrintWriter

```
import java.io.PrintWriter;
```

```
public static void main(String[] args) {
  Book book = new Book("Tytuł", "autor", 2);
  PrintWriter pw = new PrintWriter(new File("out.txt"));
  pw.println(book);
  pw.close();
```

Main.java > ...

```java
import java.io.File;
import java.io.PrintWriter;
import java.util.Scanner;
/**
 * Main
 */
public class Main {

    Run | Debug
    public static void main(String[] args){
        PrintWriter pw = new PrintWriter(new File("out.txt"));
        Scanner scanner = new Scanner(new File("./Dane.txt"));
        Book book;
        while(scanner.hasNextLine()) {
            book = new Book(scanner.nextLine());
            pw.println(book);
            System.out.println(book);
        }
        pw.close();
        scanner.close();
    }
}
```

Zadanie - kryptografia

Napisz klasę do szyfrowania/deszyfrowania danych - **Encryptor** (“szyfrator”).

Następnie zakoduj:

odczyt z jednego pliku i zapis po operacji do drugiego pliku.

Dziękuje za uwagę!

Co na następnych zajęciach?

Piotr Radzikowski
ptr.radzikowski@gmail.com
radzikowski.one.pl

Pętle w Javie

Pętle tak samo jak warunki (***if***) wykorzystywane są niemal w każdym programie. Jest to jedna z podstawowych konstrukcji wykorzystywana we wszystkich językach programowania.

Pozwalają one wykonać wielokrotnie, np. wyświetlić 10 razy “hello world”

Możemy wyróżnić takie pętle które pozwolą powtórzyć czynność z góry określoną liczbę razy, oraz niewiadomą ilość razy

Pętla while

Pętlę tą wykorzystujemy w miejscach, gdzie zakładana ilość powtórzeń jest bliżej nieokreślona, ale wiemy kiedy ma przestać ją wykonywać.

```
while (condition) {
  instruction_loop;
}
```

Pętla while

Przykład kodu:

```
public class LoopExample {

  public static void main(String[] args){
    int licznik = 0;

    while (licznik < 10) {
      System.out.println("To jest pętla");
      licznik++;
    }

    System.out.println("Koniec pętli");
  }
}
```

Pętla do ... while

od pętli while różni się zawartość pętli wykona się przynajmniej raz.

```
do {
    instruction_loop;
}
while (condition);
```

Pętla for

Różni się ona od dwóch poprzednich tym że znamy dokładnie ile razy ma się powtórzyć jakaś czynność.

```
for (init_counter_value; condition; modify_counter) {
    instruction_loop;
}
```

Pętla for

```
public class LoopExample {

  public static void main(String[] args) {
    for(int i=0; i<10; i++){
      System.out.println("To jest pętla");
    }
    System.out.println("Koniec pętli");
  }
}
```

Tablice

W języku Java są to obiekty przechowujące sekwencję zmiennych pewnego ustalonego typu.

Deklarując zmienną tablicową podajemy typ zmiennych, które będzie ona przechowywała, oraz wymiar tablicy – wymiar, nie rozmiar!

```
int[] elements = new int[5];
```

Konwersja i rzutowanie typów

```
1  public class Konwersje{
2   public static void main(String[] args){
3        int a = 5;
4        double b = 13.5;
5        int c = (int)b/a;
6        System.out.println(c);
7   }
8  }
```

Klasy osłonowe:

1. Integer.toString(5);
2. Integer.toBinaryString(10);
3. Integer.parseInt(“233”);

Konwersja i rzutowanie typów

Konwersja zamiana typu A na typ B.

```
1    public class Konwersje{
2     public static void main(String[] args){
3          int a = 5;
4          double b = 13.5;
5          double c = b/a;
6          System.out.println(c);
7     }
8    }
```

```
1    public class Konwersje{
2     public static void main(String[] args){
3          int a = 5;
4          double b = 13.5;
5          int c = b/a;
6          System.out.println(c);
7     }
8    }
```

Rzutowanie (jawna konwersja)

Rzutowanie jest w przeciwieństwie do konwersji jawne i mamy na nie wpływ. Możemy wymusić konwersję z jednego typu na inny umieszczając uprzednio w okrągłych nawiasach typ docelowy.

```
1    int a = 5;
2    double b = 13.5;
3    int c = (int)b/a;
```

Konwersję możemy więc przedstawić jako niejawne rzutowanie:

```
1    int a = 5;
2    double b = 13.5;
3    double c = b/(double)a;
```

Typy znakowe

Łączenie ciągów znakowych:

System.out.print(“pierwszy “+”drugi”);

char, czyli znak i służy do reprezentacji pojedynczych znaków kodu Unicode.

Istnieją również znaki specjalne, które muszą być poprzedzone znakiem backslash **** :

- \t - tab
- \n - nowa linia
- \r - powrót karetki
- \" - cudzysłów
- \' - apostrof
- \\ - backslash

Zadanie 1:

Napisz dane na temat książki w formacie:

Książka: “Czysty kod. Podręcznik dobrego programisty”
Autor: “Robert C. Martin”

Unicode - komputerowy zestaw znaków mający w zamierzeniu obejmować wszystkie pisma używane na świecie.