Java Cookbook pre začiatočníkov

Autor: Jakub Jahič

Tento materiál slúži ako referenčná príručka pre účastníkov kurzov Java programovania. Obsahuje kľúčové koncepty, príklady kódu a vysvetlenia od základov až po pokročilé témy.

Ďalšie vzdelávacie materiály a kurzy:

- Moje kurzy na Skillmea - Viaceré kurzy zamerané na rôzne aspekty programovania v Jave
- Java kurz na Street of Code - Komplexný kurz s mentoringom a osobne ohodnotenými zadaniami

Obsah

1. Základy Javy

1.1. Premenné a dátové typy

```
// Primitívne dátové typy
byte byteValue = 127;          // 8-bit, rozsah -128 až 127
short shortValue = 32767;      // 16-bit, rozsah -32,768 až 32,767
int intValue = 2147483647;     // 32-bit, rozsah -2^31 až 2^31-1
long longValue = 92233720368547758O7L;  // 64-bit, veľmi veľký rozsah, L na konci

float floatValue = 3.14f;      // 32-bit desatinné číslo, f na konci
double doubleValue = 3.14159;  // 64-bit desatinné číslo (presnejšie ako float)

char charValue = 'A';          // 16-bit Unicode znak
boolean boolValue = true;      // true alebo false

// Referenčné dátové typy
String name = "Jakub";         // Textový reťazec
int[] numbers = {1, 2, 3};     // Pole čísiel

// Deklarácia konštanty (nemenná hodnota)
final double PI = 3.14159;

// Zobrazenie hodnoty premennej
System.out.println(name);      // Vypíše: Jakub

// Spájanie textov
System.out.println("Volam sa " + name + " a mam rokov: " + intValue);
```

1.2. Operátory

```java
// Aritmetické operátory
int a = 10, b = 3;
System.out.println("a + b = " + (a + b));    // Sčítanie: 13
System.out.println("a - b = " + (a - b));    // Odčítanie: 7
System.out.println("a * b = " + (a * b));    // Násobenie: 30
System.out.println("a / b = " + (a / b));    // Delenie: 3 (celočíselné)
System.out.println("a % b = " + (a % b));    // Zvyšok po delení (modulo): 1

// Priradenie s operáciou
int c = 5;
c += 3;  // Ekvivalent: c = c + 3; Výsledok: 8
c -= 2;  // Ekvivalent: c = c - 2; Výsledok: 6
c *= 2;  // Ekvivalent: c = c * 2; Výsledok: 12
c /= 4;  // Ekvivalent: c = c / 4; Výsledok: 3
c %= 2;  // Ekvivalent: c = c % 2; Výsledok: 1

// Inkrementácia a dekrementácia
int i = 5;
System.out.println(i++);  // Post-inkrementácia: vypíše 5, potom zvýši na 6
System.out.println(++i);  // Pre-inkrementácia: zvýši na 7, potom vypíše 7
System.out.println(i--);  // Post-dekrementácia: vypíše 7, potom zníži na 6
System.out.println(--i);  // Pre-dekrementácia: zníži na 5, potom vypíše 5

// Porovnávacie operátory
int x = 5, y = 10;
System.out.println(x == y);    // Rovná sa: false
System.out.println(x != y);    // Nerovná sa: true
System.out.println(x < y);     // Menšie ako: true
System.out.println(x <= y);    // Menšie alebo rovné: true
System.out.println(x > y);     // Väčšie ako: false
System.out.println(x >= y);    // Väčšie alebo rovné: false

// Logické operátory
boolean p = true, q = false;
System.out.println(p && q);    // Logické AND: false (oba musia byť true)
System.out.println(p || q);    // Logické OR: true (aspoň jeden musí byť true)
System.out.println(!p);        // Logické NOT: false (negácia)
```

1.3. Reťazce a manipulácia

```java
// Vytvorenie reťazcov
String greeting = "Hello";
String name = "World";
String message = greeting + " " + name;  // Spájanie: "Hello World"

// Základné operácie
System.out.println(message.length());          // Dĺžka: 11
System.out.println(message.toUpperCase());     // "HELLO WORLD"
System.out.println(message.toLowerCase());     // "hello world"
```

```java
// Vyhľadávanie
System.out.println(message.indexOf("World"));  // 6 (pozícia, kde začína "World")
System.out.println(message.contains("Hello")); // true
System.out.println(message.startsWith("He"));  // true
System.out.println(message.endsWith("ld"));    // true

// Extrakcia podreťazcov
System.out.println(message.substring(0, 5));   // "Hello"
System.out.println(message.substring(6));      // "World"

// Nahrádzanie
System.out.println(message.replace("World", "Java")); // "Hello Java"

// Rozdelenie a spájanie reťazcov
String csv = "apple,banana,orange";
String[] fruits = csv.split(",");  // ["apple", "banana", "orange"]

String joined = String.join("-", fruits);  // "apple-banana-orange"

// Odstránenie whitespace
String padded = "  trim me  ";
System.out.println(padded.trim());  // "trim me"

// Porovnávanie
String s1 = "hello";
String s2 = "HELLO";
System.out.println(s1.equals(s2));           // false
System.out.println(s1.equalsIgnoreCase(s2)); // true

// StringBuilder - efektívna manipulácia s reťazcami
StringBuilder sb = new StringBuilder();
sb.append("Hello");
sb.append(" ");
sb.append("World");
String result = sb.toString();  // "Hello World"

// Ďalšie operácie so StringBuilder
sb.insert(5, ",");      // "Hello, World"
sb.delete(5, 7);        // "HelloWorld"
sb.reverse();           // "dlroWolleH"
sb.replace(0, 5, "Hi"); // "HiroWd"
```

1.4. Vstup od užívateľa

```java
import java.util.Scanner;    // Import potrebný pre Scanner

// Vytvorenie Scanner objektu na načítanie vstupu
Scanner scanner = new Scanner(System.in);

// Načítanie textu
System.out.println("Zadaj svoje meno: ");
```

```
String name = scanner.nextLine();
System.out.println("Ahoj " + name);

// Načítanie celého čísla
System.out.println("Zadaj svoj vek: ");
int age = scanner.nextInt();
System.out.println("Mas " + age + " rokov");

// Načítanie desatinného čísla
System.out.println("Zadaj svoju výšku v metroch: ");
double height = scanner.nextDouble();
System.out.println("Tvoja výška je: " + height + " m");

// Po použití scanner.nextInt() alebo scanner.nextDouble() a pred
scanner.nextLine()
// je vhodné vyčistiť buffer
scanner.nextLine();  // Vyčistenie bufferu

// Načítanie ďalšieho textu
System.out.println("Zadaj svoj obľúbený film: ");
String favoriteMovie = scanner.nextLine();

// Zatvorenie scanner objektu po použití
scanner.close();
```

1.5. Polia (Arrays)

```
// Deklarácia a inicializácia jednorozmerného poľa
int[] numbers = {1, 2, 3, 4, 5};

// Alternatívny spôsob deklarácie poľa
int numbers2[] = new int[5];  // Vytvorí pole s 5 prvkami (všetky sú 0)
numbers2[0] = 10;  // Nastavenie hodnoty prvého prvku
numbers2[1] = 20;
numbers2[2] = 30;
numbers2[3] = 40;
numbers2[4] = 50;

// Pristupovanie k prvkom poľa
System.out.println("Prvý prvok: " + numbers[0]);  // Indexy začínajú od 0
System.out.println("Tretí prvok: " + numbers[2]);

// Zistenie dĺžky poľa
System.out.println("Dĺžka poľa: " + numbers.length);

// Prechádzanie poľa
for (int i = 0; i < numbers.length; i++) {
    System.out.println("Prvok " + i + ": " + numbers[i]);
}

// Prechádzanie poľa pomocou enhanced for loop (foreach)
```

```java
for (int number : numbers) {
    System.out.println("Hodnota: " + number);
}

// Dvojrozmerné pole (matica)
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

// Alternatívny spôsob deklarácie
int[][] matrix2 = new int[3][3];  // 3x3 matica
matrix2[0][0] = 1;
matrix2[0][1] = 2;
// ... a tak ďalej

// Pristupovanie k prvkom dvojrozmerného poľa
System.out.println("Prvok v 2. riadku a 3. stĺpci: " + matrix[1][2]);  // Vypíše 6

// Prechádzanie dvojrozmerného poľa
for (int row = 0; row < matrix.length; row++) {
    for (int col = 0; col < matrix[row].length; col++) {
        System.out.print(matrix[row][col] + " ");
    }
    System.out.println();  // Nový riadok po každom riadku matice
}

// Prechádzanie dvojrozmerného poľa pomocou enhanced for loop
for (int[] row : matrix) {
    for (int value : row) {
        System.out.print(value + " ");
    }
    System.out.println();
}
```

1.6. Typové konverzie (Casting)

```java
// Automatická konverzia (rozšírenie, widening, implicit casting)
byte byteValue = 100;
short shortValue = byteValue;  // byte -> short
int intValue = shortValue;     // short -> int
long longValue = intValue;     // int -> long
float floatValue = longValue;  // long -> float
double doubleValue = floatValue;  // float -> double

// Explicitná konverzia (zúženie, narrowing, explicit casting)
double pi = 3.14159;
float piFloat = (float) pi;       // double -> float
long piLong = (long) pi;          // double -> long (strata desatinnej časti)
int piInt = (int) pi;             // double -> int (strata desatinnej časti)
```

```
short piShort = (short) piInt;    // int -> short
byte piByte = (byte) piShort;     // short -> byte

// Konverzie s číslami
String numberStr = "123";
int number = Integer.parseInt(numberStr);  // String -> int

String doubleStr = "3.14";
double doubleNumber = Double.parseDouble(doubleStr);  // String -> double

// Konverzia čísla na String
int x = 42;
String xStr = String.valueOf(x);  // int -> String
String xStr2 = Integer.toString(x);  // int -> String (alternatívny spôsob)

// Konverzia medzi objektami a primitívnymi typmi
Integer integerObj = 100;          // autoboxing (int -> Integer)
int primitiveInt = integerObj;     // unboxing (Integer -> int)

// Konverzia objektov
Object obj = "Hello";
String str = (String) obj;  // Cast na String
// Ak obj nie je String, vyhodí ClassCastException
```

2. Riadiace štruktúry

2.1. If-Else podmienky

```
// Jednoduchá podmienka
int age = 20;
if (age >= 18) {
    System.out.println("Si plnolety");
} else {
    System.out.println("Nie si plnolety");
}

// Podmienka s viacerými vetvami
int score = 85;
if (score >= 90) {
    System.out.println("Výborne (A)");
} else if (score >= 80) {
    System.out.println("Veľmi dobre (B)");
} else if (score >= 70) {
    System.out.println("Dobre (C)");
} else if (score >= 60) {
    System.out.println("Dostatočne (D)");
} else {
    System.out.println("Nedostatočne (F)");
}

// Vnorené podmienky
```

```java
boolean isWeekend = true;
boolean isRaining = false;

if (isWeekend) {
    if (isRaining) {
        System.out.println("Zostanem doma a budem pozerať film");
    } else {
        System.out.println("Idem von s kamarátmi");
    }
} else {
    System.out.println("Musím ísť do práce/školy");
}

// Logické operátory v podmienkach
int temperature = 25;
if (temperature > 30 && !isRaining) {
    System.out.println("Ideálne počasie na kúpalisko");
} else if (temperature > 20 || isWeekend) {
    System.out.println("Môžeme ísť na prechádzku");
}
```

2.2. Switch a Enum

```java
// Základný switch s int
int day = 3;
switch (day) {
    case 1:
        System.out.println("Pondelok");
        break;
    case 2:
        System.out.println("Utorok");
        break;
    case 3:
        System.out.println("Streda");
        break;
    case 4:
        System.out.println("Štvrtok");
        break;
    case 5:
        System.out.println("Piatok");
        break;
    case 6:
    case 7:
        System.out.println("Víkend");
        break;
    default:
        System.out.println("Neplatný deň");
        break;
}

// Switch s String (od Java 7)
```

```
String fruit = "apple";
switch (fruit) {
    case "apple":
        System.out.println("Jablko je červené");
        break;
    case "banana":
        System.out.println("Banán je žltý");
        break;
    case "orange":
        System.out.println("Pomaranč je oranžový");
        break;
    default:
        System.out.println("Neznáme ovocie");
        break;
}

// Deklarácia Enum
enum DayOfWeek {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}

// Použitie Enum v switch
DayOfWeek today = DayOfWeek.WEDNESDAY;
switch (today) {
    case MONDAY:
        System.out.println("Začiatok týždňa");
        break;
    case FRIDAY:
        System.out.println("Koniec pracovného týždňa");
        break;
    case SATURDAY:
    case SUNDAY:
        System.out.println("Víkend");
        break;
    default:
        System.out.println("Pracovný deň");
        break;
}

// Switch expression (od Java 12, štandardizovaný v Java 14)
// Nevyžaduje break a môže priamo vrátiť hodnotu
String dayType = switch (today) {
    case MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY -> "Pracovný deň";
    case SATURDAY, SUNDAY -> "Víkend";
};
System.out.println(dayType);
```

2.3. Ternárny operátor

```
// Základné použitie ternárneho operátora
int age = 20;
```

```
String status = (age >= 18) ? "dospelý" : "neplnoletý";
System.out.println(status);  // Vypíše: dospelý

// Namiesto if-else
int a = 5, b = 10;
int max = (a > b) ? a : b;
System.out.println("Väčšie číslo je: " + max);  // Vypíše: 10

// Vnorený ternárny operátor (menej prehľadný, používať opatrne)
int x = 15;
String result = (x > 10) ? ((x > 20) ? "väčšie ako 20" : "medzi 10 a 20") :
"menšie ako 10";
System.out.println(result);  // Vypíše: medzi 10 a 20

// Ternárny operátor v priradení
boolean hasDiscount = true;
double price = 100.0;
double finalPrice = hasDiscount ? price * 0.9 : price;
System.out.println("Konečná cena: " + finalPrice);  // Vypíše: 90.0
```

3. Cykly

3.1. For cyklus

```
// Základný for cyklus
for (int i = 1; i <= 5; i++) {
    System.out.println("Iterácia č. " + i);
}

// For cyklus s zostupným počítaním
for (int i = 10; i >= 1; i--) {
    System.out.println("Odpočet: " + i);
}

// For cyklus s viacerými premennými
for (int i = 0, j = 10; i < j; i++, j--) {
    System.out.println("i = " + i + ", j = " + j);
}

// For cyklus bez tela
int sum = 0;
for (int i = 1; i <= 100; sum += i, i++);
System.out.println("Súčet čísel 1-100: " + sum);

// Vnorený for cyklus - vykreslenie trojuholníka
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("* ");
    }
    System.out.println();
}
```

```java
// For-each cyklus (enhanced for loop)
int[] numbers = {1, 2, 3, 4, 5};
for (int number : numbers) {
    System.out.println("Hodnota: " + number);
}

String[] names = {"Anna", "Tomáš", "Jana"};
for (String name : names) {
    System.out.println("Meno: " + name);
}
```

3.2. While cyklus

```java
// Základný while cyklus
int i = 1;
while (i <= 5) {
    System.out.println("Číslo: " + i);
    i++;
}

// While cyklus s podmienkou na konci
int j = 10;
while (j > 0) {
    System.out.println("Odpočet: " + j);
    j--;
}

// While s nekonečným cyklom a break
while (true) {
    // Kód, ktorý sa bude opakovať
    System.out.println("Nekonečný cyklus");

    // Podmienka ukončenia
    if (Math.random() > 0.9) {  // 10% šanca na ukončenie
        System.out.println("Ukončenie cyklu");
        break;
    }
}

// Použitie continue na preskočenie iterácie
int k = 0;
while (k < 10) {
    k++;
    if (k % 2 == 0) {  // Ak je k párne
        continue;      // Preskoč zvyšok kódu v tejto iterácii
    }
    System.out.println("Nepárne číslo: " + k);
}
```

Príklad hry "Hádaj číslo" s while cyklom:

```
import java.util.Scanner;
import java.util.Random;

public class GuessNumber {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();
        int secretNumber = random.nextInt(100) + 1;  // Číslo od 1 do 100
        int attempts = 0;
        boolean guessed = false;

        System.out.println("Myslím si číslo od 1 do 100. Uhádni ho!");

        while (!guessed) {
            System.out.print("Tvoj tip: ");
            int guess = scanner.nextInt();
            attempts++;

            if (guess < secretNumber) {
                System.out.println("Príliš nízke číslo!");
            } else if (guess > secretNumber) {
                System.out.println("Príliš vysoké číslo!");
            } else {
                guessed = true;
                System.out.println("Správne! Uhádol si po " + attempts + "
pokusoch!");
            }
        }

        scanner.close();
    }
}
```

3.3. Do-While cyklus

```
// Základný do-while cyklus - vykoná sa minimálne raz
int i = 1;
do {
    System.out.println("Iterácia č. " + i);
    i++;
} while (i <= 5);

// Do-while pre validáciu vstupu
import java.util.Scanner;

Scanner scanner = new Scanner(System.in);
int number;
```

```
do {
    System.out.print("Zadaj číslo medzi 1 a 10: ");
    number = scanner.nextInt();
} while (number < 1 || number > 10);

System.out.println("Zadal si platné číslo: " + number);
```

4. Metódy

4.1. Deklarácia metód

```
// Metóda bez parametrov a bez návratovej hodnoty (void)
public static void sayHello() {
    System.out.println("Hello World!");
}

// Volanie metódy
sayHello();  // Vypíše: Hello World!

// Metóda s jedným parametrom
public static void greetPerson(String name) {
    System.out.println("Hello, " + name + "!");
}

greetPerson("John");  // Vypíše: Hello, John!

// Metóda s viacerými parametrami
public static void displayInfo(String name, int age) {
    System.out.println(name + " má " + age + " rokov.");
}

displayInfo("Anna", 25);  // Vypíše: Anna má 25 rokov.
```

4.2. Návratové hodnoty

```
// Metóda s návratovou hodnotou
public static int add(int a, int b) {
    return a + b;
}

// Použitie návratovej hodnoty
int result = add(5, 3);
System.out.println("5 + 3 = " + result);  // Vypíše: 5 + 3 = 8

// Metóda s návratovou hodnotou typu boolean
public static boolean isEven(int number) {
    return number % 2 == 0;
}
```

```java
if (isEven(10)) {
    System.out.println("Číslo je párne");
}

// Metóda s návratovou hodnotou typu String
public static String formatName(String firstName, String lastName) {
    return lastName + ", " + firstName;
}

String formattedName = formatName("John", "Doe");
System.out.println(formattedName);  // Vypíše: Doe, John

// Metóda s návratom poľa
public static int[] generateNumbers(int count) {
    int[] numbers = new int[count];
    for (int i = 0; i < count; i++) {
        numbers[i] = i + 1;
    }
    return numbers;
}

int[] generatedNumbers = generateNumbers(5);  // Vráti pole [1, 2, 3, 4, 5]
```

```java
// Metóda s primitívnym typom parametra (pass by value)
public static void incrementValue(int value) {
    value++;  // Zmení sa len lokálna kópia parametra
    System.out.println("Hodnota vo vnútri metódy: " + value);
}

int x = 10;
incrementValue(x);
System.out.println("Hodnota po volaní metódy: " + x);  // Stále 10

// Metóda s objektovým typom parametra (pass by reference)
public static void modifyArray(int[] array) {
    array[0] = 100;  // Zmení sa originálne pole
}

int[] numbers = {1, 2, 3};
modifyArray(numbers);
System.out.println("Prvý prvok poľa po volaní metódy: " + numbers[0]);  // 100

// Metóda s variabilným počtom parametrov (varargs)
public static int sum(int... numbers) {
    int total = 0;
    for (int num : numbers) {
        total += num;
    }
    return total;
}
```

```
System.out.println(sum(1, 2, 3));       // 6
System.out.println(sum(1, 2, 3, 4, 5)); // 15
```

4.3. Parametre

```
// Metóda s primitívnym typom parametra (pass by value)
public static void incrementValue(int value) {
    value++;  // Zmení sa len lokálna kópia parametra
    System.out.println("Hodnota vo vnútri metódy: " + value);
}

int x = 10;
incrementValue(x);
System.out.println("Hodnota po volaní metódy: " + x);  // Stále 10

// Metóda s objektovým typom parametra (pass by reference)
public static void modifyArray(int[] array) {
    array[0] = 100;  // Zmení sa originálne pole
}

int[] numbers = {1, 2, 3};
modifyArray(numbers);
System.out.println("Prvý prvok poľa po volaní metódy: " + numbers[0]);  // 100

// Metóda s variabilným počtom parametrov (varargs)
public static int sum(int... numbers) {
    int total = 0;
    for (int num : numbers) {
        total += num;
    }
    return total;
}

System.out.println(sum(1, 2, 3));       // 6
System.out.println(sum(1, 2, 3, 4, 5)); // 15

// Metóda s varargs a normálnymi parametrami
public static void printInfo(String name, int... scores) {
    System.out.println("Meno: " + name);
    System.out.print("Hodnotenia: ");
    for (int score : scores) {
        System.out.print(score + " ");
    }
    System.out.println();
}

printInfo("Anna", 85, 90, 78);  // Meno: Anna, Hodnotenia: 85 90 78
```

5. Práca so súbormi

5.1. Čítanie súborov

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

// Čítanie súboru pomocou Scanner
public static void readFileWithScanner() {
    try {
        File file = new File("example.txt");
        Scanner scanner = new Scanner(file);

        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            System.out.println(line);
        }

        scanner.close();
    } catch (FileNotFoundException e) {
        System.out.println("Súbor sa nenašiel: " + e.getMessage());
    }
}

// Čítanie súboru pomocou BufferedReader
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public static void readFileWithBufferedReader() {
    try (BufferedReader reader = new BufferedReader(new
FileReader("example.txt"))) {
        String line;
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        System.out.println("Chyba pri čítaní súboru: " + e.getMessage());
    }
}

// Čítanie celého súboru do reťazca (Java 11+)
import java.nio.file.Files;
import java.nio.file.Path;

public static void readFileAsString() {
    try {
        String content = Files.readString(Path.of("example.txt"));
        System.out.println(content);
    } catch (IOException e) {
        System.out.println("Chyba pri čítaní súboru: " + e.getMessage());
    }
}
```

```java
// Zistenie, či súbor existuje
File file = new File("example.txt");
if (file.exists()) {
    System.out.println("Súbor existuje!");
} else {
    System.out.println("Súbor neexistuje!");
}

// Získanie informácií o súbore
if (file.exists()) {
    System.out.println("Názov súboru: " + file.getName());
    System.out.println("Absolútna cesta: " + file.getAbsolutePath());
    System.out.println("Veľkosť súboru: " + file.length() + " bajtov");
    System.out.println("Je to súbor: " + file.isFile());
    System.out.println("Je to adresár: " + file.isDirectory());
}
```

5.2. Zápis do súborov

```java
import java.io.FileWriter;
import java.io.IOException;

// Zápis do súboru pomocou FileWriter
public static void writeToFile() {
    try (FileWriter writer = new FileWriter("output.txt")) {
        writer.write("Hello World!\n");
        writer.write("Toto je druhý riadok.\n");
        writer.write("Toto je tretí riadok.");
    } catch (IOException e) {
        System.out.println("Chyba pri zápise do súboru: " + e.getMessage());
    }
}

// Pripojenie textu na koniec existujúceho súboru
public static void appendToFile() {
    try (FileWriter writer = new FileWriter("output.txt", true)) {  // true pre
append mód
        writer.write("\nToto je pridaný text.");
    } catch (IOException e) {
        System.out.println("Chyba pri zápise do súboru: " + e.getMessage());
    }
}

// Zápis do súboru pomocou BufferedWriter
import java.io.BufferedWriter;

public static void writeWithBufferedWriter() {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt")))
{
        writer.write("Riadok 1");
```

```
            writer.newLine();  // Vloží nový riadok
            writer.write("Riadok 2");
            writer.newLine();
            writer.write("Riadok 3");
        } catch (IOException e) {
            System.out.println("Chyba pri zápise do súboru: " + e.getMessage());
        }
    }

    // Zápis celého reťazca do súboru (Java 11+)
    import java.nio.file.Files;
    import java.nio.file.Path;

    public static void writeStringToFile() {
        try {
            String content = "Hello World!\nToto je druhý riadok.";
            Files.writeString(Path.of("output.txt"), content);
        } catch (IOException e) {
            System.out.println("Chyba pri zápise do súboru: " + e.getMessage());
        }
    }

    // Vytvorenie nového súboru
    public static void createNewFile() {
        try {
            File file = new File("newfile.txt");
            if (file.createNewFile()) {
                System.out.println("Súbor bol vytvorený: " + file.getName());
            } else {
                System.out.println("Súbor už existuje.");
            }
        } catch (IOException e) {
            System.out.println("Chyba pri vytváraní súboru: " + e.getMessage());
        }
    }

    // Vymazanie súboru
    public static void deleteFile() {
        File file = new File("toDelete.txt");
        if (file.delete()) {
            System.out.println("Súbor bol vymazaný: " + file.getName());
        } else {
            System.out.println("Súbor sa nepodarilo vymazať.");
        }
    }
```

6. Základy OOP

6.1 Triedy a Objekty

```
```

```java
// Definícia triedy s atribútmi a metódami
public class BankAccount {
    // Atribúty (inštančné premenné) - enkapsulované pomocou private
    private String accountNumber;
    private String ownerName;
    private double balance;
    private boolean isLocked;

    // Konštruktor - volá sa pri vytvorení objektu
    public BankAccount(String accountNumber, String ownerName) {
        this.accountNumber = accountNumber;
        this.ownerName = ownerName;
        this.balance = 0.0;
        this.isLocked = false;
    }

    // Preťažený konštruktor (overloaded constructor)
    public BankAccount(String accountNumber, String ownerName, double
initialBalance) {
        this.accountNumber = accountNumber;
        this.ownerName = ownerName;
        this.balance = initialBalance;
        this.isLocked = false;
    }

    // Biznis logika - metódy pracujúce s dátami
    public void deposit(double amount) {
        if (isLocked) {
            System.out.println("Účet je zamknutý. Operácia nemôže byť vykonaná.");
            return;
        }

        if (amount <= 0) {
            System.out.println("Čiastka musí byť kladná.");
            return;
        }

        this.balance += amount;
        System.out.println("Vklad: " + amount + " | Nový zostatok: " +
this.balance);
    }

    public boolean withdraw(double amount) {
        if (isLocked) {
            System.out.println("Účet je zamknutý. Operácia nemôže byť vykonaná.");
            return false;
        }

        if (amount <= 0) {
            System.out.println("Čiastka musí byť kladná.");
            return false;
        }
```

```
        if (amount > balance) {
            System.out.println("Nedostatok prostriedkov.");
            return false;
        }

        this.balance -= amount;
        System.out.println("Výber: " + amount + " | Nový zostatok: " +
this.balance);
        return true;
    }

    public void lockAccount() {
        this.isLocked = true;
        System.out.println("Účet bol zamknutý.");
    }

    public void unlockAccount() {
        this.isLocked = false;
        System.out.println("Účet bol odomknutý.");
    }

    // Getter metódy (kontrolovaný prístup k dátam)
    public String getAccountNumber() {
        return accountNumber;
    }

    public String getOwnerName() {
        return ownerName;
    }

    public double getBalance() {
        return balance;
    }

    public boolean isLocked() {
        return isLocked;
    }
}

// Použitie triedy a vytvorenie objektov
public class Main {
    public static void main(String[] args) {
        // Vytvorenie objektu pomocou konštruktora
        BankAccount account1 = new BankAccount("SK123456789", "Ján Novák");

        // Volanie metód objektu
        account1.deposit(1000);
        account1.withdraw(500);

        // Zobrazenie stavu objektu pomocou getterov
        System.out.println("Majiteľ účtu: " + account1.getOwnerName());
        System.out.println("Číslo účtu: " + account1.getAccountNumber());
        System.out.println("Zostatok: " + account1.getBalance());
```

```
        // Vytvorenie druhého objektu s preťaženým konštruktorom
        BankAccount account2 = new BankAccount("SK987654321", "Eva Nová", 2000);
        System.out.println("Zostatok na účte " + account2.getAccountNumber() + ":
" + account2.getBalance());
    }
}
```

6.2 Dedičnosť

```
// Základná (rodičovská) trieda
public class Vehicle {
    protected String brand;  // protected - prístupné v podtriedach
    protected String model;
    protected int year;

    public Vehicle(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    public void start() {
        System.out.println("Vozidlo sa štartuje");
    }

    public void stop() {
        System.out.println("Vozidlo sa zastavuje");
    }

    public String getInfo() {
        return brand + " " + model + " (" + year + ")";
    }
}

// Podtrieda (dedí od Vehicle)
public class Car extends Vehicle {
    private int numberOfDoors;
    private String fuelType;

    // Konštruktor podtriedy volá konštruktor nadtriedy pomocou super()
    public Car(String brand, String model, int year, int numberOfDoors, String
fuelType) {
        super(brand, model, year);  // Volá konštruktor rodičovskej triedy
        this.numberOfDoors = numberOfDoors;
        this.fuelType = fuelType;
    }

    // Rozšírenie funkcionality
    public void honk() {
        System.out.println("Tút tút!");
    }
```

```
    // Prekrytie (override) metódy z rodičovskej triedy
    @Override
    public void start() {
        System.out.println("Auto štartuje kľúčom");
        super.start();  // Volanie metódy rodičovskej triedy
    }

    // Rozšírenie getInfo metódy
    @Override
    public String getInfo() {
        return super.getInfo() + " - " + numberOfDoors + "-dverové, palivo: " +
fuelType;
    }
}

// Ďalšia podtrieda
public class ElectricCar extends Car {
    private int batteryCapacity;
    private int range;

    public ElectricCar(String brand, String model, int year, int numberOfDoors,
                       int batteryCapacity, int range) {
        super(brand, model, year, numberOfDoors, "Elektrina");
        this.batteryCapacity = batteryCapacity;
        this.range = range;
    }

    public void charge() {
        System.out.println("Elektromobil sa nabíja");
    }

    @Override
    public void start() {
        System.out.println("Elektromobil štartuje tlačidlom");
        // Nevolá sa super.start() - úplne prekrýva metódu
    }

    @Override
    public String getInfo() {
        return super.getInfo() + ", batéria: " + batteryCapacity +
               " kWh, dojazd: " + range + " km";
    }
}

// Použitie dedičnosti
public class InheritanceExample {
    public static void main(String[] args) {
        // Vytvorenie objektov
        Vehicle vehicle = new Vehicle("Neznáma značka", "Obecné vozidlo", 2020);
        Car car = new Car("Škoda", "Octavia", 2021, 5, "Diesel");
        ElectricCar electricCar = new ElectricCar("Tesla", "Model 3", 2022, 4, 75,
500);
```

```
        // Volanie metód
        System.out.println(vehicle.getInfo());
        vehicle.start();

        System.out.println("\n" + car.getInfo());
        car.start();
        car.honk();

        System.out.println("\n" + electricCar.getInfo());
        electricCar.start();
        electricCar.charge();

        // Polymorfizmus - prístup cez rodičovský typ
        Vehicle carAsVehicle = car;  // Car môže byť použité ako Vehicle
        Vehicle electricAsVehicle = electricCar;  // ElectricCar môže byť použité
ako Vehicle

        System.out.println("\nPolymorfizmus:");
        displayVehicleInfo(vehicle);
        displayVehicleInfo(car);
        displayVehicleInfo(electricCar);
    }

    // Metóda očakáva Vehicle, ale môže prijať aj jeho podtriedy
    public static void displayVehicleInfo(Vehicle v) {
        System.out.println("Info o vozidle: " + v.getInfo());
        v.start();

        // Zistenie skutočného typu objektu
        if (v instanceof ElectricCar) {
            ElectricCar ec = (ElectricCar) v;  // Type casting
            ec.charge();
        } else if (v instanceof Car) {
            Car c = (Car) v;  // Type casting
            c.honk();
        }

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

6.3 Rozhrania (interfaces)

```
// Definícia rozhrania - všetky metódy sú automaticky public a abstraktné
public interface PaymentProcessor {
    // Deklarácia metód (bez implementácie)
    boolean processPayment(double amount);
    String getPaymentMethod();
    boolean supportsRefunds();
    double getTransactionFee(double amount);
}
```

```java
// Implementácia rozhrania CreditCardProcessor
public class CreditCardProcessor implements PaymentProcessor {
    private String cardNumber;
    private String expiryDate;

    public CreditCardProcessor(String cardNumber, String expiryDate) {
        this.cardNumber = cardNumber;
        this.expiryDate = expiryDate;
    }

    @Override
    public boolean processPayment(double amount) {
        // Implementácia spracovania platby kartou
        System.out.println("Spracovanie platby kartou: " + amount + " EUR");
        return true;
    }

    @Override
    public String getPaymentMethod() {
        return "Kreditná karta";
    }

    @Override
    public boolean supportsRefunds() {
        return true;
    }

    @Override
    public double getTransactionFee(double amount) {
        // 1.5% transakčný poplatok, minimum 0.50 EUR
        return Math.max(0.50, amount * 0.015);
    }
}

// Implementácia rozhrania BankTransferProcessor
public class BankTransferProcessor implements PaymentProcessor {
    private String accountNumber;
    private String bankCode;

    public BankTransferProcessor(String accountNumber, String bankCode) {
        this.accountNumber = accountNumber;
        this.bankCode = bankCode;
    }

    @Override
    public boolean processPayment(double amount) {
        // Implementácia spracovania platby bankovým prevodom
        System.out.println("Spracovanie bankového prevodu: " + amount + " EUR");
        return true;
    }

    @Override
    public String getPaymentMethod() {
```

```
        return "Bankový prevod";
    }

    @Override
    public boolean supportsRefunds() {
        return true;
    }

    @Override
    public double getTransactionFee(double amount) {
        // Fixný poplatok 1 EUR
        return 1.0;
    }
}

// Použitie polymorfizmu cez rozhranie
public class PaymentExample {
    public static void main(String[] args) {
        // Vytvorenie objektov rôznych platobných metód
        PaymentProcessor creditCard = new CreditCardProcessor("1234-5678-9012-
3456", "12/25");
        PaymentProcessor bankTransfer = new BankTransferProcessor("SK123456789",
"1100");

        // Práca s objektami cez spoločné rozhranie (polymorfizmus)
        processUserPayment(creditCard, 100.0);
        processUserPayment(bankTransfer, 100.0);
    }

    // Metóda pracuje s akýmkoľvek objektom, ktorý implementuje PaymentProcessor
    public static void processUserPayment(PaymentProcessor processor, double
amount) {
        System.out.println("Platobná metóda: " + processor.getPaymentMethod());
        double fee = processor.getTransactionFee(amount);
        System.out.println("Transakčný poplatok: " + fee + " EUR");
        boolean success = processor.processPayment(amount);
        System.out.println("Platba " + (success ? "úspešná" : "zamietnutá"));
        System.out.println("Podporuje vrátenie peňazí: " +
(processor.supportsRefunds() ? "Áno" : "Nie"));
        System.out.println();
    }
}
```

6.4 Records

```
// Records boli pridané v Java 14 (2020) ako preview feature a štandardizované v
Java 16 (2021)
// Record je špeciálny typ triedy určený na uchovávanie dát (data carrier)

// Tradičný spôsob bez Records - veľa kódu na jednoduchý účel
public class Person {
```

```java
    private final String name;
    private final int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        return age == person.age && Objects.equals(name, person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}

// Rovnaké pomocou Java Record - oveľa stručnejšie
public record PersonRecord(String name, int age) {
    // To je všetko! Konštruktor, gettery, equals, hashCode
    // a toString sú generované automaticky

    // Môžete pridať vlastné metódy
    public boolean isAdult() {
        return age >= 18;
    }

    // Môžete pridať validačný konštruktor
    public PersonRecord {
        if (age < 0) {
            throw new IllegalArgumentException("Vek nemôže byť záporný");
        }
    }
}

// Použitie
```

```
public class RecordExample {
    public static void main(String[] args) {
        PersonRecord john = new PersonRecord("John", 25);
        System.out.println(john.name());    // Getter: John
        System.out.println(john.age());     // Getter: 25
        System.out.println(john);           // ToString: PersonRecord[name=John,
age=25]
        System.out.println(john.isAdult()); // Vlastná metóda: true

        // Porovnanie objektov (automatický equals)
        PersonRecord john2 = new PersonRecord("John", 25);
        System.out.println(john.equals(john2));  // true

        // Použitie lokálneho odvodenia typov premenných (var) - pridané v Java 10
(2018)
        var jane = new PersonRecord("Jane", 30);
        var isAdult = jane.isAdult();  // boolean
    }
}
```

6.5. Vzťahy medzi objektami

Kompozícia

Keď jeden objekt obsahuje druhý ako svoju časť:

```
// Trieda Lecture
public class Lecture {
    private String title;
    private int duration;  // v minútach

    public Lecture(String title, int duration) {
        this.title = title;
        this.duration = duration;
    }

    public void printInfo() {
        System.out.println("Lecture: " + title + " (" + duration + " minutes)");
    }
}

// Trieda Chapter obsahuje Lecture (kompozícia)
public class Chapter {
    private String title;
    private Lecture lecture;  // Chapter obsahuje Lecture

    public Chapter(String title, Lecture lecture) {
        this.title = title;
        this.lecture = lecture;
    }

    public void printInfo() {
```

```
        System.out.println("Chapter: " + title);
        lecture.printInfo();
    }
}

// Použitie
Lecture lecture = new Lecture("Java Basics", 45);
Chapter chapter = new Chapter("Introduction", lecture);
chapter.printInfo();
```

Agregácia
Keď objekt obsahuje zoznam iných objektov:

```
import java.util.ArrayList;
import java.util.List;

// Trieda Chapter s viacerými Lecture objektami (agregácia)
public class Chapter {
    private String title;
    private List<Lecture> lectures;  // Zoznam prednášok

    public Chapter(String title) {
        this.title = title;
        this.lectures = new ArrayList<>();  // Inicializácia prázdneho zoznamu
    }

    public void addLecture(Lecture lecture) {
        this.lectures.add(lecture);
    }

    public void printInfo() {
        System.out.println("Chapter: " + title);
        for (Lecture lecture : lectures) {
            lecture.printInfo();
        }
    }
}

// Použitie
Chapter chapter = new Chapter("Java Basics");
chapter.addLecture(new Lecture("Variables", 30));
chapter.addLecture(new Lecture("Methods", 45));
chapter.printInfo();
```

6.6. Statické premenné a metódy

```
public class Student {
    // Inštančné premenné - každý študent má svoje vlastné
    private String name;
```

```
    private int age;

    // Statická premenná - zdieľaná pre všetkých študentov
    public static int totalStudents = 0;

    // Statická konštanta - príklad konštanty
    public static final int MINIMUM_AGE = 15;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
        totalStudents++;  // Zvýši sa počet študentov pri vytvorení nového
    }

    // Normálne inštančné metódy - pracujú s inštančnými premennými
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public String getName() {
        return name;
    }

    // Statická metóda - pracuje s statickou premennou
    public static int getTotalStudents() {
        return totalStudents;
    }

    public void printInfo() {
        System.out.println("Student name: " + this.name);
        System.out.println("Student age: " + this.age);
        // Môžeme použiť static premennú aj v inštančnej metóde
        System.out.println("Total students in school: " + totalStudents);
    }
}

// Použitie statických premenných a metód
public class StaticExample {
    public static void main(String[] args) {
        // Prístup k statickej konštante bez vytvárania objektu
        System.out.println("Minimum age: " + Student.MINIMUM_AGE);

        // Vytvorenie študentov
        Student student1 = new Student("John", 20);
        Student student2 = new Student("Jane", 22);
```

```
        // Prístup k statickej premennej cez triedu
        System.out.println("Total students: " + Student.totalStudents);  // 2

        // Volanie statickej metódy
        System.out.println("Total students: " + Student.getTotalStudents());  // 2

        // Volanie inštančných metód
        student1.printInfo();
        student2.printInfo();
    }
}
```

7. Výnimky

7.1. Základy výnimiek

```
// Základné použitie try-catch bloku
try {
    int result = 10 / 0;  // Spôsobí ArithmeticException
    System.out.println("Výsledok: " + result);  // Tento kód sa nevykoná
} catch (ArithmeticException e) {
    System.out.println("Nastala chyba: " + e.getMessage());  // Vypíše: Nastala
chyba: / by zero
}
System.out.println("Program pokračuje ďalej.");  // Vykoná sa normálne

// Try-catch-finally blok
try {
    int[] numbers = {1, 2, 3};
    System.out.println(numbers[5]);  // Spôsobí ArrayIndexOutOfBoundsException
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Pokúšaš sa pristúpiť mimo rozsah poľa.");
} finally {
    System.out.println("Tento blok sa vykoná vždy, bez ohľadu na výnimky.");
}

// Viacnásobné catch bloky
try {
    // Kód, ktorý môže vyhodiť rôzne výnimky
    String str = null;
    str.length();  // NullPointerException
} catch (ArithmeticException e) {
    System.out.println("Nastala aritmetická chyba.");
} catch (NullPointerException e) {
    System.out.println("Pokúšaš sa pracovať s null referenciou.");
} catch (Exception e) {
    System.out.println("Nastala iná chyba: " + e.getMessage());
}

// Try-with-resources (od Java 7)
// Automaticky zatvorí zdroje, ktoré implementujú AutoCloseable
```

```
try (Scanner scanner = new Scanner(new File("data.txt"))) {
    while (scanner.hasNextLine()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException e) {
    System.out.println("Súbor sa nenašiel: " + e.getMessage());
}
```

7.2. Typy výnimiek

```
// Hierarchia výnimiek v Jave
// Throwable
//    ├── Error - závažné chyby, ktoré aplikácia nemôže zvládnuť
//    └── Exception
//        ├── RuntimeException - unchecked exceptions (nemusíte ich deklarovať)
//        └── Ostatné - checked exceptions (musíte ich deklarovať alebo zachytiť)

// 1. Checked Exceptions - musíte ich deklarovať alebo zachytiť
// Príklady: IOException, FileNotFoundException, SQLException
public void readFile() throws IOException {  // Deklarácia výnimky pomocou throws
    FileReader fileReader = new FileReader("example.txt");
    // Práca so súborom
    fileReader.close();
}

// 2. Unchecked Exceptions (RuntimeException) - nie je potrebné ich deklarovať
// Príklady: NullPointerException, ArrayIndexOutOfBoundsException,
ArithmeticException
public int divide(int a, int b) {
    if (b == 0) {
        throw new ArithmeticException("Delenie nulou nie je povolené");  //
Vyhodenie výnimky
    }
    return a / b;
}

// 3. Error - závažné chyby, ktoré zvyčajne nezvládate
// Príklady: OutOfMemoryError, StackOverflowError
// Tieto obvykle nemusíte zachytávať

// Príklady rôznych typov výnimiek a ich zachytenia
public void exceptionExamples() {
    try {
        // Unchecked exceptions
        int[] arr = new int[5];
        arr[10] = 50;  // ArrayIndexOutOfBoundsException

        String str = null;
        int length = str.length();  // NullPointerException

        int result = 10 / 0;  // ArithmeticException
```

```
            // Checked exceptions
            File file = new File("nonexistent.txt");
            FileInputStream fis = new FileInputStream(file);  // FileNotFoundException

        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Index mimo rozsah poľa!");
        } catch (NullPointerException e) {
            System.out.println("Referencia na null!");
        } catch (ArithmeticException e) {
            System.out.println("Aritmetická chyba: " + e.getMessage());
        } catch (FileNotFoundException e) {
            System.out.println("Súbor nebol nájdený!");
        } catch (Exception e) {
            System.out.println("Všeobecná chyba: " + e.getMessage());
        }
    }
```

7.3. Zachytávanie vstupných chýb

```
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.Random;

public class GuessNumberWithExceptions {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();
        int secretNumber = random.nextInt(10) + 1;  // Číslo od 1 do 10
        boolean correctGuess = false;

        System.out.println("Hádaj číslo od 1 do 10:");

        while (!correctGuess) {
            try {
                System.out.print("Tvoj tip: ");
                int guess = scanner.nextInt();

                if (guess < 1 || guess > 10) {
                    System.out.println("Zadaj číslo v rozsahu 1-10!");
                    continue;
                }

                if (guess == secretNumber) {
                    System.out.println("Správne! Hádané číslo bolo " +
secretNumber);
                    correctGuess = true;
                } else {
                    System.out.println("Nesprávne. Skús znova.");
                }
            } catch (InputMismatchException e) {
```

```
                System.out.println("Chyba: Musíš zadať celé číslo!");
                scanner.nextLine();  // Vyčistenie bufferu po neplatnom vstupe
            }
        }

        scanner.close();
        System.out.println("Hra skončila.");
    }
}
```

7.4. Vlastné výnimky

```
// Vlastná checked výnimka
class InsufficientFundsException extends Exception {
    private double amount;

    public InsufficientFundsException(double amount) {
        super("Nedostatok prostriedkov: chýba " + amount + " EUR");
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }
}

// Vlastná unchecked výnimka
class InvalidAgeException extends RuntimeException {
    public InvalidAgeException(String message) {
        super(message);
    }
}

// Trieda, ktorá používa vlastnú výnimku
class BankAccount {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double initialBalance) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException("Vklad musí byť kladný");
        }
        balance += amount;
    }

    // Metóda, ktorá deklaruje, že môže vyhodiť checked výnimku
```

```
    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount <= 0) {
            throw new IllegalArgumentException("Výber musí byť kladný");
        }

        if (amount > balance) {
            double shortfall = amount - balance;
            throw new InsufficientFundsException(shortfall);
        }

        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }
}

// Použitie vlastnej výnimky
public class ExceptionExample {
    public static void main(String[] args) {
        BankAccount account = new BankAccount("SK123456789", 1000);

        try {
            System.out.println("Aktuálny zostatok: " + account.getBalance() + "
EUR");
            account.withdraw(1500);  // Pokus o výber väčšej sumy, než je zostatok
        } catch (InsufficientFundsException e) {
            System.out.println("Operácia zlyhala: " + e.getMessage());
            System.out.println("Chýba ti: " + e.getAmount() + " EUR");
        } finally {
            System.out.println("Konečný zostatok: " + account.getBalance() + "
EUR");
        }

        // Príklad s unchecked výnimkou - nemusíme ju deklarovať ani zachytiť
        try {
            validateAge(-5);  // Vyhodí InvalidAgeException
        } catch (InvalidAgeException e) {
            System.out.println("Chyba validácie veku: " + e.getMessage());
        }
    }

    public static void validateAge(int age) {
        if (age < 0) {
            throw new InvalidAgeException("Vek nemôže byť záporný");
        }
        if (age > 120) {
            throw new InvalidAgeException("Vek je nereálne vysoký");
        }
        System.out.println("Vek je platný: " + age);
    }
}
```

```java
// Vlastná checked výnimka
class InsufficientFundsException extends Exception {
    private double amount;

    public InsufficientFundsException(double amount) {
        super("Nedostatok prostriedkov: chýba " + amount + " EUR");
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }
}

// Vlastná unchecked výnimka
class InvalidAgeException extends RuntimeException {
    public InvalidAgeException(String message) {
        super(message);
    }
}

// Trieda, ktorá používa vlastnú výnimku
class BankAccount {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double initialBalance) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException("Vklad musí byť kladný");
        }
        balance += amount;
    }

    // Metóda, ktorá deklaruje, že môže vyhodiť checked výnimku
    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount <= 0) {
            throw new IllegalArgumentException("Výber musí byť kladný");
        }

        if (amount > balance) {
            double shortfall = amount - balance;
            throw new InsufficientFundsException(shortfall);
        }

        balance -= amount;
    }
```

```
    public double getBalance() {
        return balance;
    }
}

// Použitie vlastnej výnimky
public class ExceptionExample {
    public static void main(String[] args) {
        BankAccount account = new BankAccount("SK123456789", 1000);

        try {
            System.out.println("Aktuálny zostatok: " + account.getBalance() + "
EUR");
            account.withdraw(1500);  // Pokus o výber väčšej sumy, než je zostatok
        } catch (InsufficientFundsException e) {
            System.out.println("Operácia zlyhala: " + e.getMessage());
            System.out.println("Chýba ti: " + e.getAmount() + " EUR");
        } finally {
            System.out.println("Konečný zostatok: " + account.getBalance() + "
EUR");
        }

        // Príklad s unchecked výnimkou - nemusíme ju deklarovať ani zachytiť
        try {
            validateAge(-5);  // Vyhodí InvalidAgeException
        } catch (InvalidAgeException e) {
            System.out.println("Chyba validácie veku: " + e.getMessage());
        }
    }

    public static void validateAge(int age) {
        if (age < 0) {
            throw new InvalidAgeException("Vek nemôže byť záporný");
        }
        if (age > 120) {
            throw new InvalidAgeException("Vek je nereálne vysoký");
        }
        System.out.println("Vek je platný: " + age);
    }
}
```

8. Dátové štruktúry

8.1. List

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Collections;

public class ListExample {
```

```java
    public static void main(String[] args) {
        // ArrayList - dynamické pole, rýchly prístup, pomalé vkladanie/mazanie v
strede
        List<String> arrayList = new ArrayList<>();

        // Pridanie prvkov
        arrayList.add("Prvý");
        arrayList.add("Druhý");
        arrayList.add("Tretí");
        arrayList.add("Štvrtý");
        System.out.println("ArrayList: " + arrayList);  // [Prvý, Druhý, Tretí,
Štvrtý]

        // Pridanie na konkrétny index
        arrayList.add(2, "Vložený");
        System.out.println("Po vložení: " + arrayList);  // [Prvý, Druhý, Vložený,
Tretí, Štvrtý]

        // Získanie prvku podľa indexu
        String element = arrayList.get(1);
        System.out.println("Prvok na indexe 1: " + element);  // Druhý

        // Nastavenie hodnoty na indexe
        arrayList.set(1, "Upravený");
        System.out.println("Po úprave: " + arrayList);  // [Prvý, Upravený,
Vložený, Tretí, Štvrtý]

        // Odstránenie prvku
        arrayList.remove(2);  // Odstráni "Vložený"
        System.out.println("Po odstránení: " + arrayList);  // [Prvý, Upravený,
Tretí, Štvrtý]

        // Odstránenie podľa hodnoty
        arrayList.remove("Upravený");
        System.out.println("Po odstránení hodnoty: " + arrayList);  // [Prvý,
Tretí, Štvrtý]

        // Veľkosť zoznamu
        System.out.println("Veľkosť: " + arrayList.size());  // 3

        // Kontrola, či zoznam obsahuje prvok
        boolean contains = arrayList.contains("Tretí");
        System.out.println("Obsahuje 'Tretí': " + contains);  // true

        // Zistenie indexu prvku
        int index = arrayList.indexOf("Tretí");
        System.out.println("Index 'Tretí': " + index);  // 1

        // Vyčistenie zoznamu
        arrayList.clear();
        System.out.println("Po vyčistení: " + arrayList);  // []

        // LinkedList - rýchle vkladanie/odstránenie, pomalší prístup k prvkom
        LinkedList<String> linkedList = new LinkedList<>();
```

```
        linkedList.add("Jablko");
        linkedList.add("Banán");
        linkedList.add("Pomaranč");

        // Špeciálne metódy LinkedList
        linkedList.addFirst("Jahoda");  // Pridá na začiatok
        linkedList.addLast("Hruška");   // Pridá na koniec

        System.out.println("\nLinkedList: " + linkedList);
        System.out.println("Prvý prvok: " + linkedList.getFirst());
        System.out.println("Posledný prvok: " + linkedList.getLast());

        // Odstránenie z konca a začiatku
        String first = linkedList.removeFirst();
        String last = linkedList.removeLast();
        System.out.println("Odstránené: " + first + ", " + last);
        System.out.println("LinkedList po odstránení: " + linkedList);

        // Vytvorenie zoznamu s inicializáciou (od Java 9)
        List<Integer> numbers = List.of(5, 2, 8, 1, 9, 3);
        // numbers.add(10);  // UnsupportedOperationException - je nemenný

        // Konverzia na ArrayList pre úpravy
        List<Integer> mutableNumbers = new ArrayList<>(numbers);

        // Triedenie zoznamu
        Collections.sort(mutableNumbers);
        System.out.println("\nUtriedené čísla: " + mutableNumbers);

        // Obrátenie poradia
        Collections.reverse(mutableNumbers);
        System.out.println("Obrátené poradie: " + mutableNumbers);

        // Prehadzovanie prvkov
        Collections.shuffle(mutableNumbers);
        System.out.println("Náhodné poradie: " + mutableNumbers);

        // Nájdenie min/max
        int min = Collections.min(mutableNumbers);
        int max = Collections.max(mutableNumbers);
        System.out.println("Min: " + min + ", Max: " + max);
    }
}
```

8.2. Set

```
import java.util.HashSet;
import java.util.TreeSet;
import java.util.LinkedHashSet;
import java.util.Set;
```

```
public class SetExample {
    public static void main(String[] args) {
        // HashSet - rýchly, negarantuje poradie
        Set<String> hashSet = new HashSet<>();

        // Pridanie prvkov
        hashSet.add("Jablko");
        hashSet.add("Banán");
        hashSet.add("Pomaranč");
        hashSet.add("Jablko");  // Duplicita - nebude pridané

        System.out.println("HashSet: " + hashSet);  // Náhodné poradie, bez
duplicít

        // Kontrola, či množina obsahuje prvok
        boolean contains = hashSet.contains("Banán");
        System.out.println("Obsahuje 'Banán': " + contains);  // true

        // Odstránenie prvku
        hashSet.remove("Banán");
        System.out.println("Po odstránení: " + hashSet);

        // Veľkosť množiny
        System.out.println("Veľkosť: " + hashSet.size());

        // TreeSet - zotriedená množina
        TreeSet<String> treeSet = new TreeSet<>();
        treeSet.add("Jablko");
        treeSet.add("Banán");
        treeSet.add("Pomaranč");
        treeSet.add("Hruška");

        System.out.println("\nTreeSet (zotriedený): " + treeSet);  // Abecedne
zotriedené

        // Špeciálne metódy TreeSet
        System.out.println("Prvý prvok: " + treeSet.first());
        System.out.println("Posledný prvok: " + treeSet.last());

        // Podmnožiny
        System.out.println("Prvky menšie ako 'Jablko': " +
treeSet.headSet("Jablko"));
        System.out.println("Prvky väčšie alebo rovné 'Jablko': " +
treeSet.tailSet("Jablko"));

        // LinkedHashSet - zachováva poradie vloženia
        LinkedHashSet<String> linkedHashSet = new LinkedHashSet<>();
        linkedHashSet.add("Pomaranč");
        linkedHashSet.add("Jablko");
        linkedHashSet.add("Banán");

        System.out.println("\nLinkedHashSet (zachovanie poradia): " +
linkedHashSet);
```

```
        // Operácie s množinami
        Set<Integer> set1 = new HashSet<>(Set.of(1, 2, 3, 4, 5));
        Set<Integer> set2 = new HashSet<>(Set.of(4, 5, 6, 7, 8));

        // Prienik množín
        Set<Integer> intersection = new HashSet<>(set1);
        intersection.retainAll(set2);
        System.out.println("\nPrienik: " + intersection);  // [4, 5]

        // Zjednotenie množín
        Set<Integer> union = new HashSet<>(set1);
        union.addAll(set2);
        System.out.println("Zjednotenie: " + union);  // [1, 2, 3, 4, 5, 6, 7, 8]

        // Rozdiel množín (set1 - set2)
        Set<Integer> difference = new HashSet<>(set1);
        difference.removeAll(set2);
        System.out.println("Rozdiel (set1 - set2): " + difference);  // [1, 2, 3]
    }
}
```

8.3. Map

```
import java.util.HashMap;
import java.util.TreeMap;
import java.util.LinkedHashMap;
import java.util.Map;

public class MapExample {
    public static void main(String[] args) {
        // HashMap - rýchly, negarantuje poradie
        Map<String, Integer> hashMap = new HashMap<>();

        // Vloženie párov kľúč-hodnota
        hashMap.put("Jablko", 10);
        hashMap.put("Banán", 5);
        hashMap.put("Pomaranč", 8);
        hashMap.put("Jablko", 12);  // Prepíše predošlú hodnotu pre "Jablko"

        System.out.println("HashMap: " + hashMap);  // Náhodné poradie

        // Získanie hodnoty podľa kľúča
        int appleCount = hashMap.get("Jablko");
        System.out.println("Počet jabĺk: " + appleCount);  // 12

        // Hodnota s defaultom, ak kľúč neexistuje
        int grapeCount = hashMap.getOrDefault("Hrozno", 0);
        System.out.println("Počet hrozna: " + grapeCount);  // 0

        // Kontrola, či mapa obsahuje kľúč alebo hodnotu
        boolean hasKey = hashMap.containsKey("Banán");
```

```
        boolean hasValue = hashMap.containsValue(5);
        System.out.println("Obsahuje kľúč 'Banán': " + hasKey);  // true
        System.out.println("Obsahuje hodnotu 5: " + hasValue);   // true

        // Odstránenie páru podľa kľúča
        hashMap.remove("Banán");
        System.out.println("Po odstránení: " + hashMap);

        // Veľkosť mapy
        System.out.println("Veľkosť: " + hashMap.size());

        // Prechádzanie cez mapu - kľúče
        System.out.println("\nKľúče:");
        for (String key : hashMap.keySet()) {
            System.out.println(key);
        }

        // Prechádzanie cez mapu - hodnoty
        System.out.println("\nHodnoty:");
        for (int value : hashMap.values()) {
            System.out.println(value);
        }

        // Prechádzanie cez mapu - páry kľúč-hodnota
        System.out.println("\nPáry kľúč-hodnota:");
        for (Map.Entry<String, Integer> entry : hashMap.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }

        // TreeMap - zotriedená mapa podľa kľúčov
        TreeMap<String, Integer> treeMap = new TreeMap<>();
        treeMap.put("Jablko", 10);
        treeMap.put("Banán", 5);
        treeMap.put("Pomaranč", 8);

        System.out.println("\nTreeMap (zotriedená podľa kľúčov): " + treeMap);

        // Špeciálne metódy TreeMap
        System.out.println("Prvý kľúč: " + treeMap.firstKey());
        System.out.println("Posledný kľúč: " + treeMap.lastKey());
        System.out.println("Nižšie záznamy než 'Jablko': " +
treeMap.headMap("Jablko"));

        // LinkedHashMap - zachováva poradie vloženia
        LinkedHashMap<String, Integer> linkedHashMap = new LinkedHashMap<>();
        linkedHashMap.put("Pomaranč", 8);
        linkedHashMap.put("Jablko", 10);
        linkedHashMap.put("Banán", 5);

        System.out.println("\nLinkedHashMap (zachovanie poradia): " +
linkedHashMap);

        // Vytvorenie mapy s inicializáciou (od Java 9)
        Map<String, String> capitals = Map.of(
```

```
            "Slovensko", "Bratislava",
            "Česko", "Praha",
            "Rakúsko", "Viedeň"
        );

        System.out.println("\nHlavné mestá: " + capitals);
    }
}
```

9. Lambda výrazy a Streamy

9.1. Lambda výrazy

```
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.function.Supplier;

public class LambdaExample {
    public static void main(String[] args) {
        // 1. Lambda výraz pre jednoduché rozhranie
        // Tradičný spôsob pomocou anonymnej triedy
        Runnable traditionalRunnable = new Runnable() {
            @Override
            public void run() {
                System.out.println("Tradičný spôsob");
            }
        };
        traditionalRunnable.run();

        // Lambda ekvivalent
        Runnable lambdaRunnable = () -> System.out.println("Lambda spôsob");
        lambdaRunnable.run();

        // 2. Lambda s parametrami
        // Consumer - prijíma parameter, nič nevracia
        Consumer<String> printConsumer = message -> System.out.println("Správa: "
+ message);
        printConsumer.accept("Hello World");  // Vypíše: Správa: Hello World

        // Lambda s viacerými parametrami
        Calculator add = (a, b) -> a + b;
        Calculator subtract = (a, b) -> a - b;
        System.out.println("5 + 3 = " + add.calculate(5, 3));       // 8
        System.out.println("5 - 3 = " + subtract.calculate(5, 3));  // 2

        // 3. Lambda s blokom kódu
        Calculator multiply = (a, b) -> {
            System.out.println("Násobenie " + a + " * " + b);
            return a * b;
        };
```

```
        System.out.println("5 * 3 = " + multiply.calculate(5, 3));  // 15

        // 4. Predikát - vracia boolean
        Predicate<Integer> isEven = number -> number % 2 == 0;
        System.out.println("Je 4 párne? " + isEven.test(4));  // true
        System.out.println("Je 7 párne? " + isEven.test(7));  // false

        // Kombinácie predikátov
        Predicate<Integer> isPositive = number -> number > 0;
        Predicate<Integer> isPositiveAndEven = isPositive.and(isEven);
        System.out.println("Je 6 kladné a párne? " + isPositiveAndEven.test(6));
// true

        // 5. Function - transformácia vstupnej hodnoty na výstupnú
        Function<String, Integer> getLength = text -> text.length();
        System.out.println("Dĺžka 'Hello': " + getLength.apply("Hello"));  // 5

        // Reťazenie funkcií
        Function<Integer, Integer> doubled = n -> n * 2;
        Function<Integer, Integer> squared = n -> n * n;

        // Najprv umocnenie, potom zdvojnásobenie
        Function<Integer, Integer> squaredThenDoubled = doubled.compose(squared);
        System.out.println("5 na druhú a potom * 2: " +
squaredThenDoubled.apply(5));  // 50

        // Najprv zdvojnásobenie, potom umocnenie
        Function<Integer, Integer> doubledThenSquared = doubled.andThen(squared);
        System.out.println("5 * 2 a potom na druhú: " +
doubledThenSquared.apply(5));  // 100

        // 6. Supplier - negeneruje vstup, len vracia výstup
        Supplier<Double> randomNumber = () -> Math.random();
        System.out.println("Náhodné číslo: " + randomNumber.get());

        // 7. Vlastné funkčné rozhranie
        HelloFunction greeting = name -> "Hello, " + name + "!";
        System.out.println(greeting.sayHello("John"));  // Hello, John!
    }

    // Vlastné funkčné rozhranie (musí mať presne jednu abstraktnú metódu)
    @FunctionalInterface
    interface Calculator {
        int calculate(int a, int b);
    }

    @FunctionalInterface
    interface HelloFunction {
        String sayHello(String name);
    }
}
```

9.2. Streamy

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class StreamExample {
    public static void main(String[] args) {
        // 1. Vytvorenie streamov

        // Stream z kolekcie
        List<String> fruits = Arrays.asList("jablko", "banán", "pomaranč",
"hruška", "jahoda");
        Stream<String> fruitStream = fruits.stream();

        // Stream z poľa
        String[] namesArray = {"John", "Mary", "Bob", "Alice"};
        Stream<String> namesStream = Arrays.stream(namesArray);

        // Stream z hodnôt
        Stream<Integer> numbersStream = Stream.of(1, 2, 3, 4, 5);

        // Generovanie streamov
        Stream<Integer> infiniteStream = Stream.iterate(0, n -> n + 2);  // 0, 2,
4, 6, ...
        Stream<Integer> limitedStream = infiniteStream.limit(5);  // 0, 2, 4, 6, 8

        // 2. Základné operácie so streamami

        // Filter - vyberie prvky spĺňajúce podmienku
        List<String> aFruits = fruits.stream()
                .filter(fruit -> fruit.startsWith("j"))
                .collect(Collectors.toList());
        System.out.println("Ovocie začínajúce na 'j': " + aFruits);  // [jablko,
jahoda]

        // Map - transformuje prvky
        List<Integer> fruitLengths = fruits.stream()
                .map(String::length)
                .collect(Collectors.toList());
        System.out.println("Dĺžky názvov ovocia: " + fruitLengths);  // [6, 5, 8,
6, 6]

        // FlatMap - zlúči viacero streamov do jedného
        List<List<Integer>> nestedList = Arrays.asList(
                Arrays.asList(1, 2, 3),
                Arrays.asList(4, 5, 6),
                Arrays.asList(7, 8, 9)
```

```
        );

        List<Integer> flatList = nestedList.stream()
                .flatMap(List::stream)
                .collect(Collectors.toList());
        System.out.println("Zlúčený zoznam: " + flatList);  // [1, 2, 3, 4, 5, 6,
7, 8, 9]

        // Distinct - odstráni duplicity
        List<Integer> numbers = Arrays.asList(1, 2, 2, 3, 3, 3, 4, 5, 5);
        List<Integer> distinctNumbers = numbers.stream()
                .distinct()
                .collect(Collectors.toList());
        System.out.println("Unikátne čísla: " + distinctNumbers);  // [1, 2, 3, 4,
5]

        // Sorted - utriedi## 7. Práca s výnimkami (Exceptions)

        // Sorted - utriedi prvky
        List<String> sortedFruits = fruits.stream()
                .sorted()
                .collect(Collectors.toList());
        System.out.println("Utriedené ovocie: " + sortedFruits);  // [banán,
hruška, jablko, jahoda, pomaranč]

        // Limit a Skip
        List<String> limitedFruits = fruits.stream()
                .limit(3)
                .collect(Collectors.toList());
        System.out.println("Prvé 3 ovocia: " + limitedFruits);  // [jablko, banán,
pomaranč]

        List<String> skippedFruits = fruits.stream()
                .skip(2)
                .collect(Collectors.toList());
        System.out.println("Bez prvých 2 ovocia: " + skippedFruits);  //
[pomaranč, hruška, jahoda]

        // 3. Terminálne operácie

        // ForEach - vykoná akciu pre každý prvok
        System.out.print("Vypísanie ovocia: ");
        fruits.stream().forEach(fruit -> System.out.print(fruit + " "));
        System.out.println();

        // Count - počet prvkov
        long count = fruits.stream().count();
        System.out.println("Počet ovocia: " + count);  // 5

        // AnyMatch, AllMatch, NoneMatch
        boolean anyStartsWithB = fruits.stream().anyMatch(fruit ->
fruit.startsWith("b"));
        boolean allLongerThan3 = fruits.stream().allMatch(fruit -> fruit.length()
> 3);
```

```
        boolean noneStartsWithZ = fruits.stream().noneMatch(fruit ->
fruit.startsWith("z"));

        System.out.println("Aspoň jedno ovocie začína na 'b': " + anyStartsWithB);
// true
        System.out.println("Všetky ovocia majú viac ako 3 znaky: " +
allLongerThan3);  // true
        System.out.println("Žiadne ovocie nezačína na 'z': " + noneStartsWithZ);
// true

        // FindFirst, FindAny
        Optional<String> firstFruit = fruits.stream().findFirst();
        Optional<String> anyFruit = fruits.stream().findAny();

        System.out.println("Prvé ovocie: " + firstFruit.orElse("Žiadne"));  //
jablko
        System.out.println("Ľubovoľné ovocie: " + anyFruit.orElse("Žiadne"));  //
závisí od implementácie

        // Reduce - kombinuje prvky do jedného výsledku
        Optional<String> combinedFruits = fruits.stream()
                .reduce((a, b) -> a + ", " + b);
        System.out.println("Spojené ovocia: " + combinedFruits.orElse(""));

        // S počiatočnou hodnotou
        String combinedWithPrefix = fruits.stream()
                .reduce("Ovocia: ", (a, b) -> a + ", " + b);
        System.out.println(combinedWithPrefix);

        // Súčet čísel
        int sum = IntStream.range(1, 11).reduce(0, Integer::sum);
        System.out.println("Súčet čísel 1-10: " + sum);  // 55

        // 4. Collectors - spôsoby zbierania výsledkov

        // toList(), toSet()
        List<String> longFruits = fruits.stream()
                .filter(fruit -> fruit.length() > 5)
                .collect(Collectors.toList());
        System.out.println("Ovocia s názvom dlhším ako 5 znakov: " + longFruits);

        // joining()
        String joinedFruits = fruits.stream()
                .collect(Collectors.joining(", ", "Ovocia: ", "."));
        System.out.println(joinedFruits);  // Ovocia: jablko, banán, pomaranč,
hruška, jahoda.

        // averaging, summing, counting
        double avgLength = fruits.stream()
                .collect(Collectors.averagingInt(String::length));
        System.out.println("Priemerná dĺžka názvu ovocia: " + avgLength);

        // groupingBy - zoskupenie podľa kľúča
        Map<Integer, List<String>> fruitsByLength = fruits.stream()
```

```
                .collect(Collectors.groupingBy(String::length));
        System.out.println("Ovocia podľa dĺžky názvu: " + fruitsByLength);

        // partitioningBy - rozdelenie na dve skupiny podľa predikátu
        Map<Boolean, List<String>> partitioned = fruits.stream()
                .collect(Collectors.partitioningBy(fruit -> fruit.length() > 6));
        System.out.println("Ovocia rozdelené podľa dĺžky názvu > 6: " +
partitioned);
    }
}
```

9.3. Method references

```
import java.util.Arrays;
import java.util.List;
import java.util.function.BiFunction;
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Supplier;

public class MethodReferenceExample {
    public static void main(String[] args) {
        // 1. Referencia na statickú metódu

        // Lambda
        Function<String, Integer> parseIntLambda = s -> Integer.parseInt(s);

        // Method reference
        Function<String, Integer> parseIntRef = Integer::parseInt;

        System.out.println(parseIntLambda.apply("5"));  // 5
        System.out.println(parseIntRef.apply("5"));     // 5

        // 2. Referencia na inštančnú metódu konkrétneho objektu

        // Lambda
        Consumer<String> printLambda = s -> System.out.println(s);

        // Method reference
        Consumer<String> printRef = System.out::println;

        printLambda.accept("Hello Lambda");
        printRef.accept("Hello Method Reference");

        String prefix = "Mr. ";

        // Lambda
        Function<String, String> prefixLambda = s -> prefix.concat(s);

        // Method reference
        Function<String, String> prefixRef = prefix::concat;
```

```
        System.out.println(prefixLambda.apply("Smith"));  // Mr. Smith
        System.out.println(prefixRef.apply("Jones"));     // Mr. Jones

        // 3. Referencia na inštančnú metódu typu

        // Lambda
        Function<String, Integer> lengthLambda = s -> s.length();

        // Method reference
        Function<String, Integer> lengthRef = String::length;

        System.out.println(lengthLambda.apply("Hello"));  // 5
        System.out.println(lengthRef.apply("Hello"));     // 5

        // BiFunction príklad
        BiFunction<String, Integer, String> substringLambda = (s, i) ->
s.substring(i);
        BiFunction<String, Integer, String> substringRef = String::substring;

        System.out.println(substringLambda.apply("Hello", 2));  // llo
        System.out.println(substringRef.apply("Hello", 2));     // llo

        // 4. Referencia na konštruktor

        // Lambda
        Supplier<List<String>> listSupplierLambda = () -> new
java.util.ArrayList<>();

        // Method reference
        Supplier<List<String>> listSupplierRef = java.util.ArrayList::new;

        List<String> list1 = listSupplierLambda.get();
        List<String> list2 = listSupplierRef.get();

        // S parametrami
        Function<Integer, List<String>> sizedListLambda = size -> new
java.util.ArrayList<>(size);
        Function<Integer, List<String>> sizedListRef = java.util.ArrayList::new;

        List<String> sizedList1 = sizedListLambda.apply(10);
        List<String> sizedList2 = sizedListRef.apply(10);

        // 5. Praktické použitie
        List<String> names = Arrays.asList("John", "Jane", "Bob", "Alice");

        // Triedenie pomocou method reference
        names.sort(String::compareToIgnoreCase);
        System.out.println("Utriedené mená: " + names);

        // forEach s method reference
        System.out.print("Všetky mená: ");
        names.forEach(System.out::println);
```

```
        // map s method reference
        List<Integer> nameLengths = names.stream()
                .map(String::length)
                .toList();
        System.out.println("Dĺžky mien: " + nameLengths);

        // filter s method reference (pomocou predikátu)
        List<String> nonEmptyNames = names.stream()
                .filter(s -> !s.isEmpty())  // Nie je vhodný pre method reference
                .toList();
        System.out.println("Neprázdne mená: " + nonEmptyNames);
    }
}
```

10. Slovník programátorských výrazov

10.1. Základné príkazy

Anglický výraz	Slovenský význam	Popis
`variable`	premenná	Úložný priestor pre hodnoty v programe
`data type`	dátový typ	Typ údajov, ktoré môže premenná obsahovať (int, String, boolean, atď.)
`array`	pole	Štruktúra na uchovávanie viacerých hodnôt rovnakého typu
`string`	reťazec	Postupnosť znakov
`integer`	celé číslo	Číselná hodnota bez desatinnej časti
`float/double`	desatinné číslo	Číselná hodnota s desatinnou časťou
`boolean`	logická hodnota	Hodnota true (pravda) alebo false (nepravda)
`operator`	operátor	Symbol vykonávajúci operáciu (+, -, *, /, %, ==, !=, atď.)
`if-else`	ak-inak	Podmienená vetva vykonávania programu
`loop`	cyklus	Opakované vykonávanie bloku kódu
`for`	pre	Cyklus s definovaným počtom iterácií
`while`	pokiaľ	Cyklus, ktorý beží, pokiaľ je splnená podmienka
`do-while`	urob-pokiaľ	Cyklus, ktorý sa vykoná aspoň raz a pokračuje, pokiaľ je splnená podmienka
`break`	prerušiť	Príkaz na ukončenie cyklu
`continue`	pokračovať	Príkaz na preskočenie zvyšku iterácie a prechod na ďalšiu
`method/function`	metóda/funkcia	Blok kódu, ktorý vykonáva určitú akciu
`return`	vrátiť	Ukončenie metódy s návratom hodnoty

Anglický výraz	Slovenský význam	Popis
`parameter`	parameter	Vstupná hodnota pre metódu
`argument`	argument	Konkrétna hodnota odovzdaná metóde pri jej volaní
`compile`	kompilovať	Preklad zdrojového kódu do spustiteľnej formy
`debug`	ladiť	Hľadanie a oprava chýb v programe

10.2. OOP slovník

Anglický výraz	Slovenský význam	Popis
`class`	trieda	Šablóna pre vytváranie objektov
`object`	objekt	Inštancia triedy
`constructor`	konštruktor	Metóda volaná pri vytváraní objektu
`inheritance`	dedičnosť	Preberanie vlastností a metód od rodičovskej triedy
`encapsulation`	zapuzdrenie	Skrývanie interných detailov implementácie
`polymorphism`	polymorfizmus	Schopnosť objektu vystupovať v rôznych formách
`abstraction`	abstrakcia	Zjednodušenie zložitosti pomocou modelovania
`interface`	rozhranie	Súbor metód, ktoré trieda implementuje
`abstract class`	abstraktná trieda	Trieda, ktorú nemožno inštanciovať
`private`	súkromný	Prístupový modifikátor - prístup len v rámci triedy
`protected`	chránený	Prístupový modifikátor - prístup v rámci balíčka a podtried
`public`	verejný	Prístupový modifikátor - prístup odkiaľkoľvek
`getter`	získavač	Metóda na získanie hodnoty atribútu
`setter`	nastavovač	Metóda na nastavenie hodnoty atribútu
`static`	statický	Premenná alebo metóda patriaca triede, nie objektu
`final`	konečný	Hodnota, ktorá nemôže byť zmenená
`override`	prepísať	Nová implementácia metódy z rodičovskej triedy
`overload`	preťažiť	Viacero metód s rovnakým názvom, ale rôznymi parametrami

10.3. HTTP metódy a webový vývoj

Anglický výraz	Slovenský význam	Popis
`GET`	získať	HTTP metóda na získanie dát zo servera
`POST`	odoslať	HTTP metóda na odoslanie dát na server

Anglický výraz	Slovenský význam	Popis
`PUT`	umiestniť	HTTP metóda na aktualizáciu existujúcich dát
`DELETE`	odstrániť	HTTP metóda na odstránenie dát
`PATCH`	upraviť	HTTP metóda na čiastočnú úpravu dát
`API`	rozhranie pre programovanie aplikácií	Súbor pravidiel a protokolov na komunikáciu medzi softvérovými komponentmi
`REST`	Representational State Transfer	Architektonický štýl pre distribuované hypermediálne systémy
`JSON`	JavaScript Object Notation	Formát výmeny dát založený na JavaScript syntaxi
`XML`	Extensible Markup Language	Značkovací jazyk pre dokumenty s hierarchickou štruktúrou
`client`	klient	Zariadenie alebo program, ktorý pristupuje k službám poskytovaným serverom
`server`	server	Zariadenie alebo program, ktorý poskytuje služby iným zariadeniam
`request`	požiadavka	Správa odoslaná klientom serveru
`response`	odpoveď	Správa odoslaná serverom klientovi
`header`	hlavička	Metadáta o HTTP správe
`cookie`	keksík	Malý kúsok dát uložený v prehliadači
`session`	relácia	Stav interakcie medzi klientom a serverom
`authentication`	autentifikácia	Proces overenia identity používateľa
`authorization`	autorizácia	Proces určenia prístupových práv používateľa
`token`	token	Jedinečný identifikátor pre autentifikáciu
`endpoint`	koncový bod	URL adresa, kde API poskytuje určitú službu

10.4. Databázové pojmy

Anglický výraz	Slovenský význam	Popis
`database`	databáza	Organizovaná kolekcia dát
`table`	tabuľka	Súbor údajov usporiadaný v riadkoch a stĺpcoch
`row/record`	riadok/záznam	Jedna položka v tabuľke
`column/field`	stĺpec/pole	Atribút/vlastnosť v tabuľke

Anglický výraz	Slovenský význam	Popis
`primary key`	primárny kľúč	Jednoznačný identifikátor záznamu v tabuľke
`foreign key`	cudzí kľúč	Pole v tabuľke, ktoré odkazuje na primárny kľúč v inej tabuľke
`index`	index	Dátová štruktúra pre zrýchlenie vyhľadávania
`query`	dotaz	Požiadavka na získanie alebo manipuláciu s dátami
`SQL`	SQL	Structured Query Language - jazyk pre prácu s databázami
`CRUD`	CRUD	Create, Read, Update, Delete - základné operácie s dátami
`CREATE`	vytvoriť	SQL príkaz na vytvorenie nových tabuliek, databáz, atď.
`INSERT`	vložiť	SQL príkaz na pridanie nových záznamov do tabuľky
`SELECT`	vybrať	SQL príkaz na čítanie/získanie dát z tabuľky
`UPDATE`	aktualizovať	SQL príkaz na zmenu existujúcich dát v tabuľke
`DELETE`	vymazať	SQL príkaz na odstránenie záznamov z tabuľky
`JOIN`	spojiť	SQL operácia na kombinovanie riadkov z dvoch alebo viacerých tabuliek
`transaction`	transakcia	Sekvencia operácií vykonaných ako jediná logická jednotka práce
`commit`	potvrdiť	Ukončenie transakcie a uloženie zmien
`rollback`	vrátiť späť	Ukončenie transakcie a zahodenie zmien
`database driver`	databázový ovládač	Software, ktorý umožňuje aplikácii komunikovať s databázou
`connection pool`	pool pripojení	Metóda správy databázových pripojení
`schema`	schéma	Logické zoskupenie objektov databázy
`view`	pohľad	Virtuálna tabuľka založená na výsledku SQL príkazu
`stored procedure`	uložená procedúra	Pripravený SQL kód, ktorý môžete uložiť a opakovane spúšťať
`trigger`	spúšťač	SQL kód, ktorý sa automaticky spustí ako reakcia na určitú udalosť
`ORM`	ORM	Object-Relational Mapping - technika konverzie dát medzi objektmi a relačnými databázami