



# Podprogramy a Rekurze



# Podprogramy



# Podprogramy

Jsou základní stavební bloky kódu v Java  
nebo C#. Slouží k zapouzdření logiky,  
znovupoužitelnosti kódu a modularitě

# Vlastnosti metod

## 1. Deklarace

- Každá metoda je definována uvnitř třídy

## 2. Syntaxe

```
modifikátory návratový_typ název_metody(parametry) {  
    // Tělo metody  
}
```

Modifikátory = public, private, protected, static, final



# Typy podprogramů

**Statické** – za použití keyword static, volají se přes třídu

**Instanční** – vyžadují vytvoření objektu třídy

**Abstraktní** – nelze definovat ve standardních třídách, keyword abstract, mají pouze hlavičku, musí být přepsány (@Override) ve třídách, které dědí abstraktní třídu nebo implementují rozhraní

**Overloaded methods** (*metody s přetížením*) – více metod se stejným názvem, ale odlišnými parametry

# Příklady

## Statické

Java

```
public class MathUtils {  
    public static int add(int a, int b) {  
        return a + b;  
    }  
}
```

C#

```
public class MathUtils {  
    public static int Add(int a, int b) {  
        return a + b;  
    }  
}
```

## Overloaded

Java

```
public class Calculator {  
    public int add(int a, int b) { return a + b; }  
    public double add(double a, double b) { return a + b; }  
}
```

C#

```
public class Calculator {  
    public int Add(int a, int b) { return a + b; }  
    public double Add(double a, double b) { return a + b; }  
}
```

## Instanční

```
public class Greeting {  
    public void greet(String name) {  
        System.out.println("Hello, " + name);  
    }  
}
```

```
public class Greeting {  
    public void Greet(string name) {  
        Console.WriteLine("Hello, " + name);  
    }  
}
```

## Abstraktní

```
abstract class Animal {  
    public abstract void sound();  
}
```

```
public abstract class Animal {  
    public abstract void Sound();  
}
```



Rekurze



# Rekurze

Je to technika, kdy metoda volá sama sebe.  
Používá se pro řešení problémů, které lze  
rozdělit na menší podproblémy.



# Vlastnosti rekurze

## 1. Ukončovací podmínka

- Ukončovací podmínka, zabraňující nekonečné rekurzi

## 2. Rekurzivní volání

- Volání metody s jednodušším problémem



# Příklad rekurze

```
public class Factorial {  
    public static int factorial(int n) {  
        if (n == 0) {  
            return 1; // Základní případ  
        }  
        return n * factorial(n - 1); // Rekursivní volání  
    }  
  
    public static void main(String[] args) {  
        System.out.println(factorial(5)); // Výstup: 120  
    }  
}
```

1. Zavoláme factorial(5).
2. To zavolá factorial(4), protože  $\text{factorial}(5) = 5 * \text{factorial}(4)$ .
3. factorial(4) zavolá factorial(3), protože  $\text{factorial}(4) = 4 * \text{factorial}(3)$ .
4. factorial(3) zavolá factorial(2), protože  $\text{factorial}(3) = 3 * \text{factorial}(2)$ .
5. factorial(2) zavolá factorial(1), protože  $\text{factorial}(2) = 2 * \text{factorial}(1)$ .
6. factorial(1) zavolá factorial(0), protože  $\text{factorial}(1) = 1 * \text{factorial}(0)$ .
7. factorial(0) vrátí 1 (základní případ).

1. factorial(1) vrátí  $1 * 1 = 1$ .
2. factorial(2) vrátí  $2 * 1 = 2$ .
3. factorial(3) vrátí  $3 * 2 = 6$ .
4. factorial(4) vrátí  $4 * 6 = 24$ .
5. factorial(5) vrátí  $5 * 24 = 120$ .

```
public class Fibonacci {  
    public static int fibonacci(int n) {  
        if (n <= 1) {  
            return n; // Základní případ  
        }  
        return fibonacci(n - 1) + fibonacci(n - 2); // Rekursivní volání  
    }  
  
    public static void main(String[] args) {  
        System.out.println(fibonacci(6)); // Výstup: 8  
    }  
}
```

1. Zavoláme fibonacci(6).
2. To zavolá fibonacci(5) a fibonacci(4), protože  $\text{fibonacci}(6) = \text{fibonacci}(5) + \text{fibonacci}(4)$ .
3. fibonacci(5) zavolá fibonacci(4) a fibonacci(3), protože  $\text{fibonacci}(5) = \text{fibonacci}(4) + \text{fibonacci}(3)$ .
4. fibonacci(4) zavolá fibonacci(3) a fibonacci(2), protože  $\text{fibonacci}(4) = \text{fibonacci}(3) + \text{fibonacci}(2)$ .
5. fibonacci(3) zavolá fibonacci(2) a fibonacci(1), protože  $\text{fibonacci}(3) = \text{fibonacci}(2) + \text{fibonacci}(1)$ .
6. fibonacci(2) zavolá fibonacci(1) a fibonacci(0), protože  $\text{fibonacci}(2) = \text{fibonacci}(1) + \text{fibonacci}(0)$ .
7. fibonacci(1) vrátí 1.
8. fibonacci(0) vrátí 0.

1. fibonacci(1) vrátí 1.
2. fibonacci(0) vrátí 0.
3. fibonacci(2) vrátí  $1 + 0 = 1$ .
4. fibonacci(3) vrátí  $1 + 1 = 2$ .
5. fibonacci(4) vrátí  $2 + 1 = 3$ .
6. fibonacci(5) vrátí  $3 + 2 = 5$ .
7. fibonacci(6) vrátí  $5 + 3 = 8$ .

# Výhody a nevýhody

## VÝHODY

1. Jednoduchost a čitelnost
2. Snížení složitosti kódy
3. Lepší práce s hierarchickými datovými strukturami

## NEVÝHODY

1. Vyšší paměťové nároky
2. Může způsobit *stack overflow*
3. Omezený rozsah
4. Při špatném použití nemusí být výhodná