



Funded by the
Erasmus+ Programme
of the European Union



This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



KI201 - JAVA 4: STRUKTURE PODATAKA I ALGORITMI – DEO A

Rekurzija

Lekcija 01

PRIRUČNIK ZA STUDENTE

KI201 - JAVA 4: STRUKTURE PODATAKA I ALGORITMI – DEO A

Lekcija 01

REKURZIJA

- ✓ Rekurzija
- ✓ Poglavlje 1: Šta je rekurzija?
- ✓ Poglavlje 2: Studija slučaja: Proračun faktoriala
- ✓ Poglavlje 3: Studija slučaja: Proračun Fibonačijevih brojeva
- ✓ Poglavlje 4: Rešavanje problema upotrebom rekurzije
- ✓ Poglavlje 5: Rekurzivni pomoćni metodi
- ✓ Poglavlje 6: Studija slučaja: Hanojske kule
- ✓ Poglavlje 7: Rekurzija i iteracija
- ✓ Poglavlje 8: Repna rekurzija
- ✓ Poglavlje 9: Primeri upotrebe rekurzije
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Rekurzija je jedna tehnika koja na elegantan način rešava određene probleme programiranja koji se teško rešavaju upotrebom jednostavnih petlji.

U ovoj lekcije, naučićete da:

- opišete metod rekurzije i korist od njegove upotrebe,
- razvijete metode rekurzije a rekurzivne matematičke funkcije
- objasnite kako metod rekurzije se poziva i kakao se koristi
- rešite probleme upotrebe rekurzije
- primenite probleme sortiranja primenom rekurzije,
- primenite binarnu pretragu primenom rekurzije
- dobijete veličinu direktorijuma upotrebom rekurzije,
- rešite problem Hanojskih kula upotrebom rekurzije,
- objasnite vezu i razliku između rekurzije i iteracije.

Referenca: Y. Daniel Liang, INTRODUCTION TO JAVA PROGRAMMING (COMPREHENSIVE VERSION), Tenth Edition, Pearson, ISBN 10: 0-13-376131-2, ISBN 13: 978-0-13-376131-3

Ovo je osnovni udžbenik za ovaj predmet i preporučuje se studentima da ga koriste,

✓ Poglavlje 1

Šta je rekurzija?

POJAM REKURZIJE

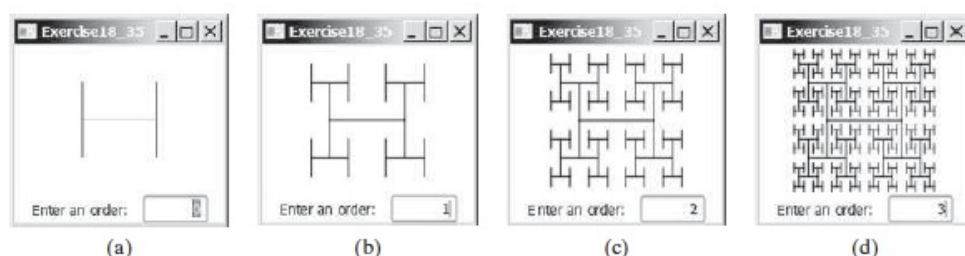
Pod rekurzivnim metodom se podrazumeva metod koji poziva sam sebe

Pod rekurzivnim metodom se podrazumeva metod koji poziva sam sebe.

Programi koji koriste rekurziju primenjuju određene rekurzivne metode. U pojedinim slučajevima, rekurzija omogućava znatno jednostavnije rešavanje pojedinih problema.

Navešćemo jedan primer. Pri projektovanju VLSI čipova, koriste se tzv. H stabla (slika 1.a) kao mreža za distribuciju vremena pri rutiranju vremenskih signala u sve delove čipa, ali tako da obezbeđuje jednak zaostatak signala svuda. Kako bi napisali program koji bi nacrtao H stabla? Rekurzija obezbeđuje jedno elegantno rešenje (slika 1). Kada se napravi program za crtanje jednog H stabla, onda se isti program poziva da nacrtá H stablo na kraju svakog dela prethodno nacrtanog H stabla (slika 1.b).

Ponovo pozivanja istog dela programa da nacrtá nova H stabla na krajevima prethodno nacrtanih H stabala (Slika 1.c), dalje širi mrežu elementarnih H stabala. Procedura ponovnog pozivanja programskog dela koji crta elementarna H stabla na krajevima prethodno nacrtanih H stabala se može ponoviti i dobiti još složeniju strukturu H stabala (slika 1.d). Ovo je primer kako se jedna složena struktura H stabala (slika 1.c) može na jednostavni način nacrtati, time što se više puta poziva isti deo programa koji crta samo jedno, elementarno H stablo. To je rekurzivni način rešavanja problema. Definišemo (ako možemo) programski deo za jedan element složene strukture, pa ga onda višestruko pozivamo da to ponovi. Kako se pozivi vrše iz prethodno realizovanog programskog dela, to se rekurzija hijerarhijski ponavlja.



Slika 1.1 Rekurzivni način crtanja složene strukture H stabla ponavljanjem crtanja jednog elementarnog H stabla (a)

VIDEO: ŠTA JE REKURZIJA?

Anim Aland: Recursion lecture 1 (13,51 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 2

Studija slučaja: Proračun faktoriijala

FAKTORIJEL I REKURZIVNI POZIV

Rekurzivni poziv je poziv istog metoda, samo sa različitim argumentom, umesto sa n , sada je sa $n-1$.

Faktoriyel broja n se može rekurzivno definisati na sledeći način:

$$0! = 1;$$

$$n! = n \times (n-1)!; n > 0$$

Šta ovo znači? Pa da idemo redom:

$$1! = 1 \times (1-1)! = 1 \times 0! = 1 \times 1 = 1$$

$$2! = 2 \times (2-1)! = 2 \times 1! = 2 \times 1 = 2$$

$$3! = 3 \times (3-1)! = 3 \times 2! = 3 \times 2 \times 1 = 6$$

.....

$$n! = n \times (n-1)! = n \times (n-1) \times (n-2) \times (n-2) \times \dots \times 1! \times 0!$$

Očigledno je da se $n!$ određuje na osnovu prethodno određenog $(n-1)!$, a da bi se odredio $(n-1)!$, treba da se prethodno odredi $(n-2)!$ itd. Sve do $0!$, tj. 1. Očigledno je da je ovo tipičan problem koji se može rešiti rekurzijom.

Neka metod `factorial(n)` računa faktoriyel nekog broja n . Ako se metod poziva za $n=0$ on odmah vraća rezultat 1, što predstavlja osnovni slučaj, tj. uslov prekida daljeg računanja. Ako se metod poziva za $n>0$, metod pojednostavljuje proračun time što ga deli na potproblem računanja faktoriijela za $n-1$. Potproblem je identičan originalnom problemu, ali je jednostavniji, tj. manji. Kako potproblem ima ista svojstva kao i originalni problem, možete pozivati isti metod samo sa različitim argumentom, tj. umesto sa n , sada sa $n-1$. To se naziva rekurzivnim pozivom.

```
If (n== 0)
    return = 1;
else
    return n * faktorial(n - 1);
```

Jedan rekurzivni poziv proizvodi veći broj drugih rekurzivnih poziva, jer se problem stalno deli na identične podprobleme, sve dok se ne dođe do potproblema koji zadovoljava uslov zaustavljanja rekurzije, tj. najjednostavnijiji slučaj problema. U slučaju proračuna faktoriijela $n!$ to je potproblem $0!$ jer je njegov rezultat, po definiciji, 1 i ne može se dalje deliti na još jednostavnije i identične potprobleme. Kada metod vrati rezultat svog osnovnog (najjednostavnijeg problema (u našem slučaju za $n=0$ $0!=1$) onda računa rezultat za jedan stepen složenijeg problema (u našem slučaju, $1! \times 0! = 1 \times 1 = 1$), koristeći vraćen rezultat prethodnog jednostavnijeg potproblema (u našem slučaju, $\text{factorial}(0)$). Proces rešavanja se tako nastavlja sve dok se ne dobije rešenje originalnog, tj. početnog problema. U našem slučaju to je $n!$, odn. $\text{factorial}(n)$. Njegovo rešenje je $n \times \text{factorial}(n-1)$.

KLASA COMPUTEFACTORTIAL

Klasa `ComputeFactorial` računa faktoriyel $n!$ nekog pozitivnog celog broja n . Metod `factorial()` je rekurzivan jer poziva sam sebe.

Ovde se daje listing klase **ComputeFactorial** koja obračunava faktoriyel nekog celog broja n , koji ne sme da bude negativan. Program pita korisnika da unese neki nenegativni ceo broj n (to znači pozitivni ili nula), za koji program računa njegov faktoriyel, tj. $n!$.

Slika 1 prikazuje dva slučaja korišćena programa (klase) **ComputeJava**, a za slučaj $n = 4$ i $n=10$. Prikazuje se unos broja i dobijeni rezultat.



Slika 2.1 Unos celog broja i dobijen njegov faktoriyel za dva slučaja

Metod **factorial()** (linije 16 – 21) direktno primenjuje matematičku definiciju rekurzije faktoriijela korišćenjem programa u Javi. Pozivanje metoda **factorial()** je rekurzivan jer sam sebe poziva. Parametar koji se prenosi u metodu **factorial()** se smanjuje za vrednost 1 kod svakog narednog poziva, a sve dok taj parametar ne dostigne vrednost 0.

Klasa **ComputeFactorial**:

```
1 import java.util.Scanner;
2
3 public class ComputeFactorial {
4     /** Main metod */
5     public static void main(String[] args) {
6         // Kreiranje objekta klase Scanner
7         Scanner input = new Scanner(System.in);
8         System.out.print("Enter a nonnegative integer: ");
9         int n = input.nextInt();
10    }
```



```

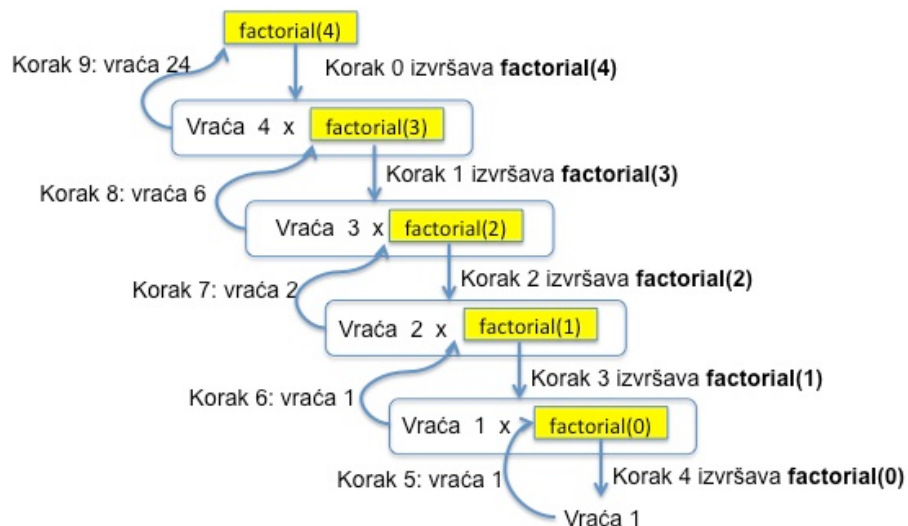
11 // Prikaz rezultata, tj. faktoriјala
12 System.out.println("Factorial of " + n + " is " + factorial(n));
13 }
14
15 /** Vraća faktoriјel za specificiran broj */
16 public static long factorial(int n) {
17     if (n == 0) // Osnovni slučaj
18         return 1;
19     else
20         return n * factorial(n - 1); // Rekurzivni poziv
21 }
22 }

```

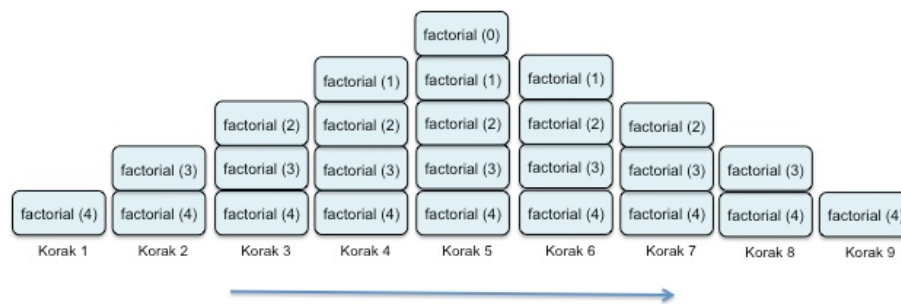
POSTUPAK IZVRŠENJA REKURZIJE

Pored direktne rekurziје, kada metod poziva sam sebe, postoji i indirektna rekurziја, kada metod poziva drugi metod, a on poziva prvi metod.

Slika 2 prikazuje kako se izvršavaju rekurzivni pozivi u slučaju da je $n = 4$. Na slici 3 prikazan je način korišćenja stekova u istom primeru.



Slika 2.2 Pozivanje metoda factorial(4) sa rekurzivnim pozivima metoda factorial



Slika 2.3 Korišćenje memoriјskih stekova za vreme izvršenja metoda factorial(4)

U ovom primeru je prikazan metod rekurziје u kome metod poziva sam sebe. To je tzv. **direktna rekurzija**. Međutim, postoji i **indirektna rekurzija**. Ona se javlja kada metod **A** poziva metod **B**, koji poziva metod **A**. Na ovaj način, indirektna rekurzija može da obuhvati više metoda. Na primer, metoda **A** poziva metod **B**, koji poziva metod **C**, a on poziva metod **A**.

Ako se ne obrati pažnja, može doći do *beskonačne rekurziје*, kada se ona nikada ne završava. Na primer:

```
public static long factorial(int n) {
    return n * factorial(n - 1);
}
```

VIDEO: REKURZIJA SA FAKTORIELIMA

Dave Feinberg: How Recursion Works (11,40 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PISANJE REKURZIVNIH METODA

Dave Feinberg: How To Write Recursive Methods (12,29 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO: PRAĆENJE REKURZIVNIH POZIVA 1

Dave Feinberg: Tracing Recursive Methods (12 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO: PRAĆENJE REKURZIVNIH POZIVA 2

Dave Feinberg: Tracing Recursive Methods 2

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADACI SA SAMOSTALNI RAD

Provera razumevanja rada sa rekurzijom

1. Šta štampaju prikazani programi? Šta je bazni slučaj, a šta rekurzivni korak?

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(  
            "Sum is " + xMethod(5));  
    }  
  
    public static int xMethod(int n) {  
        if (n == 1)  
            return 1;  
        else  
            return n + xMethod(n - 1);  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        xMethod(1234567);  
    }  
  
    public static void xMethod(int n) {  
        if (n > 0) {  
            System.out.print(n % 10);  
            xMethod(n / 10);  
        }  
    }  
}
```

Slika 2.4 Primeri za samoproveru znanja

2. Definišite rekurzivni postupak za izračunavanje izraza 2^n za neki ceo broj n .
3. Definišite rekurzivni postupak za izračunavanje x^n za ceo broj n i realan broj x .
4. Definišite rekurzivni postupak za izračunavanje $1 + 2 + 3 + \dots + n$ za neki ceo broj n .

▼ Poglavlje 3

Studija slučaja: Proračun Fibonačijevih brojeva

ŠTA JE FIBONAČIJEV NIZ?

Postoje slučajevi kada rekurzija omogućava kreiranje intuitivnih, direktnih i jednostavnih rešenja problema.

Fibonačijev niz je nazvan po srednjevekovnom matematičaru Leonardu Fibonačiju (Leonardo Fibonacci), koji je početno stvoren da bi modelovao rast broja populacije pacova. On se može iskoristiti u numeričkim optimizacijama i u raznim drugim oblastima. Fibonačijev niz počinje sa 0 i 1, a svaki naredni broj je zbir prethodna dva. Evo primera:

Fibonačijev niz: 0 1 1 2 3 5 8 13 21 34 55 89

Indeks: 0 1 2 3 4 5 6 7 8 9 10 11

Niz se može rekurzivno definisati na sledeći način:

```
fib(0) = 0;  
fib(1) = 1;  
fib(index) = fib(index - 2) + fib(index - 1); index >= 2
```

Kako naći Fibonačijev broj `fib(index)` za određenu vrednost indeksa `index`? Ako znate vrednosti za **`fib(index - 2)`** i **`fib(index - 1)`**, lako (sabiranjem) određujete broj **`fib(index)`**. Ovo pokazuje da se problem određivanja broja `fib(index)` svodi na određivanje brojeva **`fib(index - 2)`** i **`fib(index - 1)`**, što ukazuje da se problem rešava rekurzijom. Pri tome se vrednost za `index` smanjuje do 0 ili 1.

Osnovni slučaj je za vrednost `index = 0` ili za `index = 1`. Ako pozovete metod za ove vrednosti `index`, dobijate odmah rezultat (0, odn. 1). Kada zovete metod za vrednost `index >= 2`, on deli problem na dva problema računanja **`fib(index - 1)`** i **`fib(index - 2)`**. Za njihovo rešavanje se koriste rekurzivni pozivi (metod poziva samog sebe, samo menja indeks). Rekurzivni algoritam za proračun **`fib(index)`** se opisuje na sledeći način:

```
if (index == 0)  
    return 0;  
else if (index == 1)  
    return 1;  
else  
    return fib(index - 1) + fib(index - 2);
```

KLASA COMPUTEFIBONACCI

Proračun Fibonačijevih brojeva primenom rekurzije.

Listing programa (klase) ComputeFibonacci primenjuje prikazan algoritam za proračun Fibonačijevih brojeva

```
1 import java.util.Scanner;
2
3 public class ComputeFibonacci {
4     /** Main metod */
5     public static void main(String[] args) {
6         // Kreiranje objekta Scanner
7         Scanner input = new Scanner(System.in);
8         System.out.print("Enter an index for a Fibonacci number: ");
9         int index = input.nextInt();
10
11        // Nalaženje i prikaz Fibonačijevog broja
12        System.out.println("The Fibonacci number at index "
13            + index + " is " + fib(index));
14    }
15
16    /** Metod nalaženja Fibonačijevog broja */
17    public static long fib(long index) {
18        if (index == 0) // Osnovni slučaj
19            return 0;
20        else if (index == 1) // Osnovni slučaj
21            return 1;
22        else // Redukcija i rekurzivni pozivi
23            return fib(index - 1) + fib(index - 2);
24    }
25 }
```

Slika 1 prikazuje dobijen rezultat izvršenja programa **ComputeFibonacci.java** za slučaj vrednosti indeksa 1, 6 i 7.



Slika 3.1 Prikaz dobijenih rezultata izvršenja programa ComputeFibonacci.java za tri slučaja

Primena rekurzije u određivanju Fibonačijevog broja nije efikasna, jer se ponavljaju pozivi za proračun istog Fibonačijevog broja. Primenom petlje se može dobiti efikasniji način za određivanje Fibonačijevog broja. Ovde se daje primena rekurzije u određivanju Fibonačijevog broja isključivo iz pedagoških razloga.

REDOSLED REKURZIVNIH POZIVA

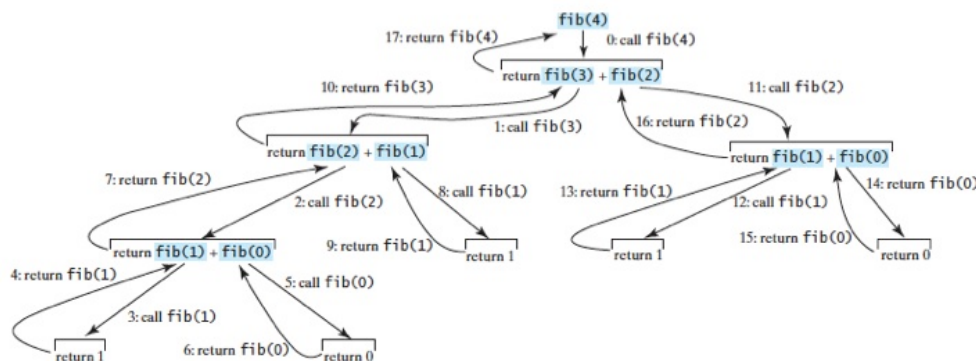
Računanje $\text{fib}(\text{index})$ zahteva otprilike dva puta više rekurzivnih poziva nego izvršenje $\text{fib}(\text{index}-1)$.

Prikazana program **ComputeFibonacci** ne prikazuje količinu posla koji se mora izvršiti da bi se dobio Fibonačijev niz. Slika 2 pokazuje redosled izvršenja rekurzivnih poziva metoda $\text{fib}()$ koji počinje pozivom **fib(4)**.

Poziv metod **fib(4)**, pokreće dva rekurzivna poziva **fib(3)** i **fib(2)** i onda se se vraća rezultat **fib(3) + fib(2)**. Međutim, postavlja se pitanje kojim se redosledom pozivaju ovi metodi?

U Javi, operandi se izvršavaju s leva na desno, te se **fib(2)** poziva posle potpunog izvršenja **fib(3)**. Na slici 2 naznačen je redosled pozivanja metoda **fib()**.

Primena rekurzije u problemu proračuna Fibonačijevim brojeva nije efikasna zbog velikog broja ponovljenih rekurzivnih poziva. Primena petlje je efikasnije rešenje. Ovde se koristi rekurzija iz edukativnih razloga.



Slika 3.2 Poziv $\text{fib}(4)$ pokreće rekurzivne pozive metoda $\text{fib}()$

Kao što je pokazano na slici 2, ima mnogo duplih rekurzivnih poziva. Na primer **fib(2)** se poziva dva puta a **fib(1)** tri puta, i **fib(0)** dva puta. U opštem slučaju, računanje **fib(index)** zahteva otprilike sva puta više rekurzivnih poziva nego što bi broj rekurzivnih poziva imalo izvršenje **fib(index-1)**. Ako koristite velike brojeve za indekse, broj rekurzivnih poziva se znatno povećava, kao što se vidi na ovoj tabeli:

Index: 2 3 4 10 20 30 40 50

Broj poziva; 3 5 9 177 21.891 2.692.537 331.160.281 2.075.316.483

VIDEO: FAKTORIJALI I REKURZIJA

What on Earth is Recursion? - Computerphile

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO: PRORAČUNI FIBONAČIJEVIH BROJAVA REKURZIJOM

Fibonacci Programming - Computerphile

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

SAMOPROVERA: ZADATAK 1

Provera razumevanja rada sa rekurzijom

1. Proveri rezultat sledeća dva programa:

```
public class Test {  
    public static void main(String[] args) {  
        xMethod(5);  
    }  
  
    public static void xMethod(int n) {  
        if (n > 0) {  
            System.out.print(n + " ");  
            xMethod(n - 1);  
        }  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        xMethod(5);  
    }  
  
    public static void xMethod(int n) {  
        if (n > 0) {  
            xMethod(n - 1);  
            System.out.print(n + " ");  
        }  
    }  
}
```

Slika 3.3 Dva uporedna primera

SAMOPROVERA: ZADATAK 2

Proverite svoje razumevanje rekurzije

Šta ne valja u donjem primeru?

```
public class Test {  
    public static void main(String[] args) {  
        xMethod(1234567);  
    }  
  
    public static void xMethod(double n) {  
        if (n != 0) {  
            System.out.print(n);  
            xMethod(n / 10);  
        }  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Test test = new Test();  
        System.out.println(test.toString());  
    }  
  
    public Test() {  
        Test test = new Test();  
    }  
}
```

Slika 3.4 Dva primera sa greškom

✓ Poglavlje 4

Rešavanje problema upotrebom rekurzije

REKURZIVNO RAZMIŠLJANJE

Mnoge probleme možete rešiti rekurzijom, ako rekurzivno razmišljate.

Svi rekurzivni metodi imaju sledeće karakteristike:

1. Metod upotrebljava **if-else** ili **switch** iskaze koji vode do različitih slučajeva.
2. Jedna ili više osnovni slučajevi (najjednostavniji slučaj) se koriste za zaustavljanje rekurzije.
3. Svaki rekurzivni poziv smanjuje početni problem, dovodeći ga bliže do osnovnog slučaja, sve dok ne postane takav slučaj.

Da bi se neki problem rešio rekurzijom, on se mora razbiti na potprobleme. Svaki podproblem je identičan početnom problemu ali je manje veličine. Za rešavanje svakog podproblema primenjuje se isti pristup rekurzivnog rešavanja.

Razmotrimo problem štampanja jedne poruke n puta. Možete razbiti problem na dva potproblema: jedan je da se odštampa poruke sada, a drugi je da se odštampa $n-1$ puta. Drugi problem je isti kao početni problem, ali je manji po veličini. Osnovni slučaj za problem je $n == 0$. Problem se može rešiti upotrebom rekurzije na sledeći način:

Metod **nPrintln()**:

```
public static void nPrintln(String message, int times) {
    if (times >= 1) {
        System.out.println(message);
        nPrintln(message, times - 1);
    } // Osnovni slučaj je times == 0
}
```

Rekurzija je svuda oko nas. Pokušajte da razmišljate rekurzivno. Na primer, kada pijete kafu. Tu proceduru možete da opišite rekurzijom. Metod **drinkCoffee()**:

```
public static void drinkCoffee(Cup cup) {
    if (!cup.isEmpty()) {
        cup.takeOneSip(); // Uzmite gutalj
        drinkCoffee(cup);
    }
}
```

```
}
```

Pretpostavimo da je šolja objekat šolje za kafu sa objektnim metodima `isEmpty()` i `takeOneSip()`. Možete problem podeliti na dva problema. Prvi je uzeti gutalj kafe. Drugi je pijenje ostatka kafe, a to je identično početnom problemu, samo je malo manji. Osnovi slučaj, kada rekurzija prestaje, jer kada šolja ostaje prazna.

PROVERA DA LI JE STRING PALINDROM

Palindrom je string (tekst) koji se isto čita i sleva na desno, i s desna na levo

Ako rekurzivno razmišljate, možete rekurzijom da rešite mnoge probleme. Na primer problem palindroma. To je string (tekst) koji se isto čita i sleva na desno, i s desna na levo. Na primer "mama" i "tata". Problem proveravanja da li je neki string palindrom može se podeliti u dva podproblema:

1. Provera da li je prva oznaka (character) jednaka poslednjoj oznaci stringa
2. Ignoriši dve krajnje oznake stringa i proveriti da li je preostali tekst palindrom.

Drugi podproblem je isti kao što je početni problem samo je manje veličine. Postoje dva osnovna slučaja:

1. Dve krajnje oznake nisu jednake
2. String je dužine 0 ili 1.

Programsko rešenje ovog problema dato je u listingu klase **RecursivePalindromeUsingSubstring**

Metod `substring()` u liniji 8 kreira novi string jednak početnom stringu, sem što nema dve krajnje oznake. Njegova provera da li je palindrom je identična proveru početnog stringa.

Klasa **RecursivePalindromeUsingSubstring**:

```
1 public class RecursivePalindromeUsingSubstring {
2     public static boolean isPalindrome(String s) {
3         if (s.length() <= 1) // Base case
4             return true;
5         else if (s.charAt(0) != s.charAt(s.length() - 1)) // Base case
6             return false;
7         else
8             return isPalindrome(s.substring(1, s.length() - 1));
9     }
10
11     public static void main(String[] args) {
12         System.out.println("Is moon a palindrome? "
13             + isPalindrome("moon"));
14         System.out.println("Is noon a palindrome? "
```

```

15     + isPalindrome("noon"));
16 System.out.println("Is a a palindrome? " + isPalindrome("a"));
17 System.out.println("Is aba a palindrome? " +
18     isPalindrome("aba"));
19 System.out.println("Is ab a palindrome? " + isPalindrome("ab"));
20 }
21 }

```

VIDEO: REKURZIJA U JAVI

Derek Banas: Java Recursion (14,11 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK 1 - STEPENOVANJE PROSLEĐENOG BROJA

Napraviti metodu za stepenovanje prosleđenog broja na prosleđeni broj koristeći rekurziju

Koristeći rekurziju napraviti metodu koja prima broj koji će biti stepenovan kao i sam stepen i proračunava broj na taj stepen.

Potpis rekurzivne metode koju treba napisati je sledeći:

```
public int stepenuj(int exp, int stepen);
```

Kao što je navedeno u prethodnom primeru, dobra je praksa prvo definisati bazni slučaj što bi u ovom slučaju mogao da bude jedan od sledeća dva:

1. 0-ti stepen bilo kog broja je uvek 1

```

if (stepen == 0) {
    return 1;
}

```

2. 1. stepen bilo kog broja je uvek taj broj

```

if (stepen == 1) {
    return exp;
}

```

Dovoljno je navesti samo 1. slučaj kao bazni.

Ukoliko funkcija nije pozvana sa argumentima koji obezbeđuju bazni slučaj onda treba rekurzivno izračunati vrednost stepena koji je za jedan manji i pomnožiti je sa eksponentom.

Naredba kojom se to izvršava je u kodu

```
exp * stepenuj(exp, stepen - 1);
```

ZADATAK 1 - REŠENJE

Rešenje zadatka koji odredjuje N-ti stepen zadanog broja M

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package z02;

/**
 *
 * @author Aleksandra
 */
public class StepenMain {

    public StepenMain() {
        System.out.println(stepenuj(5, 3));
    }

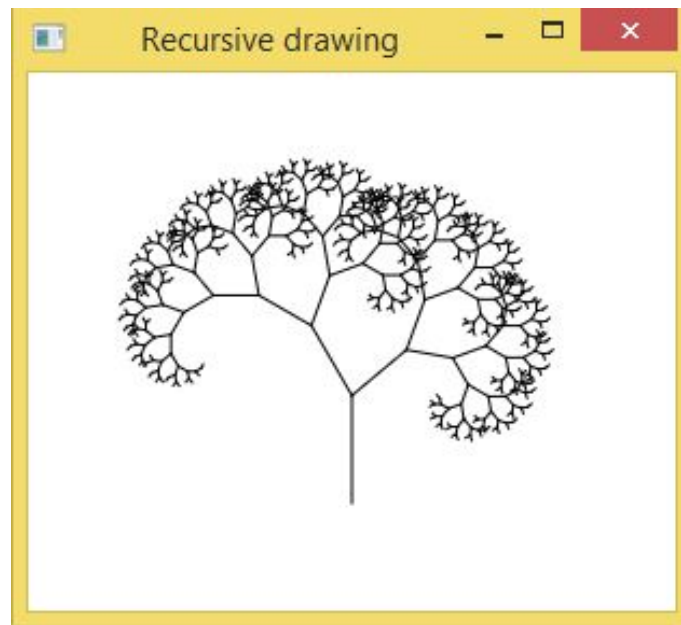
    // prima eksponent i stepen i vraca stepenovanu vrednost eksponenta
    public int stepenuj(int exp, int stepen) {
        if (exp == 0) {
            System.out.println("Eksponent ne moze biti 0.");
        }
        if (stepen == 0) {
            return 1;
        } else if (stepen == 1) {
            return exp;
        }
        return exp * stepenuj(exp, stepen - 1);
    }

    public static void main(String[] args) {
        new StepenMain();
    }
}
```

ZADATAK 2 - REKURZIVNA GRAFIKA

Cilj ovog zadatka je pravljenje rekurzivne grafike

Napraviti rekurzivnu grafiku po sledećoj slici:



Slika 4.1 Rekurzivna grafika

Glavna ideja zadatka je da se uoči broj grana koje svaki put crtamo iz jednog čvora.

U ovom slučaju, na kraj svake grane dodajemo nove DVE grane.

Zbog toga, pošto se nacrtava uvek po dve grane na kraj jedne imamo (kao u slučaju određivanja Fibonačijevog niza) dva poziva funkcije sa različitim argumentima.

```
public void draw(double x, double y, double len, double angle) {  
    gc2d.setLineWidth(1);  
    if (len > 2) {  
        double x1 = x + len * Math.cos(Math.toRadians(angle));  
        double y1 = y - len * Math.sin(Math.toRadians(angle));  
  
        gc2d.strokeLine(x, y, x1, y1);  
        draw(x1, y1, len * 0.75, angle + 30);  
        draw(x1, y1, len * 0.66, angle - 50);  
    }  
}
```

ZADATAK 2- REŠENJE

Rešenje zadatka sa rekurzivnom grafikom

```
/*  
 * To change this license header, choose License Headers in Project Properties.  
 * To change this template file, choose Tools | Templates  
 * and open the template in the editor.  
 */  
package z06;
```

```
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

/**
 *
 * @author Aleksandra
 */
public class TreeCanvas extends Canvas {

    GraphicsContext gc2d = getGraphicsContext2D();

    public TreeCanvas() {
        super();
        setWidth(300);
        setHeight(250);
        draw(150, 200, 50, 90);
    }

    public void draw(double x, double y, double len, double angle) {
        gc2d.setLineWidth(1);
        if (len > 2) {
            double x1 = x + len * Math.cos(Math.toRadians(angle));
            double y1 = y - len * Math.sin(Math.toRadians(angle));

            gc2d.strokeLine(x, y, x1, y1);

            draw(x1, y1, len * 0.75, angle + 30);
            draw(x1, y1, len * 0.66, angle - 50);
        }
    }
}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package z06;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

/**
 *
 * @author Aleksandra
 */
```

```
public class FXRecursion extends Application {

    TreeCanvas tree = new TreeCanvas();

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();

        Scene scene = new Scene(root, 300, 250);

        root.getChildren().add(tree);

        primaryStage.setTitle("Recursive drawing");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

ZADATAK 3

Napisati rekurzivni metod koji određuje broj pojavljivanja odgovarajućeg slova u stringu.

Napisati rekurzivni metod koji određuje broj pojavljivanja odgovarajućeg slova u stringu, pri čemu metod ima zaglavlje:

public static int count(String str, char a)

Na primer, count("Welcome", 'e') vraća rezultat 2.

Napisati test program koji od korisnika zahteva da unese string i karakter, a zatim određuje i prikazuje broj pojavljivanja datog karaktera u unetom stringu.

Ideja je da se koristi promenljiva result i da se ona inicijalizuje na 0.

Ukoliko je string prazan, tj. dužine 0 onda je krajnji rezultat upravo početna vrednost koja je skladištena u promenljivoj result.

U suprotnom, treba proći kroz ceo string i prebrojati karaktere koji odgovaraju traženom karakteru. Kako? Rekurzivno.

Ukoliko je string dužine veće od 0 onda ima smisla raditi ispitivanje stringa

```
if (str.length() > 0) {
```

```
    result = count(str.substring(1), a) // uzimamo u obzir rekurzivni
```

poziv koji se odnosi na podstring sa jednim karakterom manje i na tu vrednost dodajemo 0 ili 1
+ ((str.charAt(0) == a) ? 1 : 0); //1 dodajemo ako je početni karakter jednak traženom, 0 u suportom.
}

ZADATAK 3 - REŠENJE

Ršenje zadatka koji određuje broj pojavljivanja odgovarajućeg slova u stringu.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package z07;

import java.util.Scanner;

/**
 *
 * @author Aleksandra
 */
public class FindMain {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String s = input.nextLine();
        System.out.print("Enter a character: ");
        char ch = input.nextLine().charAt(0);
        int times = count(s, ch);
        System.out.println(ch + " appears " + times + (times > 1 ? " times " : "
time ") + "in " + s);
        input.close();
    }

    public static int count(String str, char a) {
        int result = 0;
        if (str.length() > 0) {
            result = count(str.substring(1), a)
                + ((str.charAt(0) == a) ? 1 : 0);
        }

        return result;
    }
}
```


ZADACI ZA SAMOSTALNI RAD

Samostalno rešavanje problema rekurzijom

1. Napisati rekurzivnu metodu koja leksikografski (kao u rečniku) poredi dva stringa
2. Napisati rekurzivnu metodu koja računa broj parnih cifara broja
3. Napisati rekurzivnu metodu koja računa broj samoglasnika u stringu
4. Napisati rekurzivnu metodu koja računa broj neparnih elemenata u nizu brojeva
5. Napisati rekurzivnu metodu koja proverava da li se uneti karakter nalazi u unetom stringu

▼ Poglavlje 5

Rekurzivni pomoćni metodi

ŠTA SU REKURZIVNI POMOĆNI METODI?

Ponekad, mogu se pronaći rekurzivna rešenja malim promenama početnog problema. Ovaj novi metod se naziva rekurzivni pomoćni metod.

Rekurzivni pomoćni metodi (**recursive helper methods**) su vrlo korisni za nalaženje rekurzivnih rešenja problema rada sa tekstovima i nizovima. Rekurzivni metod **isPalindrome()** i klasi **RecursivePalindromeUsingSubstring** nije efikasan jer kreira novi string za svaki rekurzivni poziv. Da bi se izbeglo formiranje novih stringova, možete iskoristiti indekse prve i zadnje oznake stringa za određivanje opsega podstringa. Ova dva indeksa je potrebno preneti u rekurzivni metod. Umesto ranijeg metoda **isPalindrome(String s)**, sada morate da kreirate metod: **isPalindrome(String s, int low, int high)**, i to je prikazano na listingu klase **RecursivePalindrome**.

Definisana su dva metoda **isPalindrome()**. Prvi, **isPalindrome(String s)**, proverava da li je string palindrom, a drugi, **isPalindrome(String s, int low, int high)**, proverava dali je podstring **s(low , high)** palindrom. Prvi metod prenosi string **s** sa **low=0** i **high=s.length()-1** drugom metodu. Drugi metod se može rekurzivno pozivati radi provere uslova palindroma u stalno manjim podstringovima. Praksa je pri rekurzivnom programiranju da se definiše drugi metod koji prima dodatne parametra. Takav metod je poznat kao rekurzivni pomoćni metod. Ovi metodi su vrlo korisni kod rekurzivnog rešavanja problema koji se javljaju sa strinovima i nizovima.

Klasa **RecursivePalindrom**:

```
1 public class RecursivePalindrome {
2     public static boolean isPalindrome(String s) {
3         return isPalindrome(s, 0, s.length() - 1);
4     }
5
6     private static boolean isPalindrome(String s, int low, int high) {
7         if (high <= low) // Base case
8             return true;
9         else if (s.charAt(low) != s.charAt(high)) // Base case
10            return false;
11        else
12            return isPalindrome(s, low + 1, high - 1);
13    }
14
15    public static void main(String[] args) {
```

```

16 System.out.println("Is moon a palindrome? "
17   + isPalindrome("moon"));
18 System.out.println("Is noon a palindrome? "
19   + isPalindrome("noon"));
20 System.out.println("Is a a palindrome? " + isPalindrome("a"));
21 System.out.println("Is aba a palindrome? " + isPalindrome("aba"));
22 System.out.println("Is ab a palindrome? " + isPalindrome("ab"));
23 }
24 }

```

PODSEKVENCE STRINGOVA

Primena pomoćnog rekurzivnog metoda

Treba implementirati metodu definisanu na sledeći način:

```
public static String subsequences(String word)
```

Na primer, rezultat pozivanja `subsequences("abc")` treba da bude string `"abc,ab,bc,ac,a,b,c,"`. Treba primetiti i da je prazan string isto podsekvencu, tako da imamo jedan zarez i na kraju rezultata.

Problem se lako može rešiti rekurzivno. Ako uzmemo prvo slovo reči, imamo jedan skup sa svim podsekvencama koje sadrže to slovo i drugi skup svih podsekvenci koje ne sadrže to slovo. Ova dva skupa sadrže sve moguće podsekvence.

```

1 public static String subsequences(String word) {
2     if (word.isEmpty()) {
3         return ""; // bazni slucaj
4     } else {
5         char firstLetter = word.charAt(0);
6         String restOfWord = word.substring(1);
7
8         String subsequencesOfRest = subsequences(restOfWord);
9
10        String result = "";
11        for (String subsequence : subsequencesOfRest.split(",", -1)) {
12            result += "," + subsequence;
13            result += "," + firstLetter + subsequence;
14        }
15        result = result.substring(1); // brisanje zareza sa pocetka stringa
16        return result;
17    }
18 }

```

Sveli smo rešavanje problema na rešavanje potproblema – podsekvence ostatka stringa koji se dobija kada se ukloni prvo slovo– a zatim smo koristili rešenje tog potproblema za

rešavanje problema, tako što smo uzeli svaku podsekvencu i napravili dve nove, jednu sa prvim slovom, drugu bez..

Sledeći kod je rešenje primenom direktne rekurzije:

```
private static String subsequencesAfter(String partialSubsequence, String word) {  
    if (word.isEmpty()) {  
        //bazni slucaj  
        return partialSubsequence;  
    } else {  
        //rekurzivni korak  
        return subsequencesAfter(partialSubsequence, word.substring(1))  
            + ","  
            + subsequencesAfter(partialSubsequence + word.charAt(0),  
word.substring(1));  
    }  
}
```

Potrebna nam je početna vrednost za string podsekvenci. Zato koristimo pomoćni metod subsequencesAfter, a korisnik poziva traženi metod subsequences.

```
public static String subsequences(String word) {  
    return subsequencesAfter("", word);  
}
```

ZADACI ZA INDIVIDUALAN RAD

Cilj zadatka za individualni rad jeste provežbavanje stečenog znanja na vežbama i predavanjima

Zadatak 1

Napisati program koji upotrebom rekurzivne funkcije ispituje da li je dati string palindrom.

Zadatak 2

Za dati broj pomoću rekurzivne funkcije ispisati broj koji se piše istim ciframa ali u obrnutom poretaku.

Zadatak 3

Napisati rekurzivnu funkciju koja vraća odgovor na pitanje da li je broj cifara njenog argumenta paran.

Zadatak 4

Napisati rekurzivnu funkciju koja računa zbir cifara na neparnim pozicijama, računajući skraja (sdesna nalevo).

▼ Poglavlje 6

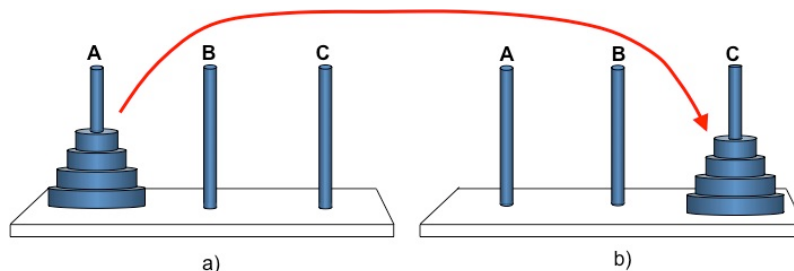
Studija slučaja: Hanojske kule

OPIS ZADATKA

Problem Hanojskih kula je klasičan problem koji se može rešavati jednostavno upotrebom rekurzije, a teško ih je rešiti na drugi način.

Hanojske kule su čuveni rekurzivni problem koji često koristi kod objašnjavanja rekurzije u programiranju. Problem se sastoji u sledećem: Dat je određen broj okruglih diskova, različitih prečnika. Svi diskovi u sredini imaju otvor. Kroz taj otvor diskovi mogu da se smeštaju na stubove (slika 1a). Treba prebaciti diskove sa stuba A na stub C. U jednom trenutku sme da se prebacuje samo jedan disk, a ni u jednom trenutku se ne sme veći disk postaviti na manji.

U Indiji postoji legenda da u nekom udaljenom hramu majmuni rade dan i noć u pokušaju da prebace 64 diska od zlata sa jednog na drugi dijamantski stub. U trenutku kada budu završili, dolazi do smaka sveta.



Slika 6.1 Hanojske kule - opis zadatka

Inicijalnu postavku diskova na stubu ćemo nazvati stablom. Tokom prebacivanja diskova sa jednog na drugi stub stalno ćete nailaziti na manja stabla, koja ćemo nazvati podstablima. Na primer, ako pokušate da prebacujete četiri diska, jedan od koraka koji će se javiti tokom prebacivanja je prikazan na slici 1b.

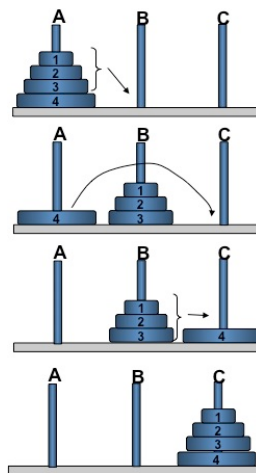
Tokom rešavanja slagalice će se ovakva podstabla javiti puno puta. To se dešava zato što je kreiranje ovakvih podstabala, jedini način da se veći disk prebaci na drugi stub. Svi manji diskovi moraju biti postavljeni na međustub.

REKURZIVNI ALGORITAM

Algoritam je sledeći: (1) Prebaciti podstablo koje sadrži $n-1$ diskova sa A na B. (2) Preostali (najveći) disk prebaciti sa A na C. (3) Prebaciti podstablo sa B na C.

Rešenje ovog problema se može izraziti rekurzivno, ako se koristi pojam podstabala, koji smo uveli. Pretpostavimo da želite da prebacite diskove sa stuba A na stub C. Pretpostavimo da na raspolaganju imate pomoćni stub B. Na stubu A se nalazi n diskova. Algoritam je sledeći (slika 2):

1. Prebaciti podstablo koje sadrži $n-1$ diskova sa A na B.
2. Preostali (najveći) disk prebaciti sa A na C.
3. Prebaciti podstablo sa B na C.



Slika 6.2 Postupak rešavanja

Na početku se diskovi nalaze na stubu A, pomoćni stub je B, a odredište stub C.

Najpre se podstablo koje se sastoji od diskova 1, 2 i 3 prebacuje na pomoćni stub B. Nakon toga se najveći disk 4, prebacuje na stub C. Posle toga se podstablo prebacuje sa B na C.

Ovo rešenje ne objašnjava kako da podstablo diskova 1, 2 i 3 prebacimo sa A na B. Sa druge strane prebaciti tri diska je lakše nego prebaciti četiri diska.

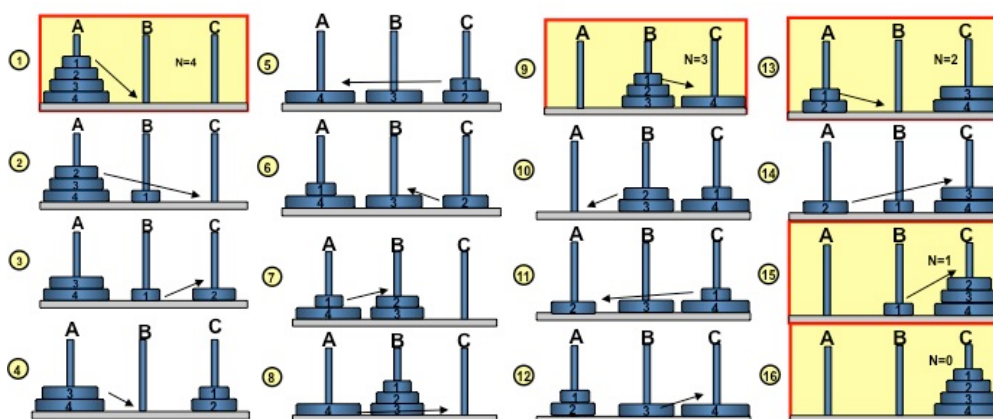
Rešenje je slično sa prethodnim. Treba prebaciti podstablo koje se sada sastoji od dva diska 1 i 2 sa na C, a onda disk 3 prebaciti sa A na B. Posle toga treba vratiti podstablo sa C na B.

Kako ćete prebaciti podstablo od dva diska sa A na C? Tako što ćete podstablo koje se sastoji samo od jednog diska 1 prebaciti sa A na B. Ovo je osnovni slučaj. Nakon toga ćete veći disk 2 prebaciti sa A na C itd.

PROBLEM SA ČETIRI DISKA

Veći problem se rešava što ga podelimo na dva manja. Problem sa četiri diska se dovodi do problema sa tri diska, i postupak se dalje nastavlja.

Na slici 3 dat je postupak rešavanja problema u slučaju četiri diska.



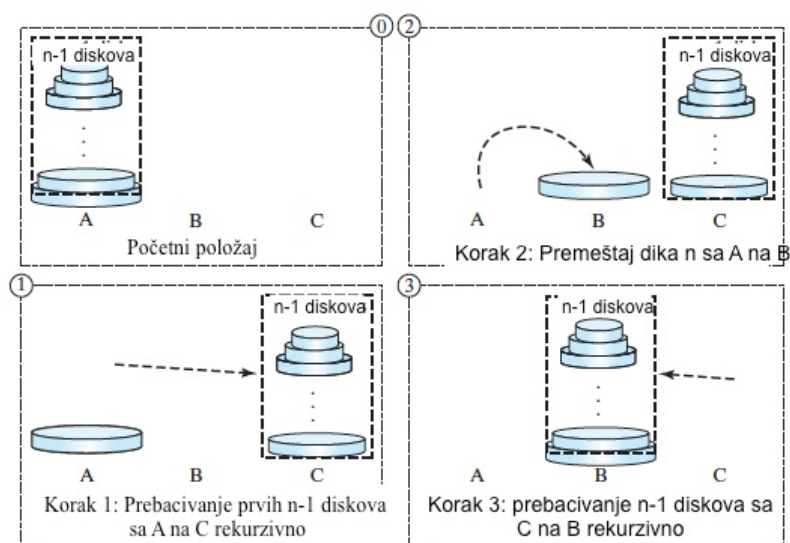
Slika 6.3 Postupak rešavanja problema sa četiri diska

ALGORITAMSKO REŠAVANJE PROBLEMA SA N DISKOVA

Algoritam: (1) Prebaciti $n-1$ diskova sa A na C rekursivno uz pomoć stuba B. (2) Preostali (najveći) disk prebaciti sa A na C. (3) Prebaciti podstablo sa B na C

Osnovi slučaj problema je kada je $n=1$. Ako je $n=1$, prebacujete disk sa A u B. Kada je $n>1$, možete da podelite početni problem na sledeća tri podproblema (slika 4):

1. Prebaciti $n-1$ diskova sa A na C rekursivno uz pomoć stuba B.
2. Preostali (najveći) disk prebaciti sa A na C.
3. Prebaciti podstablo sa B na C



Slika 6.4 Podela problema na tri podproblema

Sledeći metod prebacuje n diskova sa pozicije **fromTower** na poziciju **toTower** uz pomoć pomoćnog stuba **auxTower**:

```
void moveDisks(int n, char fromTower, char toTower, char auxTower)
```

Algoritam ovog metoda se može ovako da predstavi:

```
if (n == 1) // Uslov završetka
    Move disk 1 from the fromTower to the toTower;
else {
    moveDisks(n - 1, fromTower, auxTower, toTower);
    Move disk n from the fromTower to the toTower;
    moveDisks(n - 1, auxTower, toTower, fromTower);
}
```

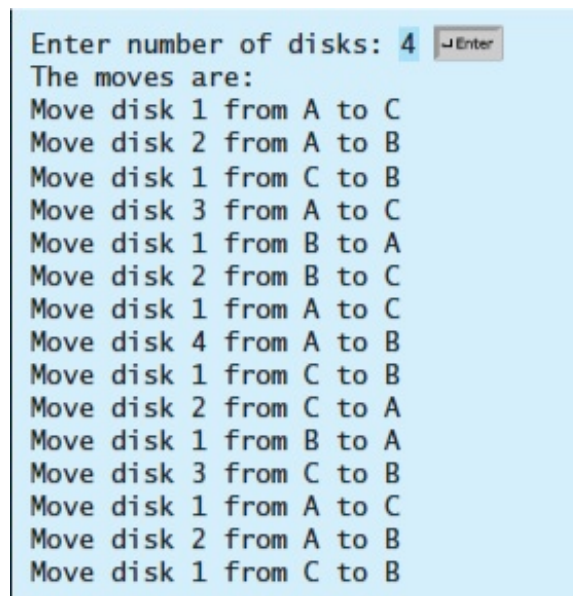
Stubovi A, B i C se tokom rešavanja različito dodeljuju pozicijama **fromTower**, **toTower** i **auxTower**.

KLASA TOWERSOFHANOI

Klasa TowersOfHanoi rekurzivno poziva metod moveDisks() koji primenjuje algoritam rekurzije premeštanjem diskova.

Listing klase TowersOfHanoi prikazuje program koji pita korisnika da upiše broj diskova i rekurzivno poziva metod **moveDisks()** koji prikazuje rešenje pomeranja diskova (slika 5).

Problem je prirodno rekurzivni. Upotrebom rekurzije moguće je naći prirodno, jednostavno rešenje. Bez rekurzije, problem bi se teže rešavao. . .



```
Enter number of disks: 4
The moves are:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
Move disk 4 from A to B
Move disk 1 from C to B
Move disk 2 from C to A
Move disk 1 from B to A
Move disk 3 from C to B
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
```

Slika 6.5 Prikaz rezultata izvršenja programa TowersOfHanoi

Klasa **TowersOfHanoi**:

```
1 import java.util.Scanner;
2
```



```

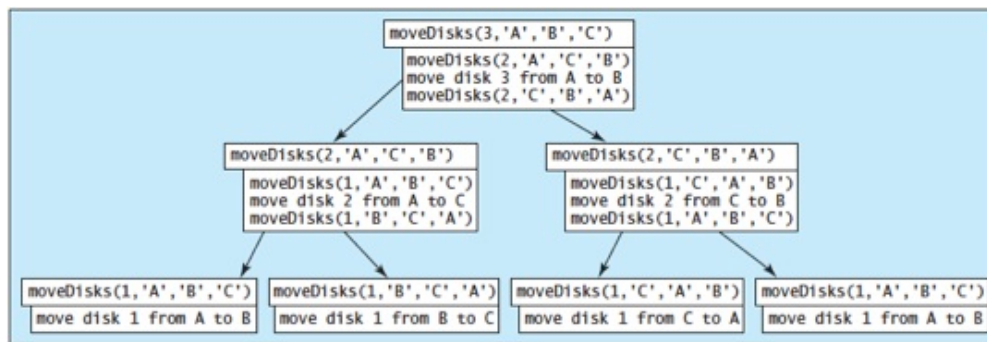
3 public class TowersOfHanoi {
4     /** Main metod */
5     public static void main(String[] args) {
6         // Kreiranje objekta klase Scanner
7         Scanner input = new Scanner(System.in);
8         System.out.print("Enter number of disks: ");
9         int n = input.nextInt();
10
11         // Nalaženje rekurzivnog rešenja
12         System.out.println("The moves are:");
13         moveDisks(n, 'A', 'B', 'C');
14     }
15
16     /** Metod nalaženja rešenja za pomeranje n diskova
17         od fromTower u toTower sa auxTower */
18     public static void moveDisks(int n, char fromTower,
19         char toTower, char auxTower){
20         if (n == 1) // Uslov zaustavljanja traženja
21             System.out.println("Move disk " + n + " from " +
22                 fromTower + " to " + toTower);
23         else {
24             moveDisks(n - 1, fromTower, auxTower, toTower);
25             System.out.println("Move disk " + n + " from " +
26                 fromTower + " to " + toTower);
27             moveDisks(n - 1, auxTower, toTower, fromTower);
28         }
29     }
30 }

```

PROCES REKURZIVNOG REŠAVANJA PROBLEMA SA TRI DISKA

Rekurzija, svojim nivoom apstrakcije, skriva iteracije i druge detalje od korisnika.

Na slici 6 prikazan je proces rekurzivnog rešavanja Hanojskih kula za slučaj 3 diska ($n=3$). Kao što vidite, jednostavnije je napisati program nego pratiti sve rekurzivne pozive. Sistem koristi memorijski stek za upravljanje pozivima. Rekurzija, svojim nivoom apstrakcije, skriva iteracije i druge detalje od korisnika.



Slika 6.6 Prikaz procesa rekurzije pri pozivu metoda moveDisks(3, 'A', 'B', 'C')

VIDEO: HANOJSKE KULE

Video from edX Course: MITx: 6.00x Introduction to Computer Science and Programming <https://www.edx.org/course/mit/6-00x/...cience/586> (5,57 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADACI ZA SAMOSTALNI RAD

Radi vežbe, uradite ove zadatke

1. Hanojske kule: Promenite program TowersOfHanoi.java tako da program pronalazi broj pokreta (prebacivanja diska) koji je potreban za prebacivanje n diskova sa kule A na kulu B (Upotrebite statičku promenljivu i inkrement uvek kada se pozove metod).

2. Decimalni u binarni brojevi: Napišite rekurzivni metod koji pretvara ceo broj u binarni broj u obliku teksta (string). Zaglavlje metoda je:

```
public static String decimalToBinarnz(int value)
```

Napišite program za testiranje koji pita korisnika da unese ceo broj i koji prikazuje odgovarajuću binarnu vrednost.

3. Binarni u decimalni broj : Napišite rekurzivni metod koji pretvara binarni broj u vidu teksta (string) u decimalni ceo broj. Zaglavlje metoda je:

```
public static int binaryToDecimal(String binaryString)
```

Napišite program za testiranje koji pita korisnika da unese binarni broj u vidu teksta i prikazuje njegovu odgovarajuću decimalnu vrednost.

▼ Poglavlje 7

Rekurzija i iteracija

KADA KORISTITI REKURZIJU, A KADA ITERACIJE?

Rekurzija je alternativni oblik upravljanja programom. Ona je ustvari ponavljanje programskih instrukcija, ali bez korišćenja programskih petlji.

Kada koristite petlje, vi definišete i telo petlje. Izvršenje instrukcija u telu petlje se ponavlja više puta, prilikom svakog izvršenja petlje. U slučaju rekurzije, metod poziva samog sebe više puta (menjajući vrednost nekog od argumenata) i na taj način se vrši ponovljeno izvršenje određenih programskih instrukcija. Da li će se rekurzija nastaviti odlučuje iskaz izbora, koji sadrži uslov za završetak rekurzije.

Primena rekurzije zahteva dosta računarskih resursa. Uvek kada program poziva neki metod, sistem mora da obezbedi memorijski prostor za sve njegove lokalne promenljive i parametre. To može da zahteva znatnu memoriju, a samim tim i vreme za upravljanje tom memorijom.

Svaki problem koji se može rešiti rekurzijom, može se rešiti i bez rekurzije. Upotrebom iteracija (ponavljanjem izvršenja tela petlji). Zašto onda koristimo rekurziju? Kod nekih problema, primena rekurzije dovodi do čistog, jasnog rešenja problema koji je po prirodi rekurzivan, a koji bi se iterativno teže i komplikovanije rešavao, kao to je to na primer problem Hanojskih kula.

Kako odlučiti da li u određenom problemu koristiti rekurziju ili iteraciju? To zavisi od prirode problema, i vašeg razumevanja problema. Po pravilu, biramo pristup koji nam se odmah intuitivno nameće, a koje odražava problem.

Po pravilu, iterativne metode su efikasnije, i ako je njihova primena jasna u određenom problemu, treba ih koristiti.

Iterative isPalindrome Method:

```
public boolean isPalindrome()
{
    int start = 0;
    int end = text.length() - 1;
    while (start < end)
    {
        char first = Character.toLowerCase(text.charAt(start));
        char last = Character.toLowerCase(text.charAt(end));
        if (Character.isLetter(first) && Character.isLetter(last))
```

```
{
    // Both are letters.
    if (first == last)
    {
        start++;
        end--;
    }
    else
        return false;
}
if (!Character.isLetter(last))
    end--;
if (!Character.isLetter(first))
    start++;
}
return true;
}
```

ZADACI ZA SAMOSTALNI RAD

Odgovorite na sledeća pitanja

1. Koja od sledećih tvrdnji je tačna?

Svaki rekurzivni metod se može pretvoriti u nerekurzivni metod.

Rekurzivni metodi zahtevaju više vremena i memorije za izvršenje nego nerekurzivni metodi.

Rekurzivni metodi su uvek jednostavniji od nerekurzivnih metoda.

Kod rekurzivnih metoda uvek postoji iskaz izbora koji proverava da se došlo do osnovnog (krajnjeg) slučaja.

2. Šta je uzrok za izuzetak prouzrokovan nedostatkom stekova?

3. Napisati rekurzivni i odgovarajući iterativni program koji:

određuje zbir prvih n brojeva

određuje proizvod n brojeva

određuje $N!$

▼ Poglavlje 8

Repna rekurzija

ŠTA JE REPNA REKURZIJA?

Metod repne rekurzije je efikasan način smanjenja potrebne veličine steka.

Pod repnom rekurzijom (tail recursion) podrazumeva se rekurzija koja nema zaostale operacije na čekanju, a koje bi trebalo izvršiti po završetku jednog rekurzivnog poziva, kao što je prikazano na slici 1.a. Metod B na slici 1.b nije metod repne rekurzije jer ima operacije na čekanju, a koje se moraju izvršiti kada se završi rekurzivni poziv.

Repna rekurzija je poželjna, jer se metod završava odmah po završetku poslednjeg rekurzivnog poziva. Zbog toga, nema potrebe da se memorišu u steku neki međupozivi.

Primenom pomoćnih parametara, nerepni rekurzivni metod se može pretvoriti u repni rekurzivni metod. Ovi parametri sadrže rezultate. Ideja je da se na ovaj način izbegnu operacije na čekanju.

Na ovaj način, možete definisati novi pomoćni rekurzivni metod sa pomoćnim parametrima. Ovaj novi metod može da koristi isti naziv kao originalni metod, ali mora da koristi drugu signaturu, tj. druge argumente.



Slika 8.1 Metod repne rekurzije i rekurzija bez repa

Repka rekurzija je poželjna jer se metod završava kada se se završi i poslednju rekurzivni poziv. Nema potrebe da se memorišu među pozivi u steku. Kompajleri mogu da optimiziraju repnu rekurziju.

PRIMER PRIMENE REPNE REKURZIJE

Nerepni metod rekurzije se često može da konvertuje u repnu rekurziju upotrebom pomoćnih parametara

Nerepni metod rekurzije se često može da konvertuje u repnu rekurziju upotrebom pomoćnih parametara. Ovi parametri sadrže rezultat. Ideja je da se ubace odložene operacije u pomoćne parametre tako da rekurzivni pozivi više nemaju odloženu operaciju. Možete definisati novi metod pomoćne rekurzije sa pomoćnim parametrima. Na ovaj način se redefiniše originalni metod sa istim imenom ali sa različitim potpisom. Na primer, ovde se daje ranije korišćeni metod **factorial()** u listingu klase **ComputeFactorialTailRecursion**, ali sada uz korišćenje rekurzije.

Klasa **ComputeFactorialTailRecursion**:

```
1 public class ComputeFactorialTailRecursion {
2     /** Vraća faktoriyel određenog broja */
3     public static long factorial(int n) {
4         return factorial(n, 1); // Poziv pomoćnog metoda
5     }
6
7     /** Pomoćni repni rekurzivni metod za faktoriyel */
8     private static long factorial(int n, int result) {
9         if (n == 0)
10             return result;
11         else
12             return factorial(n - 1, n * result); // Rekurzivni poziv
13     }
14 }
```

VIDEO: REPNA REKURZIJA

Dan Grossman: Tail Recursion (9,44 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK 1

Napraviti dve metode koje rekurzivno rade jedna sa drugom

Napraviti dve metode jednu koja daje Boolean vrednost da li je broj paran dok druga da li je neparan.

Metode treba da zovu jedne drugu, a ona koja je poslednja treba da da rešenje.

Ideja je da se naprave dve funkcije koje kao povratni tip imaju Boolean vrednost.

Jedna proverava da li je broj paran, a druga da li je neparan.

Bazni slučaj može da bude 0 koja je neparan broj.

Dakle, u metodi koja određuje paran broj imaćemo kao bazni slučaj

if(n == 0) return true;

dok ćemo u metodi za neparan broj imati

if(n == 0) return false;

Za ostale slučajeve ćemo pozvati funkciju za broj N-1 i doneti logički suprotan zaključak za N.

Dakle ako posmatramo da li je N parno, pozvaćemo funkciju za N-1 koja iposituje da li je neparna vrednost data kao argument. Ako funkcija vrati da je N-1 neparno, onda je N parno. Važi i suprotno.

ZADATAK 1- REŠENJE

Rešenje zadatka 1

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package z04;

import java.util.Scanner;

/**
 *
 * @author Aleksandra
 */
public class EvenOddMain {

    public static boolean isOdd(int n) {
        if (n < 0) {
            throw new IllegalArgumentException("Nema negativnih brojeva");
        }
        return (n == 0) ? false : isEven(n - 1);
    }

    public static boolean isEven(int n) {
        if (n < 0) {
```

```

        throw new IllegalArgumentException("Nema negativnih brojeva");
    }
    return (n == 0) ? true : isOdd(n - 1);
}

public static void main(String[] args) {
    Scanner stdIn = new Scanner(System.in);
    System.out.print("Unesite broj: ");
    int num = stdIn.nextInt();

    if (isEven(num)) {
        System.out.println(num + " je paran");
    } else {
        System.out.println(num + " je neparan");
    }

    stdIn.close();
}
}

```

ZADATAK 2

Napisati rekurzivnu funkciju `int prebroj(int x)` koja vraća broj bitova postavljenih na 1 u binarnoj reprezentaciji broja `x`.

Napisati rekurzivnu funkciju

`int prebroj(int x)`

koja vraća broj bitova postavljenih na 1 u binarnoj reprezentaciji broja `x`.

Napisati program koji testira napisanu funkciju za broj koji se učitava sa standardnog ulaza.

Ideja je da se uvek proveri bit na najnižoj poziciji (što se dobija upoređivanjem logičkim sa 1)

i ostatak bitske reprezentacije samog broja `N` koji se dobija jednim šiftovanjem u desno, ili deljenjem sa 2.

Ako je bit na najnižoj poziciji 1, onda rezultat rekurzivnog računanja treba uvećati za jedan, u suportnom rezultat rekurzivnog računanja je i rezultat za tu funkciju.

```

public static int prebroj(int x) {
    return (x == 0) ? 0 : ((x & 1) + prebroj(x / 2));
}

```

Bazni slučaj: ako ne `N=0`, bitski zapis je sačinjen samo od nula, te je rezultat 0.

ZADATAK 2 - REŠENJE

Rešenje zadatka koje računa broj bitova postavljenih na 1 u binarnoj reprezentaciji broja x.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package z08;

import java.util.Scanner;

/**
 *
 * @author Aleksandra
 */
public class BitsMain {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int s = input.nextInt();
        int t = prebroj(s);
        System.out.println(t);
        input.close();
    }

    public static int prebroj(int x) {
        return (x == 0) ? 0 : ((x & 1) + prebroj(x / 2));
    }
}
```

ZADACI ZA INDIVIDUALAN RAD

Cilj zadatka za individualni rad jeste provežbavanje stečenog znanja na vežbama i predavanjima

Zadatak 1

Napisati program u kome se korišćenjem rekurzivne funkcije izračunava NZD brojeva x i y koji se unose kroz konzolu.

Zadatak 2

Napisati program koji rekurzivno pretvara decimalan broj u binarni broj.

Zadatak 3

Napisati program koji prima broj a potom sabira sve brojeve tog broja. Npr $2323232 = 2+3+2+3+2+3+2$.

Zadatak 4

Napisati program koji rekurzivno traži najveći element niza.

▼ Poglavlje 9

Primeri upotrebe rekurzije

ZADATAK 1

Cilj zadatka je primena rekurzije kod matematičkih sumiranja

Zadatak 1. Napisati program koji računa zbir prvih N brojeva korišćenjem rekurzivne funkcije. Matematički je suma prvih n brojeva predstavljena jednačinom:

$$\sum_{i=1}^n i = 1 + 2 + 3 + \dots + n$$

Opsti oblik matematičke jednačine koja služi za rešavanje sume prvih N brojeva korišćenjem rekurzije se može napisati na sledeći način:

$$\sum_{i=1}^n i = n + \sum_{i=1}^{n-1}$$

```
public class RecSuma
{
    // Izracunati zbir prvih N celih brojeva - rekurzivno
    public static long sum_rec( int n )
    {
        if( n == 1 )
            return 1;
        else
            return sum_rec( n - 1 ) + n;
    }

    // Izracunati zbir prvih N celih brojeva - iterativno
    public static long sum_iter( int n )
    {
        long sum_iter = 0;
        for(int i = 1; i <= n; i++)
            sum_iter += i;
        return sum_iter;
    }

    // Testiranje programa
    public static void main( String [ ] args )
    {
        for( int i = 1; i <= 10; i++ )
            System.out.println( sum_rec( i ) );
        System.out.println( sum_rec( 8882 ) );
    }
}
```

```

        System.out.println( sum_iter( 8882 ) );
    }
}

```

ZADATAK 2 I 3

Cilj ovih zadataka je primena rekurzije kod matematičkih sumiranja

Zadatak 2 . Napisati rekurzivni algoritam koji određuje zbir sledeće serije brojeva:

$$m(i) = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{i}$$

Napisati test program koji prikazuje sumu $m(i)$ za $i = 1, 2, \dots, 10$.

```

public class Zadatak2 {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++)
            System.out.println(m(i));
    }

    public static double m(int i) {
        if (i == 1)
            return 1;
        else
            return m(i - 1) + 1.0 / i;
    }
}

```

Zadatak 3 Napisati rekurzivni algoritam koji određuje zbir sledeće serije brojeva:

$$m(i) = 1 + \frac{1}{2} + \frac{2}{3} + \dots + \frac{i}{i+1}$$

Napisati test program koji prikazuje sumu $m(i)$ za $i = 1, 2, \dots, 10$.

```

public class Zadatak3 {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++)
            System.out.println(m(i));
    }

    public static double m(int i) {
        if (i == 1)
            return 1.0 / 2;
        else
            return m(i - 1) + i * 1.0 / (i + 1);
    }
}

```

ZADATAK 4

Cilj zadatka je primena rekursije kod brojanja pojavljivanja određenih karaktera u stringu

Zadatak 4. Napisati rekurzivni metod koji određuje broj pojavljivanja odgovarajućeg slova u stringu, pri čemu metod ima zaglavlje:

```
public static int count(String str, char a)
```

Na primer, `count("Welcome", 'e')` vraća rezultat 2. Napisati test program koji od korisnika zahteva da unese string i karakter, a zatim određuje i prikazuje broj pojavljivanja datog karaktera u unetom stringu.

```
import java.util.Scanner;

public class Zadatak4 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String s = input.nextLine();
        System.out.print("Enter a character: ");
        char ch = input.nextLine().charAt(0);
        int times = count(s, ch);
        System.out.println(ch + " appears " + times + (times > 1 ? " times " : " time
") + "in " + s);
    }

    public static int count(String str, char a) {
        int result = 0;
        if (str.length() > 0)
            result = count(str.substring(1), a) +
                ((str.charAt(0) == a) ? 1 : 0);

        return result;
    }
}
```

ZADATAK 5 I 6

Cilj zadatka je primena rekursije kod brojanja velikih slova u stringu i nizu karaktera

Zadatak 5 (*Određivanje broja velikih slova u stringu*) Napisati rekurzivni metod koji kao rezultat vraća broj velikih slova u stringu. Napisati i test program koji od korisnika zahteva da unese string, a zatim ispisuje broj velikih slova stringa.

```
import java.util.Scanner;

public class Zadatak5 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String s = input.nextLine();
        System.out.println("The uppercase letters in " + s + " is " +
countUppercase(s));
    }

    public static int countUppercase(String str) {
        return countUppercase(str, str.length() - 1);
    }

    public static int countUppercase(String str, int high) {
        if (high < 0)
            return 0;
        else
            return countUppercase(str, high - 1) +
                (Character.isUpperCase(str.charAt(high)) ? 1 : 0);
    }
}
```

Zadatak 6 (*Određivanje broja velikih slova u nizu*) Napisati rekurzivni metod koji kao rezultat vraća broj velikih slova u nizu karaktera. Neophodno je da kreirate sledeća dva metoda. Drugi je pomoćni rekurzivni metod

public static int count(**char**[] chars)

public static int count(**char**[] chars, **int** high)

Napisati i test program koji od korisnika zahteva da unese listu karaktera u jednoj liniji, a zatim ispisuje broj velikih slova u listi karaktera.

```
public class Zadatak6 {
    public static void main(String[] args) {
        System.out.println(count(new char[]{'W', 'e', 'l', 'c', 'o', 'm', 'E'}));
    }

    public static int count(char[] chars) {
        return count(chars, chars.length - 1);
    }

    public static int count(char[] chars, int high) {
        if (high >= 0) {
            return count(chars, high - 1) +
                (Character.isUpperCase(chars[high]) ? 1 : 0);
        }
        else
            return 0;
    }
}
```

ZADATAK 7

Napisati rekurzivnu funkciju `int prebroj(int x)` koja vraća broj bitova postavljenih na 1 u binarnoj reprezentaciji broja `x`

Napisati rekurzivnu funkciju `int prebroj(int x)` koja vraća broj bitova postavljenih na 1 u binarnoj reprezentaciji broja `x`.

Napisati program koji testira napisanu funkciju za broj koji se učitava sa standardnog ulaza u heksadekadnom formatu.

```
#include <stdio.h>

int prebroj(int x)
{
    /* Izlaz iz rekurzije */
    if (x == 0)
        return 0;

    if (x && (1 << (sizeof(x) * 8 - 1)))
        return 1 + prebroj(x << 1);
    else
        return prebroj(x << 1);
}

int main()
{
    int x;
    scanf("%x", &x);
    printf("%d\n", prebroj(x));

    return 0;
}
```

ZADACI ZA SAMOSTALNO VEŽBANJE

Na osnovu materijala sa predavanja i vežbi uraditi samostalno sledeće zadatke:

Zadatak 1. Napisati rekurzivni algoritam koji određuje zbir sledeće serije brojeva:

$$m(i) = \frac{1}{3} + \frac{2}{5} + \frac{3}{7} + \frac{4}{9} + \frac{5}{11} + \frac{6}{13} \cdots + \frac{i}{2i+1}$$

Napisati test program koji prikazuje sumu $m(i)$ za $i = 1, 2, \dots, 10$.

Zadatak 2. (Štampanje karaktera stringa u obrnutom poretku) Napisati rekurzivni metod koji prikazuje string u inverznom poretku u konzoli, korišćenjem sledećeg zaglavlja:

```
public static void reverseDisplay(String value)
```

Na primer, `reverseDisplay("abcd")` će prikazati rezultat `dcba`. Napisati test program koji od korisnika zahteva da unese string a zatim ga prikazuje u inverznom poretku.

Zadatak 3. (*Permutacije stringova*) Napisati iterativni i rekurzivni metod koji štampa sve permutacije stringova. Na primer, za string **abc**, biće oštampano:

abc, acb, bac, bca, cab, cba

Zadatak 4. Napisati rekurzivnu funkciju koja ispituje da li je uneti string palindrom. Testirati rad funkcije u glavnom programu.

Zadatak 5. Napisati rekurzivnu funkciju koja štampa sve permutacije brojeva od 1 do 4. Npr, 1234 je jedna permutacija, 2134 druga, itd... Testirati rad funkcije u glavnom programu. Kako bi implementirali funkciju koja radi sa N različitih cifara ($N < 9$)?

▼ Zaključak

REZIME

Glavne pouke lekcije

1. Rekursivni model direktno ili indirektno poziva sam sebe. Da bi se zaustavilo izvršenje rekursivnog modela, mora da postoji jedan ili više osnovnih slučaja.
2. Rekurzija je alternativni oblik upravljanja izvršenjem programa (u odnosu na iterativne načine). Omogućava ponavljanje izvršenja dela programa bez korišćenja programskih petlji. Rekurzija se koristi radi specificiranja jednostavnih, jasnih rešenja za prirodno rekursivne probleme koje bi na drugi način bilo teško rešiti (na primer, problem Hanojskih kula).
3. Po nekad, potrebno je promeniti originalni metod sa pomoćnim parametrima, a da bi se rekursivno pozivao. Rekursivni pomoćni metod se definiše za ove potrebe.
4. Primena rekurzije troši značajne računarske resurse. Uvek kada program poziva rekursivni metod, sistem mora da mu dodeli memoriju za sve njegove lokalne promenljive i parametre. To može da potroši dosta memorije i zahteva dodatno vreme za upravljanje memorijom.
5. Rekursivni metod se naziva repno rekursivnim ako posle njegovog izvršenja ne ostaju nikakve odložene operacije. Neki kompajleri mogu da izvrše optimizaciju repne rekurzije radi smanjivanja veličine memorijskog steka.

