Poskusni test JAVA II

Opomba: Pravi test bo zagotovo krajši, zato boste imeli na voljo dovolj časa.

[5 T / -3T] Imamo naslednji del kode

```
k = 0;
while (k < 22) {
   if (k % 3  == 0) System.out.print(k , " ");
   k = k + 2;
}
```

Kaj izpiše ta del kode

(A) 4 16
(B) 4 10 16
(C) 0 6 12 18
(D) 1 4 7 10 13 16 19
(E) 0 2 4 6 8 10 12 14 16 18
☻ nič od tega

[5 T / -3T] Dana je naslednja rekurzivna metoda

```
public static int cudno(int n) {
   if (n  == 0) return 1;
   return 3 * cudno(n - 1);
}
```

Kakšno vrednost vrne klic `cudno(5)` ?

(A) 0
(B) 3
(C) 81
☻ 243
(E) 6561
(F) Nič od zgoraj navedenega

[15T] Pri ocenjevanju skakalcev je v navadi, da se najboljša in najslabša ocena zavržeta, od preostalih števil pa se izračuna povprečje. Če je več enakih najmanjših (največjih) ocen, zavržemo le eno. V metodi `skakalnoPovprecje`, ki za dano tabelo realnih števil vrnila povprečje njenih elementov, pri čemer naj najmanjšega in največjega števila ne upošteva, manjkajo pogoji. Predpostaviš lahko, da je dolžina tabele večja od 2.

```
public static double skakalnoPovprecje(double[] tabela) {
       int dolzina = tabela.length;
       int i = 1;
       double vsota = tabela[0];
       double minimum = tabela[0]; // kandidat za najmanjse
       double maksimum = tabela[0]; // kandidat za najvecje
       while (i < dolzina) {
              vsota = vsota + tabela[i];
              if (tabela[i] < minimum)
                 minimum = tabela[i]; // novi kandidat za najmanjse
              if (tabela[i] > maksimum)
                 maksimum = tabela[i]; // novi kandidat za najvecje
              i = i + 1;
       }
       // ker je dolzina > 2, ni problemov z deljenjem
```

```
            return (vsota - minimum - maksimum) / (dolzina - 2);
        }
```

[15 T/-10 T]Denimo, da smo dobili naslednji izpis

```
1 1 1 1 1
2 2 2 2
3 3 3
4 4
5
```

Obkroži tiste delčke kode, ki generirajo ta izpis

```
(A) int j = 1;
    while (j <= 5)
    {
      int k = 1;
      while (k <= 5)
      {
         System.out.print(j + " ");
         k = k + 1;
      }
      System.out.println();
      j = j + 1;
    }

(B) int j = 1;
    while (j < 5)
    {
      int k = 1;
      while (k <= 5)
      {
         System.out.print(j + " ");
         k = k + 1;
      }
      System.out.println();
      j = j + 1;
    }

(C) int j = 1;
    while (j <= 5)
    {
      int k = 1;
      while (k <= j)
      {
         System.out.print(j + " ");
         k = k + 1;
      }
      System.out.println();
      j = j + 1;
    }

(D) int j = 1;
    while (j <= 5)
    {
      int k = 1;
      while (k <= j)
      {
         System.out.print(j + "");
         k = k + 1;
      }
      System.out.println();
      j = j + 1;
    }

(E) int j = 1;
    while (j <= 5)
    {
      int k = 5;
      while (k >= 1)
      {
         System.out.print(j + " ");
         k = k - 1;
      }
      System.out.println();
      j = j + 1;
    }

(F) int j = 1;
    while (j <= 5)
    {
      int k = 5;
      while (k >= j)
      {
         System.out.print(j + " ");
         k = k - 1;
      }
      System.out.println();
      j = j + 1;
    }

(G) int j = 1;
    while (j <= 5)
    {
      int k = j;
      while (k <= 5)
      {
         System.out.print(j + " ");
         k = k + 1;
      }
      System.out.println();
      j = j + 1;
    }

(H) System.out.println("1 1 1 1 1");
    System.out.println("2 2 2 2");
    System.out.println("3 3 3");
    System.out.println("4 4");
    System.out.println("5");
```

[15 T] Kaj naredi naslednji program Pregledno označi morebitne presledke!

```
public class Diagram {
   public static void main(String[] kaj) {
      int i, j, k;      // Pomozni stevci.
```

```
        String[] args = {"1", "3", "0", "2", "3", "4", "2"};

        int max = Integer.parseInt(args[0]);
        i = 1;
        while (i < args.length) {
           j = Integer.parseInt(args[i]);
           if (j > max)
              max = Integer.parseInt(args[i]);
           i = i + 1;
        }

        k = 0;
        while (k < max) {
           i = 0;
           while (i < args.length) {
              j = Integer.parseInt(args[i]);
              if (max - k > j) {
                 System.out.print(" ");
              } else {
                 System.out.print("*");
              }
              i = i + 1;
           }
           System.out.println();
           k = k + 1;
        }
     }
  }
}
```

Program izpiše (namesto presledka je ~)

```
~~~~~*~
~*~~**~
~*~****
**~****
```

V splošnem nariše program histogram podatkov.

[9T] Za vsakega od naslednjih opisov zapišite deklaracijo metode (kakšen tip metoda vrača in kakšne argumente sprejme) Metod ni treba napisati!

a) Metoda, ki v podani tabeli celih števil poišče indeks največjega števila in ga vrne.

```
public static int indNaj(int[] tab)
```

b) Metoda, ki na grafični objekt g nariše kvadrat s stranico a in levim zgornjim kotom v točki (x,y).

```
public static void kvadrat(Graphics g, int a, int x, int y)
```

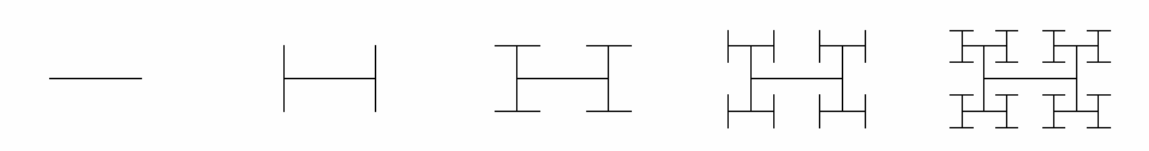
c) Metoda, ki vrne število različnih znakov v danem nizu.

```
public static int razlicnih(String niz)
```

[15T] Dan je razred `LogoZelva` z objektnimi metodami `public void fd(int d)`, `public void bk(int d)`, `public void right(int kot)` in `public void left(int kot)`. Prva premakne želvo naprej za d enot (pri tem se seveda nariše črta dolžine d), druga jo premakne nazaj za d enot (tudi tu nastane črta dolžine d). Smer želve se pri obeh ukazih ne spremeni.

Metoda right spremi smer želve za dani parameter stopinj v desno, metoda left pa obrne smer želve v levo. Pri the dveh metodah se položaj želve ne spremeni, pa tudi ne nariše se nič.

Dopolnite statično metodo `antena(zelva, n, d, f)`, ki kot parametre dobi `LogoZelvo zelva`, naravni števili `n` in `d` ter realno število `f` ter nariše sliko, sestavljeno iz črt, kot je prikazano na spodnjih slikah. Prvi parameter je objekt tipa `LogoZelva`, ki pozna ukaze `fd`, `bk`, `right` in `left` z znanimi pomeni, drugi parameter določa red slike (spodnje slike so redov 0, 1, 2, 3 in 4), tretji dolžino osnovne črte, in četrti razmerje med dolžinami končnih črt na sliki reda `i` in dolžinami končnih črt na sliki reda `i - 1`. Na spodnjih slikah je `f = 0.7`.



```
public static void antena(LogoZelva z, int n, int d, double f)
{ // narise anteno. Po koncu risanja je zelva v prvotnem polozaju
      if (n == 0)
      {
            z.fd(d);
            z.bk(d); // da smo v zacetnem stanju
      }
      else
      {
            z.fd(d);
            z.left(90);
            z.bk((int)(d * f / 2)); // sem "spodaj"
            antena(z, n - 1, (int)(d * f), f);
            z.fd((int)(d * f / 2));
            z.right(90); // sem na koncu osnovne crte;
            z.bk(d);
            z.left(90);
            z.bk((int)(d * f / 2)); // sem "spodaj"
            antena(z, n - 1, (int)(d * f), f);
            z.fd((int)(d * f / 2));
            z.right(90); // sem v zacetnem polozaju
      }
}
```

[5 + 10T] Dana je metoda `naj`:

```
public static int naj(int a, int b)
{
   if (a < b)
   { return a; }
   else
   { return b;}
}
```

Kaj izpiše naslednji del programa

```
int[] tt = {12, 54, 5};
System.out.println("Rezultat metode naj je ", naj(tt[0], tt[1]));
```

Ta del programa izpiše: _____ `Rezultat metode naj je 12` _____

Uporabite to metodo zato, da sestavite metode `najmanjsi`, ki vrne najmanjše od treh (3) celih števil. Namig: V svoji metodi kličite metodo naj z ustreznimi parametri.

```
public static int najmanjsi(int a, int b, int c)
{
   return naj(a, naj(b, c));
}
```

[15T] Napiši program `Vzorec.java`, ki iz ukazne vrstice dobi parameter `n (1 <= n <= 9)` in izpiše `n` vrstic po `n` števil. Ob klicu `Vzorec 8` dobimo:

```
0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1
0 1 2 0 1 2 0 1
0 1 2 3 0 1 2 3
0 1 2 3 4 0 1 2
0 1 2 3 4 5 0 1
0 1 2 3 4 5 6 0
0 1 2 3 4 5 6 7
```

Ideja:

Z metodo `Integer.parseInt(arg[0])` pretvorimo argument iz ukazne vrstice v n. Nato v zanki izpišemo n vrstic.

V vrstici vrstica izpisujemo ostanke pri deljenju številke stolpca – 1 z vrstica.

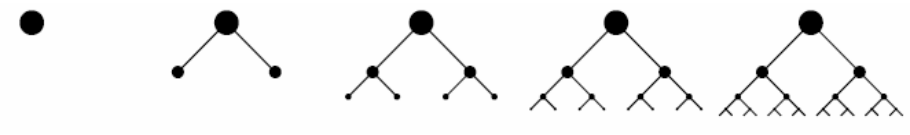
Rešitev:

```
public class ostankiVrstic {
       public static void main(String[] arg) {

              int n = Integer.parseInt(arg[0]);

       int vrstica = 1;
       while (vrstica <= n) {
              // izpisi tekoco vrstico
              int stolpec = 0; // stejmo od 0 dalje
              while (stolpec < n) {
                     // izpisi ustrezen ostanek
                     System.out.print(stolpec % vrstica + " ");
                     stolpec = stolpec + 1;
              }
              // konec vrstice
              System.out.println();
              vrstica = vrstica + 1;
       }
       }
}
```

[25T] Sestavi metodo `narisiDrevo`, ki nariše polno dvojiško drevo dane globine. Na spodnji sliki so prikazana takšna drevesa globin 0, 1, 2, 3 in 4. Uporabi rekurzijo. Predpostavi, da je zgornje vozlišče drevesa vedno na koordinatah (400, 50), začetni polmer krogca 30, prvi vodoravni odmik 200 in navpični odmik 100. Polmer krogca se na vsakem nivoju prepolovi, odmika pa zmanjšata za 30%. Predpostaviš lahko tudi, da globina ni nikoli tako velika, da bi bile težave z risanjem. Navedi, kako pokličeš metodo za risanje drevesa globine 5.



Ideja:

Če dobro pogledamo, je drevo globine n sestavljeno iz kroga, ki je v levo in v desno povezano z drevesom globine n – 1. Poleg globine je drevo določeno še s koordinatama središča kroga. Poznati moramo še odmika na tem nivoju in polmer krogca.

Vsako drevo (globine 0 ali več) je krogec na ustreznem mestu. Če pa rišemo drevo globine več kot 0, pa narišemo še povezovalni črti in levo in desno drevo.

Rešitev:

```
public static void narisiDrevo(Graphics g, int globina,
   int sr_x, int sr_y, int vodOdmik, int navpOdmik, int polmer) {
// v vsakem primeru narišemo krogec
      int lx = sr_x - polmer;
      int ly = sr_y - polmer;
      g.fillOval(lx, ly, 2 * polmer, 2 * polmer);
      if (globina > 0) { // narisemo se levo in desno drevo
         // crta do levega drevesa
         int nx = sr_x - vodOdmik; // sredisce levega drevesa
         int ny = sr_y + navpOdmik;
         // crta do tam
         g.drawLine(sr_x, sr_y, nx, ny);
         // levo drevo
         narisiDrevo(g, globina - 1, nx, ny,
                     vodOdmik/2, navpOdmik/2, polmer * 7 / 10);
         // crta do desnega drevesa
         nx = sr_x + vodOdmik; // sredisce desnega drevesa, y se ne spremni!
         // crta do tam
         g.drawLine(sr_x, sr_y, nx, ny);
         // desno drevo
         narisiDrevo(g, globina - 1, nx, ny,
                     vodOdmik/2, navpOdmik/2, polmer * 7 / 10);
   }
}
```

Klic metode:

```
narisiDrevo(g, 5, 400, 50, 200, 100, 30);
```

[25T] Sestavite metodo `brezPonovitev`, ki iz tabele celih števil, kjer velja `x[i] <= x[i+1]` naredi novo tabelo, kjer brišemo vse večkratne pojavitve elementov, ki se v tabeli pojavijo večkrat, a urejene tako, da velja `x[i] > x[i+1]`.

Na primer, ko pokličemo `brezPonovitev` na tabeli

```
{1, 1, 2, 3, 4, 5, 5, 6, 6, 6, 8, 10, 10}
```

dobimo tabelo

```
{10, 8, 6, 5, 4, 3, 2, 1}
```

Ideja:

Najprej naredimo novo tabelo, ki naj bo velika kot prvotna tabela.

Potrebovali bomo indeks, ki bo povedal kje smo v novi in kje v stari tabeli. Prvi element tabele enostavno prepišemo. Nato pregledamo preostale elemente. Če je tekoči element `stare tabele različen` od zadnjega prepisanega v novo tabelo, ga prepišemo, drugače ne.

Potem naredimo še tabelo za rezultat ustrezno veliko in prepiše elemente vanjo v obratnem vrstnem redu.

Rešitev:

```
public static int[] brezPonovitev(int[] tabela) {
	int dolzina = tabela.length;
	int[] nova = new int[dolzina];
	// prepisemo prvega
	nova[0] = tabela[0];
	// pregledamo preostale
	int kjeNova = 1;
	int kjeStara = 1;
	while (kjeStara < dolzina) {
		if (nova[kjeNova - 1] != tabela[kjeStara]) {
		   nova[kjeNova] = tabela[kjeStara]; // prepišemo
		   kjeNova = kjeNova + 1; // v novi tabeli smo šli naprej
		 }
		kjeStara = kjeStara + 1;
	}
	// prelozili smo kjeNova elementov
	// naredimo prav veliko tabelo
	int[] rezultat = new int[kjeNova];
	int i = 0;
	while (i < kjeNova) {
		rezultat[kjeNova – 1 - i] = nova[i];
		i = i + 1;
	}
	return rezultat;
}
```