Midterm 2019 Meno a priezvisko: _____

obsahuje 5 príkladov, spolu 6+5+5+5+6 = 27 bodov

1) [6 bodov] Prvočíselný

Prvočísel je nekonečne veľa, elegantný Euklidov dôkaz sporom hovorí: Ak by ich bolo konečne veľa, tak nech sú to p_1, p_2, ..., p_n, tak potom ich súčin plus jedna $p_1*p_2* ...*p_n + 1$ nie je deliteľný žiadnym z p_i (lebo zvyšok po delení je 1), ergo, je to prvočíslo, alebo existuje nejaké iné-nespomenuté prvočíslo, ktoré ho delí. A to je spor, a preto je prvočísel NEKONEČNE veľa...

Nič menej, túto Euklidovu historku nepotrebujete k riešeniu príkladu, poslúžila len ako motivácia na príklad.

_____

Vašou úlohou je napísať program, ktorý nájde koľko prvých prvočísel p_i treba vynásobiť, aby ich súčin plus jedna NEBOLO prvočíslo, a teda bolo zložené číslo. K cieľu vás snáď dovedú pomocné úlohy:

Úlohy:

a) **[1 bod]** Definujte metódu `public static boolean isPrime(long n)`, ktorá zistí o čísle n, či je prvočíslo. Predopokladajte, že n > 0, a 1 nie je prvočíslo !

b) **[2 body]** Definujte statickú metódu `public static List<Long> primes(long n)`, ktorá vráti zoznam **prvých n prvočísel**. Predopokladajte, že n > 0.

c) **[2 bod]** Napíšte kód, ktorý nájde najmenšie n také, že $p_1*p_2* ...*p_n + 1$ je zložené, **vypíše n a súčin+1**.

d) **[1 bod]** Mersennove prvočíslo je prvočíslo tvaru 2^p-1, v BiLandii sú to samé jednotky. Ak p nie je prvočíslo, tak 2^p-1, nie je prvočíslo (bez dôkazu, ten si nájdete po midterme). Ak ale p **je** prvočíslo, tak 2^p-1, **nemusí byť prvočíslo**. Napíšte kód, ktorý nájde najmenšie prvočíslo p také, že 2^p-1 je zložené číslo.

Hint: v oboch prípadoch vystačíte s typom **long**. Ak náhodou neviete, ako sa slušne počíta 2^p, tak pozor, Math.pow(double a, double b) vráti a^b, ale je to double.

———— riešenia príkladu 1 píšte na tento list (aj z druhej strany) alebo si vypýtajte ďalší list

2) **[5 bodov] Stromový**

Trieda **`Node`** definuje uzol stromu, ktorý s číslom obsahuje podstromy pre všetky vlastné delitele tohoto čísla `number`. Vlastné delitele n sú kladné čísla menšie ako n, ktoré delia n bez zvyšku. Takže, vlastné delitele 24 sú 1,2,3,4,6,8,12. Uzol s hodnotou 24 má 7 podstromov pre 7 vlastných deliteľov, na poradí v zozname `divisors` nezáleží. Číslo 1 nemá vlastné delitele, preto zoznam podstromov je `divisors` prázdny, obr. dole.

```
public class Node {
   Integer number;
   List<Node> divisors;
   public Node(Integer number, List<Node> divisors) {… toto nemusíte písať }
}
```

Trieda `Node` má automaticky vygenerovaný konštruktor. V triede `Node` ďalej definujte:

Úlohy:

a) **[1 bod]** metódu `public String toString()`, ktorá rekurzívne prevedie celý strom do reťazca, detaily formátovania nie sú rozhodujúce, príklad máte dole,

b) **[1 bod]** konštruktor `public Node(Integer n)`, ktorý rekurzívne vytvorí strom deliteľov pre `n`.

c) **[1 bod]** metódu `int depth()`, ktorá vráti hĺbku stromu. Hĺbka vrchola, ktoré obsahuje 1 je 0.

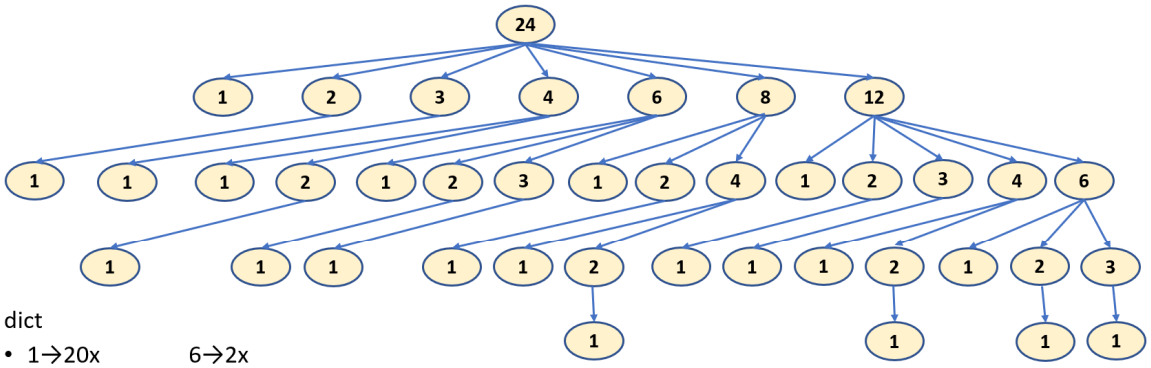
d) **[2 body]** metódu `Map<Integer, Integer> dict()`, ktorá vráti zobrazenie/slovník/frekvenčnú tabuľku, ktorý deliteľ sa v strome koľkokrát nachádza. Na obrázku máte Node(24), v ktorom sa nachádza číslo 1...20x, číslo 2...8x, číslo 3...4x, a t.ď. Toto zobrazenie **musí byť utriedené** podľa kľúča.

Príklady:

```
Node(2)= Node(2,[Node(1,[])])   Node(2).depth()= 1      Node(2).dict()= {1=1, 2=1}

Node(6)= Node(6,[Node(1,[]), Node(2,[Node(1,[])]), Node(3,[Node(1,[])])])
Node(6).depth()= 2      Node(6).dict()= {1=3, 2=1, 3=1, 6=1}

Node(24)= Node(24,[
                Node(1,[]),
                Node(2,[Node(1,[])]),
                Node(3,[Node(1,[])]),
                Node(4,[Node(1,[]), Node(2,[Node(1,[])])]),
                Node(6,[Node(1,[]), Node(2,[Node(1,[])]), Node(3,[Node(1,[])])]),
                Node(8,[Node(1,[]), Node(2,[Node(1,[])]), Node(4,[Node(1,[]), Node(2,[Node(1,[])])])]),
                Node(12,[Node(1,[]),
                         Node(2,[Node(1,[])]),
                         Node(3,[Node(1,[])]),
                         Node(4,[Node(1,[]), Node(2,[Node(1,[])])]),
                         Node(6,[Node(1,[]), Node(2,[Node(1,[])]), Node(3,[Node(1,[])])])])])
Node(24).depth()= 4                                     Node(24).dict()= {1=20, 2=8, 3=4, 4=3, 6=2, 8=1, 12=1, 24=1}
Node(25)= Node(25,[Node(1,[]), Node(5,[Node(1,[])])])   Node(25).depth()= 2     Node(25).dict()= {1=2, 5=1, 25=1}
```



dict

- 1→20x 6→2x
- 2→8x 8→1x
- 3→4x 12→1x
- 4→3x 24→1x

depth→4

———————— riešenia príkladu 2 píšte na tento list alebo si vypýtajte ďalší list

3) [5 bodov]
Úlohy:

a) Tento kód

```
for(int i = 0; i < s.length; i++) {
  for(int j = 0; j < s[i].length; j++) {
    System.out.print(s[i][j]);
} }
```

skončí výpisom reťazca 123456789 a chybou takto

```
123456789Exception NullPointerException
```

Modré je výstup cez System.out.print do out, červené je chyba/výnimka do System.err.

Definujte premennú **s** a **inicializujte** tak, aby k popísanej chybe došlo presne podľa výstupu.

b) Tento kód

```
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
System.out.println(a[0].equals(a[1]));
System.out.println(a[1].equals(a[0]));
```

skončí výpisom:

```
false
true
false
```

a chybou takto:

```
Exception java.lang.NullPointerException
```

Definujte a **inicializujte** premenné **s1, s2, a** tak, aby k popísanej chybe došlo.

c) Tento kód vypíše dva riadky

```
Integer[]   a = {1};
Integer[][] b = {a,a};
Integer[][] c = (Integer[][])b.clone();
b[0][0] = 99;
System.out.printf ("%d %d %d %d\n", a[0], b[1][0], c[0][0], c[1][0]);
System.out.println(                a[0] +b[1][0] +c[0][0] +c[1][0]);
```

Aké budú vypísané riadky ?

d) Definujte dve triedy **Pes** a **Macka** tak, aby ste vedeli vytvoriť dvojprvkové pole s inštanciami tried **Pes** a **Macka**. Ak to ide, vytvorte také pole. Ak to nejde, napíšte **NEEXISTUJE**.

e) Definujte triedu Zajac tak, aby posledný riadok vypísal 1:

```
HashSet<Zajac> pp = new HashSet<>();
pp.add(new Zajac());
pp.add(new Zajac());
System.out.println(pp.size());
```

——— riešenia príkladu 3 píšte na tento list, aj zozadu, alebo si vypýtajte ďalší list

4) [5 bodov] Priatelia

V triede **`Friends`** sú priatelia reprezentovaní v dátovej štruktúre `Map<String, Set<String>> friends`, teda k menu je množina jeho priateľov. Tá môže byť prázdna (`Set.of()`), ale nemôže to byť **`null`**. Zobrazenie **`friends`** je vlastne popis grafu priateľstiev, kde vrcholy grafu sú mená - osoby, a orientované hrany smerujú od osoby k jej priateľom. Priateľstvo je nesymetrická relácia, a sú takí, čo nikoho nemajú... A už aj na obrázku, aj v príklade vidíte, že graf má/môže mať cyklus (po šípkach Palo->Jano->Palo).

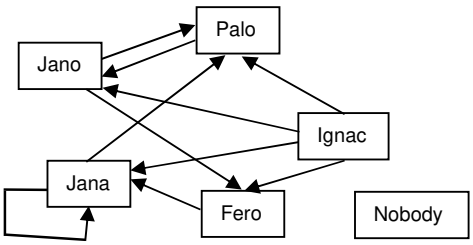
Úlohy:

a) **[2 body]** Definujte metódu `public Set<String> reflexive()`, ktorá vráti množinu samoľúbich priateľov, teda takých, čo sú sami sebe priateľmi.

b) **[3 body]** Definujte metódu `public Set<String> transitive(String s)`, ktorá vráti množinu, do ktorej patrí osoba `s`, jeho priatelia, priatelia priateľov, atd. Teda tranzitívny uzáver priateľov vstupného parametra `s`.

Príklad:

```
friends = Map.of("Jano",   Set.of("Palo", "Fero"),
                 "Palo",   Set.of("Jano"),
                 "Nobody", Set.of(),
                 "Jana",   Set.of("Palo", "Jana"),
                 "Fero",   Set.of("Jana"),
                 "Ignac",  Set.of("Fero", "Palo", "Jano", "Jana"));
Friends fr = new Friends();
fr.reflexive()              [Jana]      jediná Jana má seba vo svojich priateľoch
fr.transitive("Jano")       [Palo, Jana, Jano, Fero]          pozri obrázok
fr.transitive("Ignac")      [Palo, Ignac, Jana, Jano, Fero]
```



———— riešenia príkladu 4 píšte na tento list, aj zozadu, alebo si vypýtajte ďalší list

5) [6 bodov] Streamový

a) [1bod] Nájdite aspoň jedno číslo patriace do tohoto zoznamu:

```
IntStream.range(2,30).filter(n -> IntStream.range(2,n).filter(i->n%i == 0).sum() == 0).boxed()
     .collect(Collectors.toList())
```

V zozname sa nachádza číslo: ...

b) [1bod] Koľko prvkov má výsledná **množina. Pozor, pýtame sa na množinu, nie zoznam** ?

```
Stream.of(1,2,3).flatMap(i -> Stream.of(3,2,1).map(j -> i*j)).collect(Collectors.toSet())
```

Počet prvkov: ...

c) [1bod] Koľko je

- `IntStream.range(0,100).map(i -> i*6).filter(i -> i % 3 == 0).count() == ... ?`
- `IntStream.range(0,100).map(i -> i*3).filter(i -> i % 6 == 0).count() == ... ?`

Bez použitia **for**/**while-cyklu** definujte **usporiadaný** zoznam typu `List<Integer>` prirodzených čísel menších ako `MAX` (použite ako parameter, napr. `int MAX = 1000`), ktoré:

d) [1bod] v desiatkovom zápise končia aspoň dvomi deviatkami, napr. 3999.

e) [1bod] v desiatkovom zápise končia práve dvomi deviatkami, napr. 399, ale nie 3999 ani 19.

f) [1bod] v dvojkovom zápise sú to palindromy, napr. 5, alebo 7.

Hint: akisto viete (a my vás nechceme nachytať), že

- `IntStream.range(0,100)` je stream čísel 0, 1, ..., 99, ale 100 tam nepatrí,
- `.boxed()` je len konverzia z `IntStream` do `Stream<Integer>`
- `.sorted()` utriedi `Stream<Integer>`
- `Integer.toBinaryString` prevedie do dvojkovej, `StringBuffer.reverse` zrkadlovo otočí `StringBuffer`

———————— riešenia príkladu 5 píšte na tento list (aj z druhej strany) alebo si vypýtajte ďalší list