Opravovacia časť

Dve hodiny pred odovzdaním projektu má nemenovaný študent projekt, v ktorom sú doleuvedené triedy. Smutne sa pozerá do Eclipsu, ktorý mu podčiarkol značné množstvo kódu, ktoré je chybné. Popri tom je však v kóde aj niekoľko logických či návrhových chýb.

Vžite sa do úlohy úbohého cvičiaceho, zistite všetky chyby a naznačte spôsob ich opravy. Ak vidíte chyby v návrhu, zdôvodnite ich slovne a navrhnite ideu riešenia. Zároveň vytknite a naznačte opravu logických chýb, aby to nemenovanému študentovi pracovalo tak, ako má.

```
// Názov triedy má byť veľkým písmenom
public class Matica {
	private int[][] matica = new ArrayList<ArrayList<int>>();
	private int[][] matica;

	public matica(int hura, int hura2) {
		matica = new int[hura][hura2];
	}
	public Matica(int početRiadkov, int početStĺpcov) {
		matica = new int[početRiadkov][početStĺpcov];
	}

	Public getPocetRiadkov(matica Matica) {
	public int getPocetRiadkov() {
		return Return matica.length;
	}

	// public s malým „p“, chýba návratový typ, parameter matica je zbytočný
	Public getPocetStlpcou(matica Matica) {
		Return matica[0].length;
	}

	public int pocetStlpcov() {
		return matica[0].length;
	}

	Public int nastavPrvok(int chobotina, chobotina2,
				Prvok prvok)
	{
		Matica[chobotina][chobotina2] = prvok;
	}
}
-----------------------------------------------
public class vektor extends matica {
	public vektor() {
		//vytvori prazdny vektor
		super();
	}
	Public vektor(int rozmer) {
		matica.super(10,0);
	}

	Public getPocetStlpcou(matica Matica) {
		return 0;
	}
	Public getPocetRiadkov() {
		Return matica[0].length
	}
	// odmocnina (x_1*x_1 ... x_n * x_n)
	Public dajNormu() {
		Int norma = 0;
```

Komentár [r1]: deklarujeme pole, ale vytvárame ho ako ArrayList, čo je blbosť

Komentár [r2]: hlúpe názvy premenných

Komentár [r3]: public s malým „p“, chýba návratový typ, parameter *matica* je zbytočný

Komentár [r4]: Return s malým m

Komentár [r5]: Hlúpe názvy parametrov, prečo je **prvok** typu Prvok? **matica** s malým m

Komentár [r6]: Vektor nemá prečo dediť od matice. Vektor je síce „matica, ktorá má len jeden riadok“, ale to narúša Liskovovej substitučný princíp.

Komentár [r7]: U rodiča neexistuje implicitný konštruktor.

Komentár [r8]: Volanie **matica.super** je blud, stačí **super(10, 0)**

Komentár [r9]: Chýba návratový typ metódy

Komentár [r10]: Chýba návratový typ. Metóda robí presne to isté, čo rodičovská metóda.

```
            For(i = 0; i < matica[0].length) {
                  Norma = norma + matica.prvky.get(i);
            }
            Return Math.sqrt(norma);
      }
}
-----------------

public class jednotkovamatica extends matica {
  Public int nastavPrvok(int chobotina, chobotina2, Prvok prvok} {
    if(chobotina == chobotina 2) {
      if(prvok != 1) {
         throw new IllegalArgumentException(
            "Na diagonále musí byť jednotka!");
       }}}}
-----------------
import java.util.List;

public static void main(String[] args) {
       // chcem zratat sucet matic v poli
       List<jednotkovamatica> matice = new List<jednotkovamatica>;
       matice.get(0).nastavPrvok(2, 3, 2, 3);
       matice.get(1).nastavPrvok(2, 3, 2, 3);
       // dalej nestiham, neviem pokracovat
}
```

Komentár [r11]: Chýba návratový typ,; veľké-malé písmená; vzorec je hlúpy

Komentár [r12]: Nie je dôvod na dedičnosť, narúša to Liskovovej SP.

Komentár [r13]: Hlúpe názvy premenných

Komentár [r14]: Prvok má byť zrejme typu **int** v hlavičke metódy.

Komentár [r15]: List je interfejs, nedá sa vytvárať jeho inštancia. Chýbajú zátvorky pri konštruktore.

Komentár [r16]: **get(0)** vráti **null**, žiadna inštancia tam totiž nie je – získame NullPointerException

Ako to spraviť inak?

Vzhľadom na Liskovovej substitučný princíp nie je veľmi dôvod na hierarchiu dedičnosti, hoci je to dôvod na zamyslenie. Nejeden reálny projekt definuje **JednotkovúMaticu** ako potomka **Matice**, ale napr. multiplatformná knižnica pre používateľské rozhranie QT poskytuje len metódu **isIdentity()**, ktorou vieme zistiť, či je inštancia matice jednotková. Samozrejme, tento príklad je nejasný, ale situácia **Vektor extends Matica** je už povážlivejšia. Matematická definícia sa síce môže nazerať na vektor ako na „jednoriadkovú" maticu, ale to je tak všetko. Vektor neponúka žiadne operácie navyše, dramaticky obmedzuje funkcionalitu matice, a teda dedičnosť nemá zmysel.

Mohli by sme teda spraviť dve nezávislé triedy **Matica** a **Vektor** a do matice zaviesť konštruktor **public Matica(Vektor v)**, ktorým by sme vedeli vyrobiť z vektora maticu.

Normu vektora vieme zrátať ako odmocninu zo súčtu druhých mocnín zložiek vektora.

Vytvoriť inštanciu matíc je potom jednoduché:

```
List<Matica> matice = new ArrayList<Matica>();
Matica m1 = new Matica(2, 2);
m1.nastavPrvok(0, 0, 14);
...
Matica m2 = new Matica(2, 2);
m1.nastavPrvok(0, 0, 10);
...
matice.add(m1);
matice.add(m2);
```

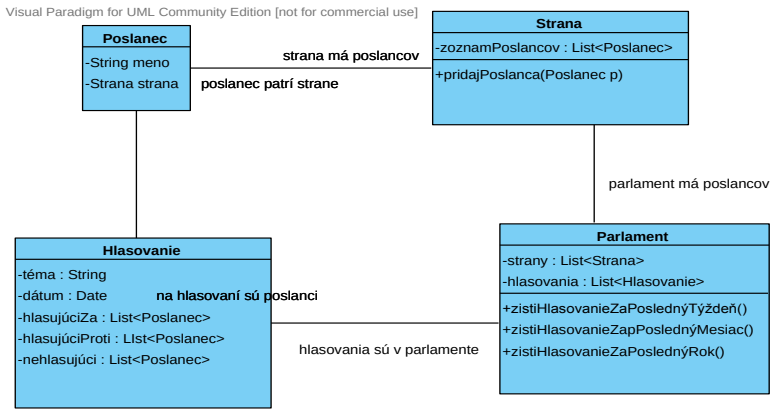
Kreatívna časť

Národná rada nemenovanej republiky sa rozhodla zaviesť informačný systém. Tento parlament pozostáva zo 150 poslancov, ktoré prináležia niektorej zo strán, ale, samozrejme, existujú v ňom aj nezávislí poslanci. V parlamente prebiehajú hlasovania na ľubovoľnú tému, kde je žiaduce evidovať nielen to, koľko poslancov bolo za, proti, či zdržalo sa, ale aj to, ktorý konkrétny poslanec zaujal aký postoj. Národná rada chce mať navyše prehľad, aké hlasovania prebehli za posledný deň, mesiac a rok, a štatistiku o ich priebehoch. Politická strana chce mať zase prehľad o tom, ako hlasovali ich jednotliví poslanci v hlasovaniach (za posledný deň, mesiac, rok).

Navrhnite pre tento informačný systém triedy, uveďte a slovne popíšte ich inštančné premenné a metódy. Zvážte a odôvodnite použitie dedičnosti (ak si myslíte, že dedičnosť je potrebná, slovne to zdôvodnite; ak si myslíte, že potrebná nie je, zdôvodnite to tiež).

Implementujte metódu (t. j. napíšte kód v programovacom jazyku), ktorá pre daného poslanca a dané hlasovanie zistí zoznam všetkých jeho spolustraníkov, ktorí hlasovali rovnako ako on.

Naimplementujte tester (t. j. napíšte kód v programovacom jazyku) s metódou main(), v ktorej ukážete príklady použitia vami navrhnutého informačného systému a tried (predpokladajte, že všetky metódy všetkých tried už máte implementované). V testeri ukážte použitie minimálne 5 metód používaných v triedach informačného systému.



Základný nástrel tried je uvedený na obrázku, pričom jediné metódy, ktoré zisťujú štatistiku, sú uvedené rovno v **Parlamente**.

Tento parlament pozostáva zo 150 poslancov, ktoré prináležia niektorej zo strán, ale, samozrejme, existujú v ňom aj nezávislí poslanci.

150 poslancov by zodpovedalo 150tim inštanciám triedy Poslanec.

Nezávislí poslanci môžu patriť do inštancie triedy **Strany**, ktorá bude mať názov „nezávislí“.

V parlamente prebiehajú hlasovania na ľubovoľnú tému, kde je žiaduce evidovať nielen to, koľko poslancov bolo za, proti, či zdržalo sa, ale aj to, ktorý konkrétny poslanec zaujal aký postoj.

Zavedieme triedu **Hlasovanie** s troma zoznamami, kde budeme evidovať hlasujúcich za, proti a zdržanlivých.

Alternatíva:

Stačí zaviesť dva zoznamy – pre súhlasiacich a pre odporcov. Poslanci, ktorí nehlasovali, sú tí, ktorí sú v parlament a nie sú v žiadnom z týchto dvoch zoznamov.

Alternatíva 2:

Vieme zaviesť vymenovaný typ

```
public enum TypHlasovania { ZA, PROTI, ZDRZIAVA_SA };
```

A v hlasovaní mať mapu

```
private Map<Poslanec, TypHlasovania> stavHlasovania;
```

Následne vieme vyrobiť niekoľko metód, ktoré vedia zistiť hlasovanie konkrétneho poslanca

```
public List<Poslanec> getHlasujúciZa() {
	List<Poslanec> hlasujúciZa = new ArrayList<Poslanec>();
	for(Entry e : stavHlasovania) {
		if(e.getValue() == ZA) {
			hlasujúciZa.add(e.getKey());
		}
	}
}
```

Národná rada chce mať navyše prehľad, aké hlasovania prebehli za posledný deň, mesiac a rok...

Tomu zodpovedajú metódy **hlasovanieZaPoslednýXXX()**.

Alternatíva (pokročilé riešenie):

Vytvoriť triedu **Obdobie**, ktoré bude interne obsahovať počet dní, za ktoré chceme štatistiku

```
public class Obdobie {
	private int početDní;
	public Class(int početDní) {
		this.početDní = početDní;
	}

	public int getPočetDní() {
		return početDní;
	}

	public static Obdobie ROK = new Obdobie(365);
	public static Obdobie MESIAC = new Obdobie(31);
	public static Obdobie TYZDEN = new Obdobie(7);

}
```

Následne stačí jediná metóda

```
public Parlament {
	//…
```

```
        public List<Hlasovanie> zistiHlasovania(Obdobie o) {
              //…
        }
}
```

...a štatistiku o ich priebehoch.

Štatistiku zistíme z inštancií, ktorá vrátia metódy **zistiHlasovaniaZaXXX()**.

Politická strana chce mať zase prehľad o tom, ako hlasovali ich jednotliví poslanci v hlasovaniach (za posledný deň, mesiac, rok).

Tu existujú dve možnosti:

Možnosť 1

Politická strana má metódu **List<Hlasovanie> zistiHlasovaniaPoslancaZaXXX(Poslanec p)**. Problém však spočíva v tom, že politická **Strana** nevie pristupovať k hlasovaniam – to dokáže len **Parlament**. Do **Strany** by sme mohli uviesť inštančnú premennú typu **Parlament**.

Možnosť 2

Politická strana sa bude musieť na to spýtať **Parlamentu**, čiže metódy **List<Hlasovanie> zistiHlasovaniaPoslancaZaXXX(Poslanec p)** vložíme do triedy **Parlament.**

Pokročilá možnosť:

Využijeme triedu **Obdobie** a dvojparametrovú metódu **List<Hlasovanie> zistiHlasovania(Poslanec p, Obdobie o)**.

Zvážte a odôvodnite použitie dedičnosti (ak si myslíte, že dedičnosť je potrebná, slovne to zdôvodnite; ak si myslíte, že potrebná nie je, zdôvodnite to tiež).

Dedičnosť nie je potrebná, žiadne triedy nie sú ani potenciálnymi kandidátmi.

Implementujte metódu (t. j. napíšte kód v programovacom jazyku), ktorá pre daného poslanca a dané hlasovanie zistí zoznam všetkých jeho spolustraníkov, ktorí hlasovali rovnako ako on.

Metódu by sme mohli umiestniť do **Parlamentu**:

```
public Parlament {
   //…
   public List<Poslanec> rovnakoHlasujúciSpolustraníci(Poslanec poslanec,
                                                       Hlasovanie hlasovanie)
   {
       List<Poslanec> rovnakoHlasujúci = new ArrayList<Poslanec>();
       // zisti typ hlasovania
       String typHlasovania = zistiTypHlasovania(poslanec, hlasovanie);
       if("ZA".equals(typHlasovania)) {
           for(Poslanec p : h.getHlasujúciZa()) {
              rovnakoHlasujúci.add(p);
           }
           return rovnakoHlasujúci;
```

```
        }
        if("PROTI".equals(typHlasovania)) {
            for(Poslanec p : h.getHlasujúciProti()) {
               rovnakoHlasujúci.add(p);
            }
            return rovnakoHlasujúci;
        }
        if("ZDRZAL_SA".equals(typHlasovania)) {
            for(Poslanec p : h.getHlasujúciProti()) {
               rovnakoHlasujúci.add(p);
            }
            return rovnakoHlasujúci;
        }
    }
    private String zistiTypHlasovania(Poslanec p, Hlasovanie h) {
        if(hlasovanie.getHlasujúciZa().contains(p)) {
            return "ZA";
        }
        if(hlasovanie.getHlasujúciProti().contains(p)) {
            return "PROTI";
        }
        return "ZDRZAL_SA";
    }
}
```

Samozrejme, tento kód by sa dal vylepšiť: reťazcové konštanty by sa dali nahradiť skutočnými konštantami.

V prípade použitia mapy z poslancov na typy hlasovania by bolo treba kód poupraviť.

Naimplementujte tester (t. j. napíšte kód v programovacom jazyku) s metódou main(), v ktorej ukážete príklady použitia vami navrhnutého informačného systému a tried (predpokladajte, že všetky metódy všetkých tried už máte implementované). V testeri ukážte použitie minimálne 5 metód používaných v triedach informačného systému.

```
public class ParlamentTester {
  public static void main(String[] args) {
    Poslanec p1 = new Poslanec("Milan X");
    Poslanec p2 = new Poslanec("Jozef Y");
    Strana spj = new Strana("Strana priatelov Javy");
    spj.pridajPoslanca(p1);
    p1.setStrana(spj);

    Strana stranaNepriatelov = new Strana("Strana nepriatelov Javy");
    stranaNepriatelov.pridajPoslanca(p2);
    p2.setStrana(stranaNepriatelov);

    Parlament snr = new Parlament();
    snr.add(spj);
    snr.add(stranaNepriatelov);

    Hlasovanie h = new Hlasovanie("O Jave 7");
    h.pridajHlasujucehoZa(p1);
    h.pridajHlasujucehoProti(p2);

    List<Hlasovania> hlasovania = parlament.zistiHlasovaniaZaPoslednyRok();
  }
}
```