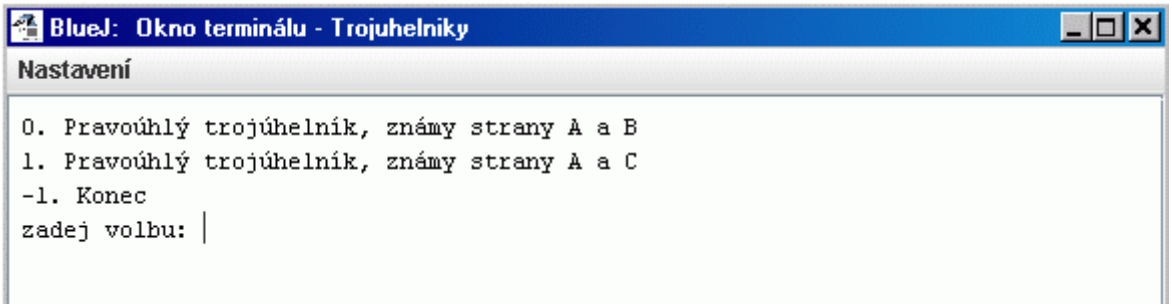
17.Projekt Trojúhelníky

17.1. Základní popis, zadání úkolu

Pracujeme na projektu Trojúhelníky, který je ke stažení na `java.vse.cz`. Aplikace je napsána s textovým uživatelským rozhraním – v BlueJ se vypisuje a načítá vstup z klávesnice v samostatném okně pojmenovaném Terminál. Pro spuštění vytvořte instanci třídy `Trojuhelniky` a zavolejte metodu `zakladniCyklus()`. Výsledek spuštění v BlueJ vidíte na obrázku 17.1.



Obrázek 17.1 Začátek komunikace aplikace Trojúhelníky po spuštění

V tomto projektu budeme řešit tyto úkoly:

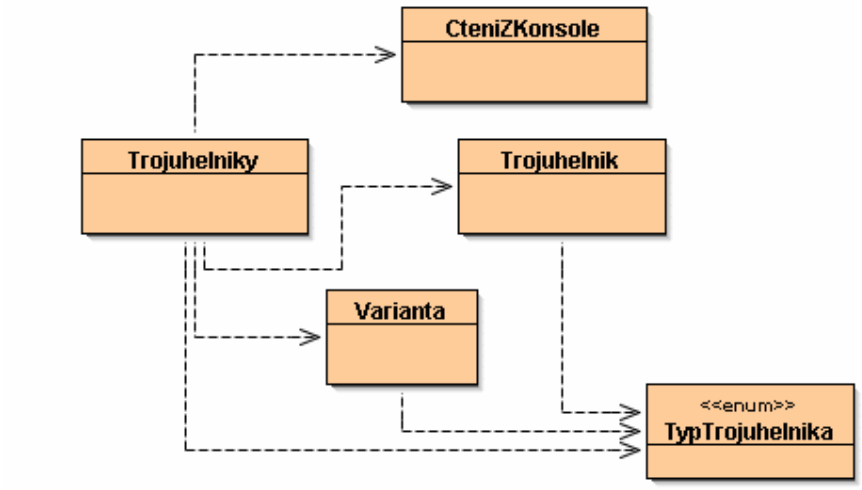
- napsat metodu `main()` pro spuštění aplikace,
- přidat další variantu trojúhelníka, pro kterou bude aplikace umět spočítat parametry trojúhelníka (rovnostranný trojúhelník),
- doplnit detekci chybových stavů a generování výjimek do třídy `Trojuhelnik`,
- ošetřit vzniklé výjimky ve třídě `Trojuhelniky`.

Tento projekt má následující cíle:

- ukázat základní použití výčtového typu v Javě,
- ukázat používání a vytváření statických metod,
- ukázat používání výjimek v Javě.

17.2. Struktura tříd

Projekt Trojúhelníky se skládá z následujících tříd (obrázek z BlueJ):



Obrázek 17.2 Diagram tříd aplikace Trojúhelníky z BlueJ

17.3. Popis komunikace mezi objekty

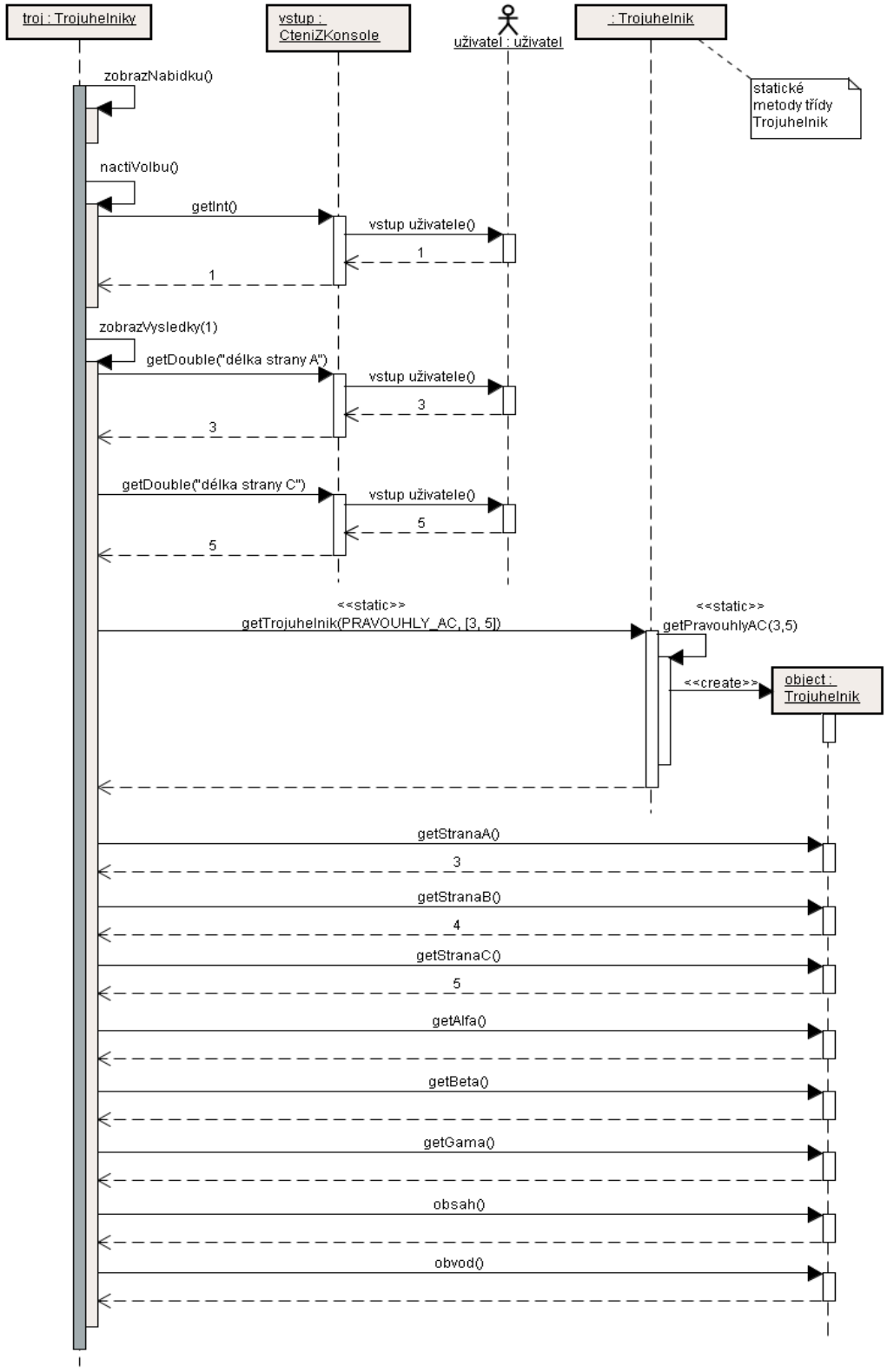
Při vytváření instance třídy `Trojuhelniky` se jednotlivé varianty (instance třídy `Varianta`) vloží do seznamu. Identifikace varianty se provádí pomocí konstanty z výčtového typu `TypTrojuhelnika`. Dále je v konstruktoru vytvořena instance třídy `CteniZKonzole`. Po spuštění metody `zakladniCyklus()` je vypsáno menu a pomocí instance třídy `CteniZKonzole` načtena volba uživatele. Cyklus probíhá dokud není zadána volba `-1` pro konec. Po načtení volby od uživatele se v metodě `zobrazVysledky()` zkontroluje platnost volby. Na základě údajů o variantě jsou pomocí třídy `CteniZKonzole` načteny další údaje o trojúhelníku (délky stran, úhly atd.). Pokud uživatel nezadá číslo, ale např. písmeno A, zopakuje se čtení z konzole. Dostaneme tedy pouze číselné parametry.
Pokračuje se voláním statické metody `getTrojuhelnik()` třídy `Trojuhelnik`, která na základě varianty a parametrů vrátí instanci třídy `Trojuhelnik`. Pokud zadané údaje nepředstavují trojúhelník (např. pokud ve variantě pravoúhlý AC je strana A delší než strana C), je vrácena hodnota `null`. Následně se vypíší informace o trojúhelníku nebo chybové hlášení, pokud trojúhelník nelze vytvořit. Výpis výsledků, když uživatel zadá volbu 0 a délky stran 5 a 8, vidíte na obrázku 17.3.

```
 0. Pravoúhlý trojúhelník, známy strany A a B
 1. Pravoúhlý trojúhelník, známy strany A a C
-1. Konec
zadej volbu: 0
délka strany A: 5
délka strany B: 8
=== parametry trojuhelniku ===
  strany: a=      5,000000  b=      8,000000  c=      9,433981
  uhly:   alfa=  32,005383  beta=  57,994617  gama=  90,000000
  obvod:  22,433981
  obsah:  20,000000

 0. Pravoúhlý trojúhelník, známy strany A a B
 1. Pravoúhlý trojúhelník, známy strany A a C
-1. Konec
zadej volbu:
```

Obrázek 17.3 Ukázka komunikace s aplikací Trojúhelníky

Následující diagram zobrazuje průběh komunikace při dotazu na jeden trojúhelník. Začíná se zobrazením nabídky voleb, končí zobrazením hodnot o trojúhelníku na obrazovce. V diagramu je též zachycen vstup údajů od uživatele.



Obrázek 17.4 Diagram zobrazuje průběh komunikace při dotazu na jeden trojúhelník

17.4. Popis výčtového typu (enum) TypTrojuhelnika

```
 1 /**
 2  * Výčtový typ TypTrojuhelnika představuje jednotlivé typy
 3  * trojúhelníků, se kterými umí pracovat třídy aplikace
 4  * Trojuhelniky.
 5  * @author     Jarmila Pavlíčková
 6  * @version    1.0, duben 2005
 7  */
 8 public enum TypTrojuhelnika {
 9  PRAVOUHLY_A_B, PRAVOUHLY_A_C, ROVNOSTRANNY;
10 }
```

Výčtový typ *`TypTrojuhelnika`* obsahuje konstanty představující jednotlivé typy trojúhelníků, pro které aplikace umí vypočítat jejich strany, úhly, obvod a obsah. V deklaraci výčtového typu je již další předpřipravená konstanta *`ROVNOSTRANNY`* pro typ trojúhelníka, který máme do aplikace přidat. Výčtový typ se používá na dvou místech:

- jako datový atribut třídy *`Varianta`*, která popisuje podrobnosti o příslušné variantě trojúhelníka (kolik je potřeba zadat hodnot, text dotazů na jednotlivé hodnoty),
- jako parametr při vytváření vlastního trojúhelníka – na základě typu se vybere ve statické metodě *`getTrojuhelnik()`* třídy *`Trojuhelnik`* příslušná metoda pro vytvoření instance třídy *`Trojuhelnik`*.

17.5. Popis kódu třídy Trojuhelnik

```
 1 /**
 2  * Třída popisuje trojúhelník.
 3  * Pro získání instance existuje větší množství statických
 4  * metod, které vytvářejí trojúhelník na základě jednotlivých
 5  * známých parametrů.
 6  *
 7  * @author  Luboš Pavlíček
 8  * @version 1.0 srpen 2004
 9  */
10 public class Trojuhelnik{
11  // datové atributy trojúhelníka
12  private double stranaA;
13  private double stranaB;
14  private double stranaC;
15
16  /**
17   * Vytvoření trojúhelníka při znalosti všech tří stran.
18   */
19
20  private Trojuhelnik(double stranaA, double stranaB,
21                                  double stranaC)  {
22  this.stranaA = stranaA;
23  this.stranaB = stranaB;
24  this.stranaC = stranaC;
25  }
26
```

```
27   /**
28    * Statická metoda pro získání instance pravoúhlého
29    * trojúhelníka (tj. úhel gama je 90°)
30    * při zadání velikosti strany A a strany B (tj. obou odvěsen).
31    *
32    * @param  stranaA  délka strany A
33    * @param  stranaB  délka strany B
34    * @return instance třídy Trojuhelnik, která má úhel gama 90°
35    *            a zadanou délku strany A a strany B
36    */
37   public static Trojuhelnik getPravouhlyAB (double stranaA,
38                                             double stranaB) {
39   return new Trojuhelnik(stranaA, stranaB,
40                Math.sqrt(stranaA*stranaA + stranaB*stranaB));
41   }
42
43   /**
44    * Statická metoda pro získání instance pravoúhlého
45    * trojúhelníka (tj. úhel gama je 90°) při zadání
46    * velikosti strany A a strany C (tj. jedné odvěsny a přepony).
47    *
48    * @param  stranaA  délka strany A (odvěsna)
49    * @param  stranaC  délka strany C (přepona)
50    * @return instance třídy Trojuhelnik, která má úhel gama 90°
51    *                 a zadanou délku strany A a strany C
52    */
53   public static Trojuhelnik getPravouhlyAC (double stranaA,
54                                     double stranaC) {
55      if (stranaC <= stranaA) {
56           return null;
57      }
58      else {
59           return new Trojuhelnik(stranaA, Math.sqrt(stranaC
60                     *stranaC - stranaA*stranaA), stranaC);
61      }
62   }
63
64   /**
65    * Obecná statická metoda pro získání instance trojúhelníka,
66    * kdy je zadán typ trojúhelníka (viz enum TypTrojuhelnika)
67    * a pole s potřebnými parametry.
68    *
69    * @param  typ      typ trojúhelníka
70    * @param parametry pole s parametry pro příslušný typ
71    *                  trojúhelníka
72    * @return instance třídy Trojuhelnik příslušného typu
73    *                a odpovídající zadaným parametrům
74    */
```

```
 75  public static Trojuhelnik getTrojuhelnik (TypTrojuhelnika typ,
 76                                  double [] parametry) {
 77    if (typ == TypTrojuhelnika.PRAVOUHLY_A_B) {
 78       if (parametry.length < 2) {
 79            return null;
 80       }
 81       return getPravouhlyAB(parametry[0], parametry[1]);
 82    }
 83    if (typ == TypTrojuhelnika.PRAVOUHLY_A_C) {
 84       if (parametry.length < 2) {
 85            return null;
 86       }
 87       return getPravouhlyAC(parametry[0], parametry[1]);
 88    }
 89    return null;
 90  }
 91
 92  /**
 93   * Vrací délku strany A trojúhelníka
 94   *
 95   * @return  délka strany A trojúhelníka
 96   */
 97  public double getStranaA() {
 98    return stranaA;
 99  }
100
101  /**
102   * Vrací délku strany B trojúhelníka
103   *
104   * @return  délka strany B trojúhelníka
105   */
106  public double getStranaB() {
107    return stranaB;
108  }
109
110  /**
111   * Vrací délku strany C trojúhelníka
112   *
113   * @return  délka strany C trojúhelníka
114   */
115  public double getStranaC() {
116    return stranaC;
117  }
118
119  /**
120   * Vrací velikost úhlu alfa (proti straně A) trojúhelníka
121   * ve stupních
122   *
123   * @return  velikost úhlu alfa trojúhelníka.
124   */
125  public double getAlfa() {
126    return Math.toDegrees(Math.acos( (stranaB*stranaB +
127       stranaC*stranaC - stranaA*stranaA)/(2*stranaB*stranaC)));
128  }
129
130    /**
131     * Vrací velikost úhlu beta (proti straně B) trojúhelníka
132     *  ve stupních
```

```
133     *
134     * @return  velikost úhlu beta trojúhelníka.
135     */
136    public double getBeta() {
137       return Math.toDegrees(Math.acos( (stranaA*stranaA +
138       stranaC*stranaC - stranaB*stranaB)/(2*stranaA*stranaC)));
139    }
140
141    /**
142     * Vrací velikost úhlu gama (proti straně C) trojúhelníka
143     *  ve stupních
144     *
145     * @return  velikost úhlu gama trojúhelníka.
146     */
147    public double getGama() {
148       return Math.toDegrees(Math.acos( (stranaB*stranaB +
149       stranaA*stranaA - stranaC*stranaC)/(2*stranaB*stranaA)));
150    }
151
152    /**
153     * Vypočte obvod trojúhelníka.
154     *
155     * @return obvod trojúhelníka
156     */
157    public double obvod () {
158    return (stranaA + stranaB + stranaC);
159    }
160
161    /**
162     * Vypočte obsah trojúhelníka.
163     *
164     * @return obsah trojúhelníka
165     */
166    public double obsah () {
167       double polObvod=obvod()/2;
168       return Math.sqrt(polObvod*(polObvod-stranaA) *(polObvod-
169                           stranaB) * (polObvod - stranaC));
170    }
171 }
```

Třída `Trojuhelnik` představuje trojúhelník a poskytuje metody pro výpočet jeho parametrů (tj. úhly, obvod a obsah). V konstruktoru třídy předpokládáme, že trojúhelník je vždy zadán třemi stranami. Tento konstruktor je deklarován jako `private` a tudíž ho nelze spustit mimo třídu. Jednotlivé instance trojúhelníků, zadávané podle volby uživatele, nejsou tedy vytvářeny přímo voláním konstruktoru, ale pomocí statické metody `getTrojuhelnik()`, ve které se udává typ trojúhelníka (konstanta z `TypTrojuhelnika`). Toto řešení (privátní konstruktor) je zvoleno z toho důvodu, že přes konstruktory není možné rozlišit jednotlivé typy trojúhelníků, neboť některé varianty se neliší pořadím či počtem parametrů (např. trojúhelník se třemi libovolnými stranami má tři vstupní parametry, trojúhelník se dvěmi stranami a jedním úhlem má také tři vstupní parametry). Metoda `getTrojuhlenik` podle konstanty výčtového typu `TypTrojuhelnika` zavolá odpovídající statickou metodu. Metoda zkontroluje validitu dat (např. v pravoúhlém trojúhelníku nesmí být strana A delší než strana C) a případně dopočítá chybějící strany trojúhelníka, aby mohla spustit konstruktor. Jsou vytvořeny dvě takové metody `getPravouhlyAB()` a `getPravouhlyAC()`, které pomocí Pythagorovy věty dopočítávají třetí stranu trojúhelníka. Ostatní metody ve třídě `Trojuhelnik` jsou již metodami instance a pro vytvořený trojúhelník spočítají úhly, obvod a obsah.

17.6. Popis kódu třídy Trojuhelniky

```
 1  import java.util.ArrayList;
 2  import java.util.List;
 3  /**
 4   * Základní třída aplikace Trouhelniky zobrazuje nabídku.
 5   * Na základě vybrané varianty se zeptá na příslušné parametry
 6   * a poté zobrazí informace o trojúhelníku.
 7   *
 8   * @author  Luboš Pavlíček
 9   * @version 1.0 srpen 2004
10   */
11  public class Trojuhelniky {
12
13    private List<Varianta> varianty;
14    private CteniZKonsole vstup;
15    /**
16     * Vytváří instanci třídy Trojuhelniky
17     */
18    public Trojuhelniky() {
19       varianty = new ArrayList<Varianta>();
20       varianty.add(new Varianta(TypTrojuhelnika.PRAVOUHLY_A_B,
21                     "Pravoúhlý trojúhelník, známy strany A a B",
22                     "délka strany A", "délka strany B", null));
23       varianty.add(new Varianta(TypTrojuhelnika.PRAVOUHLY_A_C,
24                     "Pravoúhlý trojúhelník, známy strany A a C",
25                     "délka strany A", "délka strany C", null));
26       vstup = new CteniZKonsole();
27    }
28
29    /**
30     * Metoda zobrazí nabídku z pole variant a přidá volbu Konec.
31     */
32    private void zobrazNabidku() {
33       for (int i = 0; i< varianty.size(); i++) {
34            Varianta var = varianty.get(i);
35            System.out.printf("%2d. %s%n",i,var.getPopis());
36       }
37       System.out.println("-1. Konec");
38    }
39
40    /**
41     * Metoda pro přístušnou volbu načte potřebné parametry,
42     * vytvoří Trojuhelnik a zobrazí informace o trojúhelniku.
43     *
44     * @param volba  - zvolená varianta z pole varianty
45     */
46    private void zobrazVysledky(int volba) {
47    if ((volba < 0) | (volba >= varianty.size())) {
48       System.out.println("tuto volbu neznam");
49       return;
50    }
51    Varianta var = varianty.get(volba);
52    String dotaz1=var.getDotaz1();
53    String dotaz2=var.getDotaz2();
54    String dotaz3=var.getDotaz3();
55    double [] parametry = new double[var.getPocetParametru()];
```

```
56    if (dotaz1 != null) {
57       parametry[0] = vstup.getDouble(dotaz1);
58    }
59    if (dotaz2 != null) {
60       parametry[1] = vstup.getDouble(dotaz2);
61    }
62    if (dotaz3 != null) {
63       parametry[2] = vstup.getDouble(dotaz3);
64    }
65    Trojuhelnik troj = Trojuhelnik.getTrojuhelnik(var.getTyp(),
66                            parametry);
67    if (troj == null) {
68       System.out.println("!!! metoda getTrojuhelnik nevratila
69                     pro zadane parametry trojuhelnik !!!!");
70       System.out.println("\t typ: " + var.getTyp());
71       for (int i=0; i < parametry.length; i++) {
72            System.out.printf("\tparametry[%d] : %f%n"
73                     i,parametry[i]);
74       }
75    }
76    else {
77       System.out.println("=== parametry trojuhelniku ===");
78       System.out.printf(" strany: a=%14f b=%14f c=%14f%n",
79            troj.getStranaA(),troj.getStranaB(),
80            troj.getStranaC());
81       System.out.printf(" uhly:  alfa=%11f beta=%11f
82            gama=%11f%n",troj.getAlfa(),
83            troj.getBeta(),troj.getGama());
84       System.out.printf(" obvod:  %f%n",troj.obvod());
85       System.out.printf(" obsah:  %f%n%n",troj.obsah());
86    }
87    }
88
89    /**
90     * Metoda přečte volbu uživatele.
91     * Neprovádí se kontrola přípustnosti vstupu.
92     *
93     * @return volba uživatele
94     */
95    private int nactiVolbu() {
96    return vstup.getInt("zadej volbu");
97    }
98
99    /**
100    * Metoda zajišťuje základní cyklus aplikace. Tj.
101    * <UL>
102    *  <LI>zobrazení menu
103    *  <LI>načtení volby od uživatele
104    *  <LI>načtení potřebných parametrů
105    *  <LI>zobrazení informací o trojúhelníku
106    * </UL>
107    * Toto pořadí se opakuje do té doby, než uživatel zvolí
108    *  volbu Konec.
109    */
```

```
110   public void zakladniCyklus() {
111   int volba = 0;
112   zobrazNabidku();
113   volba = nactiVolbu();
114   while (volba != -1) {
115      zobrazVysledky(volba);
116      zobrazNabidku();
117      volba = nactiVolbu();
118   }
119   System.out.println("Konec programu");
120   }
121 }
```

Všimněte si v metodě `zobrazNabidku()` způsobu procházení seznamu pomocí cyklu `for` s řídící proměnnou cyklu – tato varianta je zvolena, aby bylo možné vypsat i pořadové číslo nabídky.
U některých výstupů je použito formátování řetězců v metodě `printf` – viz popis třídy `String` v kapitole 5.4.

17.7. Postup řešení

17.7.1. Vytvoření metody main, pro spouštění aplikace z příkazové řádky

Jak bylo uvedeno v úvodu, při spuštění projektu vytvoříme instanci třídy `Trojuhelniky` a zavoláme metodu `zakladniCyklus()`. Do metody `main` ve třídě `Trojuhelniky` tento postup zapíšeme v kódu:

```
1   /**
2    * Metoda main pro spuštění aplikace
3    *
4    * @param args pole argumentů vstupní řádky – nezpracovávají se
5    */
6   public static void main (String [] args) {
7   Trojuhelniky troj = new Trojuhelniky();
8   troj.zakladniCyklus();
9   }
```

17.7.2. Přidání nového typu trojúhelníka (rovnostranného)

Typ `ROVNOSTRANNY` je již ve výčtovém typu `TypTrojuhelnika` uveden. Musíme provést změny pouze ve třídách `Trojuhelnik` a `Trojuhelniky`.
Ve třídě `Trojuhelniky` doplníme do seznamu variant (datový atribut varianty typu `List`[37] na řádku 13 výpisu třídy `Trojuhelniky`) novou variantu s těmito parametry:

- prvním parametrem je typ trojúhelníka – použijeme konstantu `ROVNOSTRANNY` z výčtového typu `TypTrojuhelnika`,
- druhým parametrem je text zobrazovaný v nabídce – vypíše se „Rovnostranný trojúhelník“,
- třetím až pátým parametrem jsou texty dotazů na hodnoty vkládané uživatelem – první dotaz bude na délku strany („délka strany“), více dotazů není potřeba, proto za čtvrtý a pátý parametr použijeme konstantu `null`.

Následující kód přidáme do třídy `Trojuhelniky` za řádek číslo 25.

```
varianty.add(new Varianta(TypTrojuhelnika.ROVNOSTRANNY,
     "Rovnostranný trojúhelník", "délka strany",null,null));
```

[37] Datový atribut varianty je typu `List`, konkrétní instance je typu `ArrayList`, viz volání konstruktoru na řádku 19 zdrojového kódu třídy `Trojuhelnik`.

Další úpravy se budou týkat kódu třídy *Trojuhelnik*. Musíme napsat statickou metodu *getRovnostranny()*, která na základě délky strany (parametru) vytvoří instanci trojúhelníka. Kód této metody je uveden v následujícím výpisu:

```
 1 /**
 2  * Statická metoda pro získání instance rovnostranného
 3  *trojúhelníka při zadání velikosti strany.
 4  *
 5  * @param  strana  délka strany
 6  * @return instance třídy Trojuhelnik, která má všechny úhly 60°
 7  * a zadanou délku strany
 8  */
 9 public static Trojuhelnik getRovnostranny (double strana) {
10  return new Trojuhelnik(strana, strana, strana);
11 }
```

Dále upravíme kód metody *getTrojuhelnik()*, která dle typu trojúhelníka rozhodne, kterou konkrétní metodu pro vytvoření trojúhelníka zavolá. Do kódu této metody připíšeme rozhodování pro typ *ROVNOSTRANNY* – spustí se metoda *getRovnostranny()*. Následující kód se umístí do metody *getTrojuhelnik()* za řádek 88 ve výpisu kódu třídy.

```
if (typ == TypTrojuhelnika.ROVNOSTRANNY){
     if (parametry.length < 1){
          return null;
     }
     return getRovnostranny(parametry[0]);
}
```

17.7.3. Zjišťování chyb ve třídě Trojuhelnik, generování výjimek

Aplikace již funguje téměř dobře, chybí ale detekce některých chybových stavů. Na následujícím obrázku je vidět situace, že lze zadat trojúhelník se zápornou délkou strany.

```
délka strany A: 3
délka strany B: -1
=== parametry trojuhelniku ===
  strany: a=      3,000000  b=     -1,000000  c=      3,162278
  uhly:   alfa= 108,434949  beta=  18,434949  gama=  90,000000
  obvod:  5,162278
  obsah:  1,500000
```

Obrázek 17.5 Výpis aplikace při zadání záporné délky strany

Některé chybové stavy jsou již detekovány, obecnost chybového hlášení příliš nepomůže s určením důvodu problému. Chyba se projeví např. při zadání stejně dlouhé strany *a* a *c* u pravoúhlého trojúhelníka.

```
délka strany A: 3
délka strany C: 3
!!! metoda getTrojuhelnik nevratila pro zadane parametry trojuhelnik !!!!
         typ: PRAVOUHLY_A_C
         parametry[0] : 3,000000
         parametry[1] : 3,000000
```

Obrázek 17.6 Chybové hlášení při zadání stejných délek stran A a C

Pro detekci chybových stavů se většinou používá příkaz `if` s příslušnými podmínkami. Je potřeba však ještě vyřešit způsob, jak předat informaci o chybě včetně vhodného popisu z místa detekce na místo, kde se může chyba ošetřit (např. vypsat hlášení uživateli). Někdy máme štěstí, detekce a ošetření chyby je na jednom místě. Častější je, že se jedná o různá místa v kódu aplikace, často i o různé třídy. Ve většině současných jazyků lze použít dva mechanismy:

- Přes návratovou hodnotu metody – tento způsob je již nyní použit např. při kontrole, zda strana `c` je delší než strana `a`, podívejte se na řádky 55 až 57 ve třídě `Trojuhelnik`. V případě chybového stavu se vrací hodnota `null`. Nevýhodou tohoto mechanismu jsou komplikace při návrhu a používání metod a omezené možnosti informování o chybových stavech. Tento mechanismus též nelze použít v konstruktorech (konstruktory nemají návratovou hodnotu).
- Pomocí výjimek, kdy někam umístíme kód ošetřující chybu (blok `try catch`) a jinam umístíme kód oznamující, že vznikla chyba (vyvolání výjimky pomocí `throw`). Nevýhodou výjimek je jejich pomalost, výhodou jednodušší návrh metod, přehlednější kód.

My budeme používat mechanismus výjimek – při zjištění chybného vstupního parametru (např. záporná délka strany) vygenerujeme výjimku `IllegalArgumentException`. Tato standardní výjimka svým názvem dobře vystihuje charakter chyby, je zbytečné vytvářet vlastní výjimky. Ošetření chybových stavů – výpis vhodného chybového hlášení umístíme do třídy `Trojuhelniky`, neboť je vhodné operace pro komunikaci s uživatelem soustředit do jedné třídy (je to přehlednější, jednoznačně jsou rozděleny funkce mezi třídy, usnadní to v budoucnu výměnu textového rozhraní za grafické).
V konstruktoru třídy `Trojuhelnik` budeme detekovat dva chybové stavy – délka některé strany je menší nebo rovna nule a součet libovolných stran je menší nebo roven straně třetí:

```
/**
 * Vytvoření trojúhelníka při znalosti všech tří stran.
 *  @exception IllegalArgumentException  - pokud je některá
 *   strana <= 0 nebo pokud je součet libovolných dvou stran menší .*   či
roven straně třetí
 */
private Trojuhelnik(double stranaA, double stranaB,
              double stranaC) throws IllegalArgumentException {
    if ((stranaA <=0) | ( stranaB <= 0) | (stranaC <= 0)) {
         throw new IllegalArgumentException(
                        "strana trojúhelníka musí být větší než 0");
    }
    if ((stranaA + stranaB <= stranaC) |
                  (stranaA + stranaC <= stranaB) |
                  (stranaB + stranaC <= stranaC)) {
         throw new IllegalArgumentException("součet dvou stran "+
                  "trojúhelníka musí být větší než strana třetí");
    }
    this.stranaA = stranaA;
    this.stranaB = stranaB;
    this.stranaC = stranaC;
}
```

Všimněte si, že popis výjimky je uveden i v dokumentačních komentářích. V hlavičce metody nemusí být v našem případě uvedeno `throws IllegalArgumentException`, neboť tato výjimka nepatří mezi povinně odchytávané, je však vhodnější ji uvádět, neboť čtenář kódu si možnosti vzniku výjimky všimne hned na začátku metody.
Metodu `getPravouhlyAB()` nebudeme doplňovat o detekci chybových stavů – všechny v úvahu přicházející chybové stavy budou detekovány v konstruktoru třídy `Trojuhelnik`. Je však opět vhodné doplnit komentáře k metodě o možnost vzniku výjimky, podobně i hlavičku metody o `throws`.
V metodě `getPravouhlyAC()` nahradíme vracení hodnoty `null` generováním výjimky:

```
public static Trojuhelnik getPravouhlyAC (double stranaA,
                  double stranaC) throws IllegalArgumentException {
     if (stranaC <= stranaA) {
          throw new IllegalArgumentException("u pravoúhlého " +
               "trojúhelníka musí být strana C větší než strana A");
     }
     else {
          return new Trojuhelnik(stranaA,
               Math.sqrt(stranaC*stranaC - stranaA*stranaA), stranaC);
     }
}
```

Obdobně v metodě *getTrojuhelnik()* nahradíme vracení hodnoty *null* generováním výjimky:

```
     throw new IllegalArgumentException(
                  "nedostatečný počet parametrů v poli");
```

17.7.4. Ošetřování výjimek ve třídě Trojuhelniky

Používání výjimek má jednu výhodu – pokud je programátor neošetří, použije se standardní mechanismus ošetření – vypíší se informace o chybě a aplikace se ukončí. Pokud nyní zadáme zápornou délku strany, tak se zobrazí na konzole následující výpis:

```
java.lang.IllegalArgumentException: strana trojúhelníka musí být větší než 0
        at Trojuhelnik.<init>(Trojuhelnik.java:25)
        at Trojuhelnik.getPravouhlyAB(Trojuhelnik.java:46)
        at Trojuhelnik.getTrojuhelnik(Trojuhelnik.java:97)
        at Trojuhelniky.zobrazVysledky(Trojuhelniky.java:62)
        at Trojuhelniky.zakladniCyklus(Trojuhelniky.java:103)
        at Trojuhelniky.main(Trojuhelniky.java:112)
```

Obrázek 17.7 Výpis aplikace při zadání záporné délky strany

Z tohoto popisu chyby bude programátor většinou nadšen – ve výpisu vidí typ a popis chyby i kde v kódu ji má hledat – výjimka *IllegalArgumentException* vznikla na řádku 25 v konstruktoru třídy *Trojuhelnik* a konstruktor byl volán z metody *getPravouhlyAB()* na řádku 46. Tato metoda byla volána z metody *getTrojuhelnik()*, ta byla volána z metody *zobrazVysledky()* ve třídě *Trojuhelniky*, atd. Spokojenost uživatele však bude minimální. Výjimky budeme odchytávat a ošetřovat ve druhé části metody *zobrazVysledky()*. Umístíme sem konstrukci *try catch*, v bloku *try* bude volání metody *getTrojuhelnik()* a výpis parametrů trojúhelníka při správně zadaných parametrech. Blok *catch* bude pouze jeden, v něm budeme odchytávat výjimku *IllegalArgumentException*. Zprávu předávanou přes výjimku získáme pomocí metody *getMessage()*.

```
private void zobrazVysledky(int volba) {
     if ((volba < 0) | (volba >= varianty.size())) {
          System.out.println("tuto volbu neznam");
          return;
     }
     Varianta var = varianty.get(volba);
     String dotaz1=var.getDotaz1();
     String dotaz2=var.getDotaz2();
     String dotaz3=var.getDotaz3();
     double [] parametry = new double[var.getPocetParametru()];
```

```
    if (dotaz1 != null) {
         parametry[0] = vstup.getDouble(dotaz1);
    }
    if (dotaz2 != null) {
         parametry[1] = vstup.getDouble(dotaz2);
    }
    if (dotaz3 != null) {
         parametry[2] = vstup.getDouble(dotaz3);
    }
    try {
         Trojuhelnik troj = Trojuhelnik.getTrojuhelnik(var.getTyp(),
                                                       parametry);
         System.out.println("=== parametry trojuhelniku ===");
         System.out.printf(" strany: a=%14f  b=%14f  c=%14f%n",
              troj.getStranaA(),troj.getStranaB(),troj.getStranaC());
         System.out.printf(" uhly:  alfa=%11f beta=%11f gama=%11f%n",
              troj.getAlfa(),troj.getBeta(),troj.getGama());
         System.out.printf("  obvod:  %f%n",troj.obvod());
         System.out.printf("  obsah:  %f%n%n",troj.obvod());
    }
    catch (IllegalArgumentException exc) {
         System.out.println("!!! chybné vstupní parametry: "+
                   exc.getMessage());
         System.out.println("\t typ: " + var.getTyp());
         for (int i=0; i < parametry.length; i++) {
              System.out.printf("\t parametry[%d] : %f%n", i,
                            parametry[i]);
         }
    }
}
```

17.8. Domácí úkoly

1. Doplňte aplikaci o další typy trojúhelníků:
   - obecný trojúhelník, zadány tři strany,
   - obecný trojúhelník, zadány dvě strany a úhel,
   - obecný trojúhelník, zadány dva úhly a strana.

   U všech typů doplňte i kontrolu, zda zadané údaje opravdu představují trojúhelník (libovolný úhel i součet dvou úhlů musí být menší než 180 stupňů, součet dvou stran trojúhelníka musí být vždy větší než třetí strana atd.).

2. Upravte aplikaci, aby se varianty při výpisu číslovaly od jedničky a ne od nuly.

3. Upravte předchozí projekt Kalkulačka tak, aby používal pro uložení operace výčtový typ (enum).

4. Nastudujte v dokumentaci Javy možnosti rozšiřování výčtového typu enum a zkuste sloučit třídu `Varianta` a výčtový typ `TypTrojuhelnika` do jednoho výčtového typu.