1. Projekt Kalkulačka

1.1. Základní popis, zadání úkolu

Pracujeme na projektu Kalkulačka, který je ke stažení na `java.vse.cz`. Po otevření v BlueJ vytvoříme instanci třídy `Kalkulacka` a zavoláme metodu `show()`. Výsledkem je spuštění grafické aplikace představující jednoduchou kalkulačku viz Obrázek 1.1.

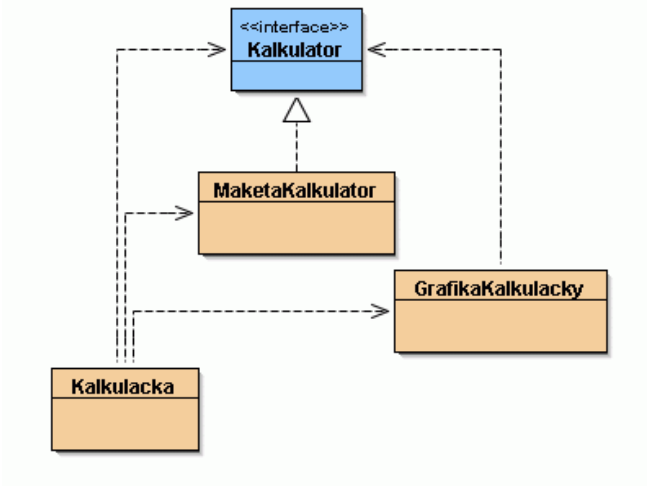
Obrázek 1.1 Vzhled kalkulačky po spuštění

Naším úkolem bude „naučit kalkulačku počítat“, zatím neumí nic. Abychom mohli kontrolovat, že naše implementace kalkulačky funguje správně, připravíme si sadu jednotkových testů. Vytvoříme tedy základ třídy implementující rozhraní kalkulátor, vytvoříme testy a pak dokončíme implementaci třídy obsahující logiku kalkulačky tak, aby všechny testy skončily úspěšně. Pro vytváření a spouštění testů použijeme testovací framework JUnit, který je zakomponován i v BlueJ.
Tento projekt má následující cíle:

- ukázat jednu z možností oddělení grafického rozhraní od věcné třídy,
- vysvětlit jak třída implementuje rozhraní,
- naučit se vytvářet jednotkové testy pomocí JUnit,
- navrhnout datové atributy pro třídu implementující rozhraní Kalkulator,
- procvičit základní operace s primitivními datovými typy (čísla, znaky, logická hodnota),
- naučit se příkaz `if` a vytváření podmínek.

1.2. Struktura tříd

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



Obrázek 1.2 Diagram tříd projektu Kalkulačka, převzato z BlueJ

Třída *Kalkulacka* slouží ke spuštění aplikace, propojuje instanci třídy *GrafikaKalkulacky* s implementací rozhraní *Kalkulator*. Všimněte si, že v deklaraci datového atributu kalk je uveden typ *Kalkulator* tedy rozhraní. Teprve při inicializaci je vytvořena instance konkrétní třídy, která toto rozhraní implementuje, v našem příkladě je to na začátku třída *MaketaKalkulacky*.

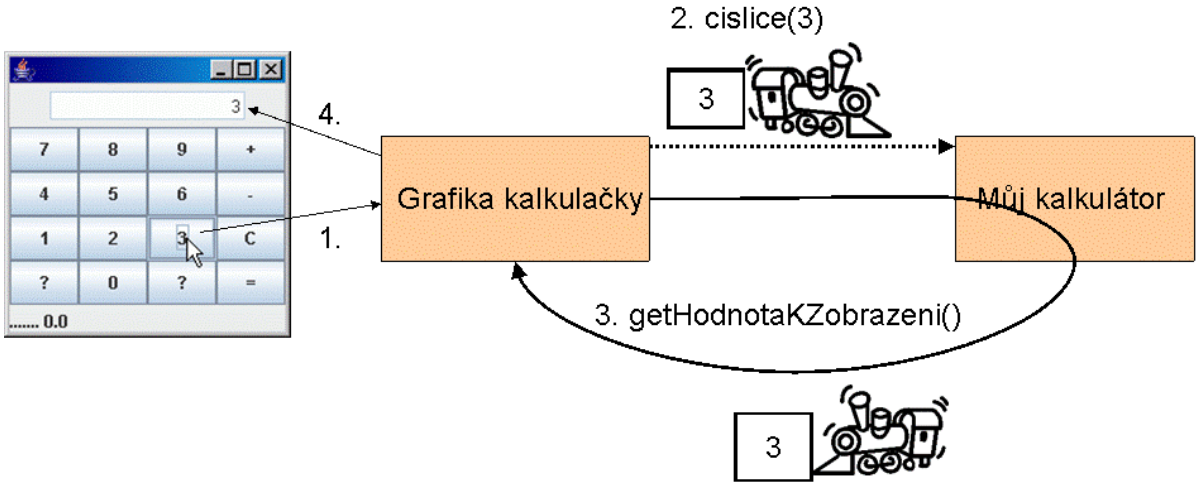
```
// datove atributy instanci

Kalkulator kalk;
GrafikaKalkulacky gui;

/**
 * Konstruktor pro vytvoreni instance tridy Kalkulacka
 */

public Kalkulacka() {
//Inicializujte atributy instance

     kalk = new MaketaKalkulator();
     gui = new GrafikaKalkulacky(kalk);
}
```

Třída *MaketaKalkulacky*, jak její název napovídá, neslouží jako plnohodnotná implementace rozhraní *Kalkulator*. Představuje pouze maketu (zástupný objekt, v angličtině se používá termín mock object), která slouží k otestování toho, zda je správně naimplementována grafika aplikace. Proto se také zároveň s grafickým uživatelským rozhraním kalkulačky otevře okno textového výstupu, do kterého maketa kalkulátoru zapisuje jaké metody byly volány. Maketa kalkulátoru neosahuje žádnou inteligenci, proto se na displeji kalkulačky zobrazuje neustále stejná hodnota (12345).
Třída *GrafikaKalkulacky* má na starosti zobrazování grafického uživatelského rozhraní a podle toho, jaké tlačítko uživatel stiskne, volá metody z rozhraní *Kalkulator*.
Rozhraní *Kalkulator* obsahuje deklaraci metod, které musí být naimplementovány pro správné fungování této kalkulačky.

1.3. Popis komunikace mezi objekty

Nyní si popíšeme, jak by vypadala komunikace mezi objekty, kdyby jsme měli správně naimplementovanou funkčnost kalkulačky.
V konstruktoru třídy *Kalkulacka* se vytváří instance implementace třídy *Kalkulator* a instance třídy *GrafikaKalkulacky*, která jako parametr při vzniku získá odkaz na vytvořenou instanci třídy *Kalkulator*. V metodě *show()* se zobrazí grafické rozhraní kalkulačky a předá se mu řízení komunikace.
Komunikace mezi uživatelem a instancemi tříd *GrafikaKalkulacky* a implementace rozhraní *Kalkulator* probíhá v této posloupnosti:

1. Uživatel stiskne některou klávesu (na obrázku je to stisknutí tlačítka s hodnotou 3).
2. Instance třídy *GrafikaKalkulacky* vyhodnotí, které tlačítko to bylo a zavolá odpovídající metodu ze třídy implementující rozhraní *Kalkulator*. Na našem obrázku zavolá metodu *cislice()* a jako parametr jí předá číslo 3, které odpovídá stisknuté klávese. Tato metoda zpracuje poslaný údaj.
3. Instance třídy *GrafikaKalkulacky* zavolá metodu *getHodnotaKZobrazeni()* instance třídy implementující rozhraní *Kalkulator*. Tato metoda vrátí hodnotu, která má být zobrazena na displeji.
4. Instance třídy *GrafikaKalkulacky* zobrazí na displeji obdrženou hodnotu.



Obrázek 1.3 Znázornění komunikace mezi jednotlivými instancemi při stisknutí číslice 3

1.4. Popis kódu rozhraní Kalkulator

```
 1 /**
 2  *
 3  * Toto rozhraní definuje metody kalkulátoru.
 4  * Rozhraní má tři skupiny metod:
 5  * a) pomocí metody getHodnotaKZobrazeni grafická třída zjišťuje
 6  *    co má zobrazit na displeji,
 7  * b) metody cislice, plus, minus, rovnaSe a vymaz se volají
 8  *    při stisknutí příslušné klávesy na kalkulačce,
 9  * c) metody getAutor a getVerze jsou informační
10  *
11  * Úkolem studentů je vytvořit třídu,
12  *  která bude implementovat toto rozhraní.
13  *
14  * @author     Lubos Pavlicek
15  * @version    2.0 (13 June 2006)
16  */
17 public interface Kalkulator
18 {
19     /**
20      * Metoda vrací hodnotu, která se má zobrazit
21      * na displeji kalkulacky.
22      * Tato metoda se volá po zavolání metody odpovídající
23      * stisku tlačítka.
24      *
25      * @return   hodnota k zobrazení
26      */
27     public int getHodnotaKZobrazeni();
28
29     /**
30      * metoda se volá při stisknutí tlačítka s číslicí na kalkulačce.
31      * Parametrem je hodnota na stisknuté klávese.
32      *
33      * @param hodnota    hodnota na stisknutém tlačítku,
34      *                   hodnota je v rozsahu od 0 do 9
35      */
36     public void cislice(int hodnota);
37
```

```
38      /**
39       * metoda se volá při stisknutí tlačítka "+" (plus) na kalkulačce
40       */
41      public void plus();
42
43      /**
44       * metoda se volá při stisknutí tlačítka "-" (minus) na kalkulačce
45       */
46      public void minus();
47
48      /**
49       * metoda se volá při stisknutí tlačítka "=" (rovná se)
50       * na kalkulačce
51       */
52      public void rovnaSe();
53
54      /**
55       * metoda se volá při stisknutí tlačítka "C" (clear) na kalkulačce
56       */
57      public void vymaz();
58
59      /**
60       * metoda vrací jméno autora, např. "autor: Jan Novák"
61       *
62       * @return   řetězec se jménem autora
63       */
64      public String getAutor();
65
66      /**
67       * metoda vrací označení verze, např. "verze 1.0.3"
68       *
69       * @return   řetězec s verzí programu
70       */
71      public String getVerze();
72  }
```

1.5. Postup řešení:

Řešení si rozdělíme do několika kroků:

- vytvoření základu implementace rozhraní Kalkulator,
- vytvoření sady testů pomocí JUnit,
- doplnění autora a verze,
- vkládání čísla (číselné klávesy),
- výmaz čísla (klávesa C),
- sčítání dvou čísel (klávesy + a =),
- sčítání více čísel,
- odčítání.

1.5.1. Vytvoření základu implementace rozhraní Kalkulator

Abychom mohli začít psát testy, potřebujeme mít základ třídy, kterou chceme testovat. Musíme znát rozhraní této třídy (zde ve smyslu, jaké metody třída poskytuje), funkčnost ještě nebude naimplementována, ale kód musí jít přeložit. Vytvoříme tedy třídu, která se bude jmenovat `MujKalkulator` a bude implementovat rozhraní `Kalkulator`. Později instancí třídy `MujKalkulator` nahradíme instanci třídy `MaketaKalkulatoru` v konstruktoru třídy `Kalkulacka`.

Jaký bude konkrétní postup. Vytvoříme novou třídu, která bude mít následující hlavičku:

```
public class MujKalkulator implements Kalkulator
```

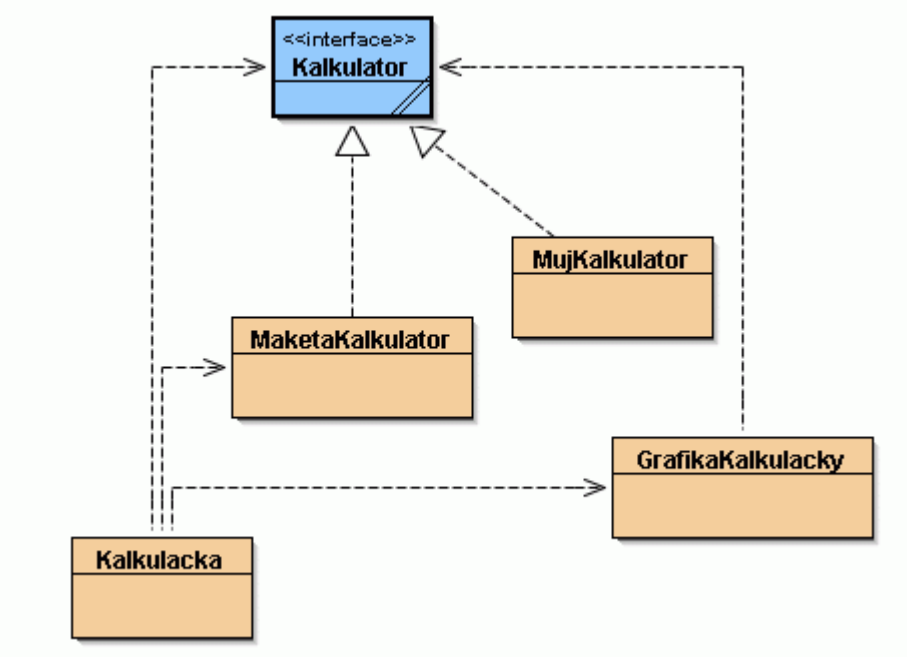
Dovnitř do kódu této třídy nakopírujte všechny metody z rozhraní `Kalkulator`, včetně komentářů. Poté musíme jednotlivé metody naimplementovat tak, aby kód bylo možno přeložit. Pokud metoda nevrací hodnotu její prázdná implementace je jednoduchá, tvoří ji pouze počáteční a koncová závorka. Těchto metod je ve třídě `MujKalkulator` několik, např. prázdná implementace metody `plus()` vypadá takto:

```
public void plus(){
}
```

Metody, které vracejí hodnotu, musejí, aby šly přeložit, obsahovat klíčové slovo return a za ním hodnotu odpovídajícího typu. Doplňují se vždy implicitní hodnoty pro daný typ, tj. pro čísla 0, pro znak prázdný znak, pro booleanovskou hodnotu false a pro referenční typy hodnota null. Pokud je návratová hodnota metody typu `String` , je možné uvést také prázdný String tj.““. Prázdná implementace metod třídy `MujKalkulator`, které vracejí hodnotu, je uvedena v následujícím výpise.

```
public int getHodnotaKZobrazeni(){
   return 0;
}
//část kódu vynechána

public String getAutor(){
    return null;
}

public String getVerze(){
     return "";
}
```

Všimněte si, že i v diagramu tříd v BlueJ je znázorněno, že třída `MujKalkulator` implementuje rozhraní `Kalkulator`.



Obrázek 1.4 Diagram tříd projektu Kalkulačka po vytvoření třídy MujKalkulator.

1.5.2. Vytvoření testů pomocí JUnit

Nyní můžeme začít vytvářet testy. Jak již bylo uvedeno výše, použijeme testovací framework JUnit. Je to freeware nástroj, určený pro testování tříd napsaných v Javě (tento nástroj je také napsaný v Javě). Momentálně je k dispozici ke stažení na stránkách www.junit.org ve verzi 4.4. Součástí BlueJ je i podpora pro testování pomocí JUnit, využívána je verze 3.8.1.

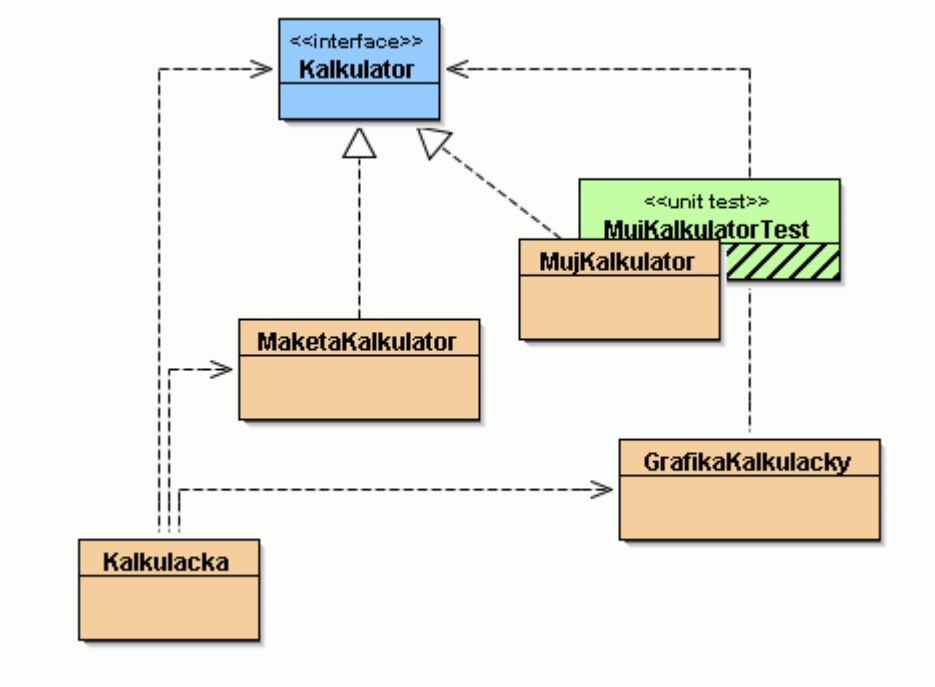
Pro testy platí několik pravidel:

- ke každé třídě, kterou chceme testovat, vytváříme samostatnou třídu s testy,
- třída s testy se jmenuje stejně jako testovaná, navíc má v názvu slovo test např. `MujKalkulatorTest`,
- testovací třída je potomkem třídy `junit.framework.TestCase`,
- testovací třída obsahuje metodu `protected void setUp()`, která se spouští před každým testem,
- testovací třída obsahuje metodu `protected void tearDown()`, která se spouští po každém testu,
- pro testování výsledků se používají metody `assertEquals()` a `fail()`.

Pomocí BlueJ je vytváření testů poměrně jednoduché, řadu těchto pravidel uplatní prostředí samo. Pro vytváření testů je nutné mít v BlueJ zpřístupněny nástroje pro testování. To uděláte pomocí volby **Nástroje ➔Nastavení** a v zobrazeném okně pak na záložce **Různé** zaškrtnete volbu **Zobrazit nástroje pro testování**. Vlevo dole, nad lištou virtuálního stroje pak přibudou tyto volby:

Obrázek 1.5 Volby pro testování přístupné v základním okně BlueJ

Základ testovací třídy v BlueJ vytvoříme velmi snadno. Klikněte pravým tlačítkem myši na grafické znázornění třídy `MujKalkulator` a v pop-up menu vyberte volbu **Vytvořit testovací třídu**. BlueJ vygeneruje základ testovací třídy, kterou v souladu s pravidly pojmenuje `MujKalkulatorTest`. V class diagramu je testovací třída zobrazena zeleně a umístěna tak, aby bylo jasné ke které třídě testy patří, viz Obrázek 1.6.



Obrázek 1.6 Class diagram projektu Kalkulačka po vytvoření testovací třídy.

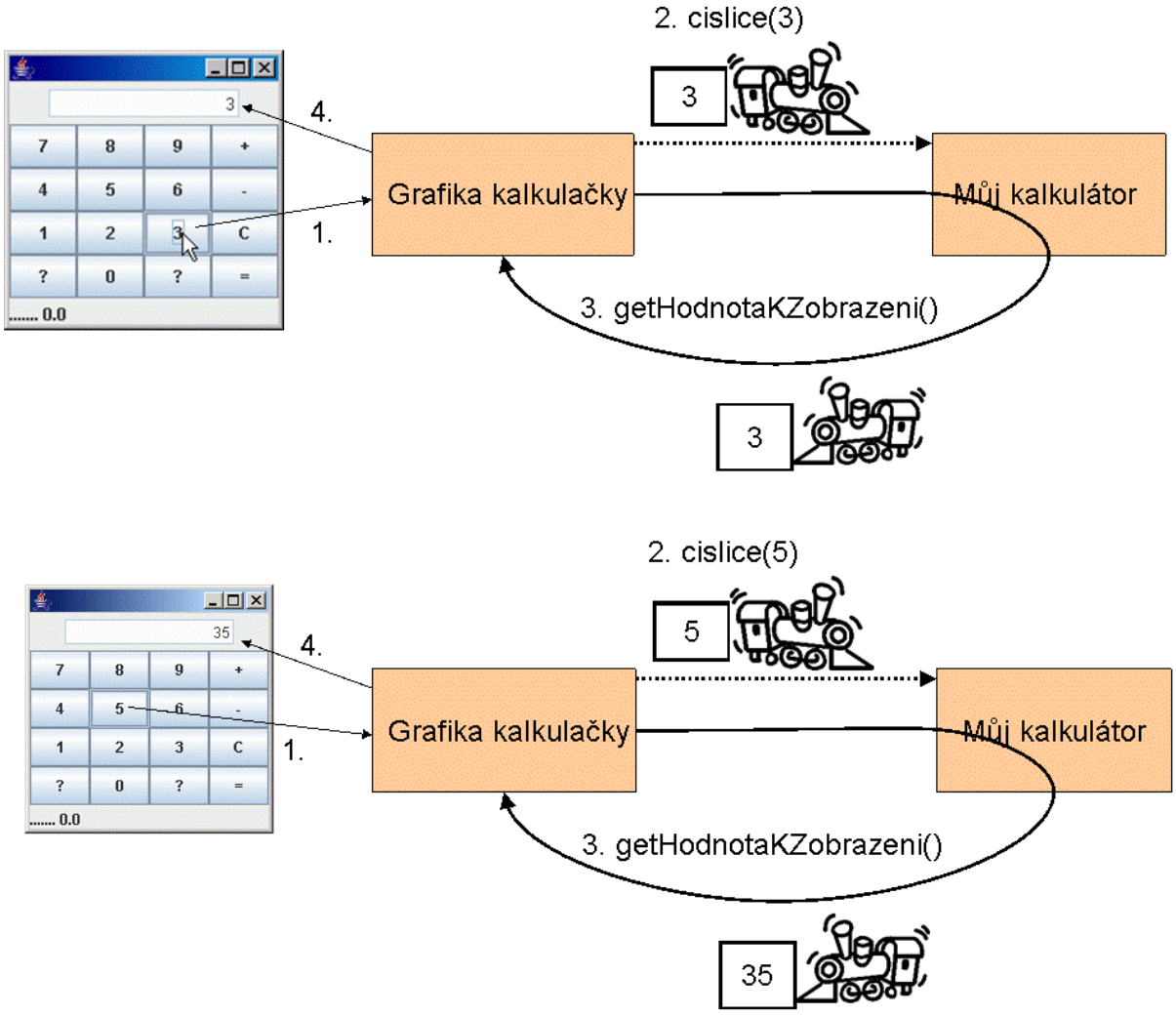
BlueJ za vás splnil i další podmínky platné pro testy v JUnit, třída `MujKalkulatorTest` je potomkem třídy `junit.framework.TestCase`, obsahuje metody `setUp()` a `tearDown()`. Navíc je zde prázdná testovací metoda `testXXX()`, která má sloužit jako vzor testovací metody. Tuto metodu doporučuji před překladem testu smazat.
Nyní se musíme zamyslet nad tím, jaké testy připravíme. Měli bychom otestovat, zda implementace kalkulátoru vrací správné výsledky na zadaná čísla a operace. Začneme od těch nejjednodušších po složitější. Také je třeba si uvědomit, že uživatel může např. dvakrát stisknout znaménko, zadávat delší příklady atd. V následující tabulce je uveden seznam testů, výsledky některých jsou jasné (35 je vždy 35, 25 + 12 = 37). Tam, kde je zdvojené znaménko apod., se musíte rozhodnout, jak bude vaše kalkulačka reagovat. 35 ++ může být v některé implementaci 35 a v jiné 70.

kombinace kláves	**hodnota na displeji**
3 5	35
3 5 C	0
3 5 + 2 =	37
3 5 + 2 = 3	3
3 5 + 2 + 4 =	41
3 5 + 2 +	
3 5 + +	
3 5 + =	
3 5 + + =	
3 5 + 2 - =	
3 5 - 2 =	33
3 5 - 2 - =	
3 5 = =	
3 5 - -	
3 5 + 2 = =	
3 5 + + 1 2 =	
3 5 - - 1 2 =	
3 5 - 2 + 3 =	36

Kromě číselných testů bychom měli otestovat i metody, které vracejí jméno autora a verzi. Na začátku každého testu musíme mít vytvořenou instanci třídy MujKalkulator, její vytvoření tedy může být součástí metody setUp(). Kód nemusíme psát sami, opět použijeme BlueJ. Vytvoříme instanci třídy MujKalkulator pomocí volby v příručním menu a pak v příručním menu testovací třídy použijeme volbu **Zásobník odkazů→testovací přípravek**. Vytvořená instance ze zásobníku odkazů zmizí a automaticky dojde ke změně a překladu kódu třídy s testy. Když si zdrojový kód prohlédnete najdete zde mimo jiné i tento kód:

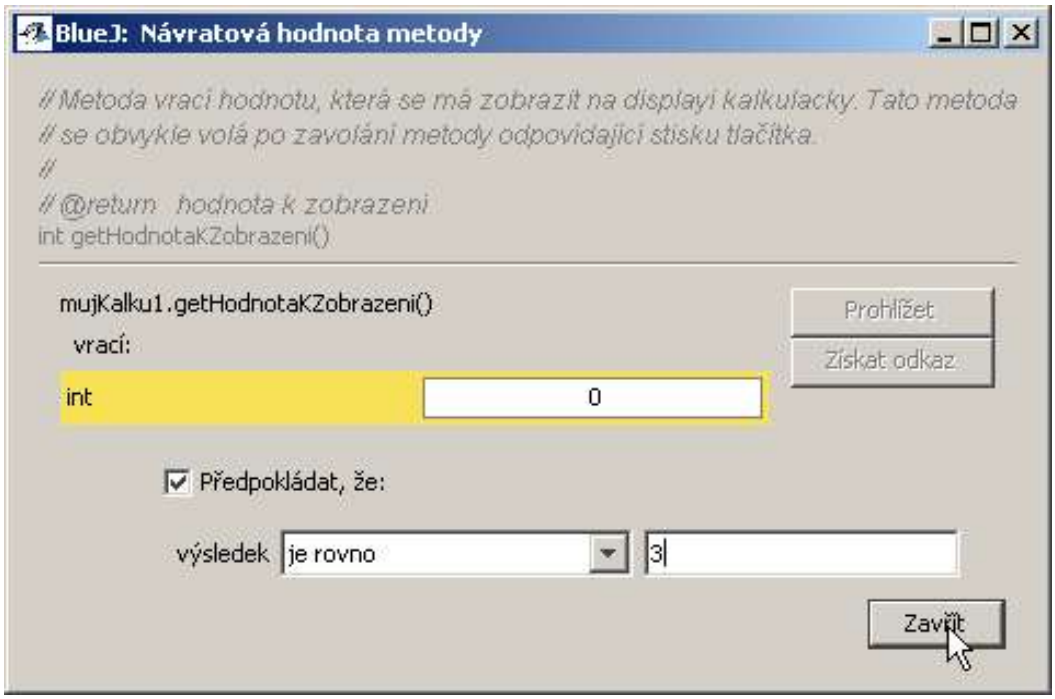
```
private MujKalkulator mujKalku1;

//část kódu vynechána

protected void setUp()
{
     mujKalku1 = new MujKalkulator();
}
```

Teď vytvoříme první test, který zkontroluje, že lze vkládat čísla, v našem testu konkrétně 35. Na vkládání čísel více testů dělat nebudeme, budeme věřit, že když kalkulačka umí 35 umí i další čísla. Jak probíhá zadávání čísla 35 na kalkulačce, jaké metody implementace rozhraní Kalkulator jsou volány? Připomeňme si to pomocí obrázků.



Obrázek 1.7 Znázornění komunikace mezi jednotlivými instancemi při zadávání čísla 35.

Testy vytvoříme pomocí BlueJ. V příručním menu u testové třídy vyberte volbu **Vytvořit testovací metodu...** a pojmenujte ji. Doporučuji jméno související s obsahem testu, v našem případě to může být 35 (BlueJ samo přidá na začátek metody slovo test, takže vznikne platný Java identifikátor a jsou splněny i podmínky platné pro JUnit). V zásobníku odkazů se objeví instance třídy `MujKalkulator`, kterou jsme předtím uložili do testového přípravku. Nyní budeme postupně volat metody, které by při reálném fungování kalkulačky volala instance třídy `GrafikaKalkulacky`. Zavoláme tedy metodu `cislice()` s parametrem 3 a pak metodu `getHodnotaKZobrazeni()`. Tato metoda vrací hodnotu a je možné otestovat ji pomocí metody `assertEquals()`. Jak se prostřednictvím BlueJ nastaví volání metody `assertEquals()` v tomto konkrétním případě ukazuje Obrázek 1.8. Dále pokračujeme obdobně, zavoláme metodu `cislice()` s parametrem 5 a metodu `getHodnotaKZobrazeni()`. U metody `getHodnotaKZobrazeni()` opět nastavíme testování výstupní hodnoty, tentokrát by se měla rovnat 35. Poté ukončíme nahrávání testu pomocí tlačítka **Ukončit** v menu pro testy viz Obrázek 1.5.

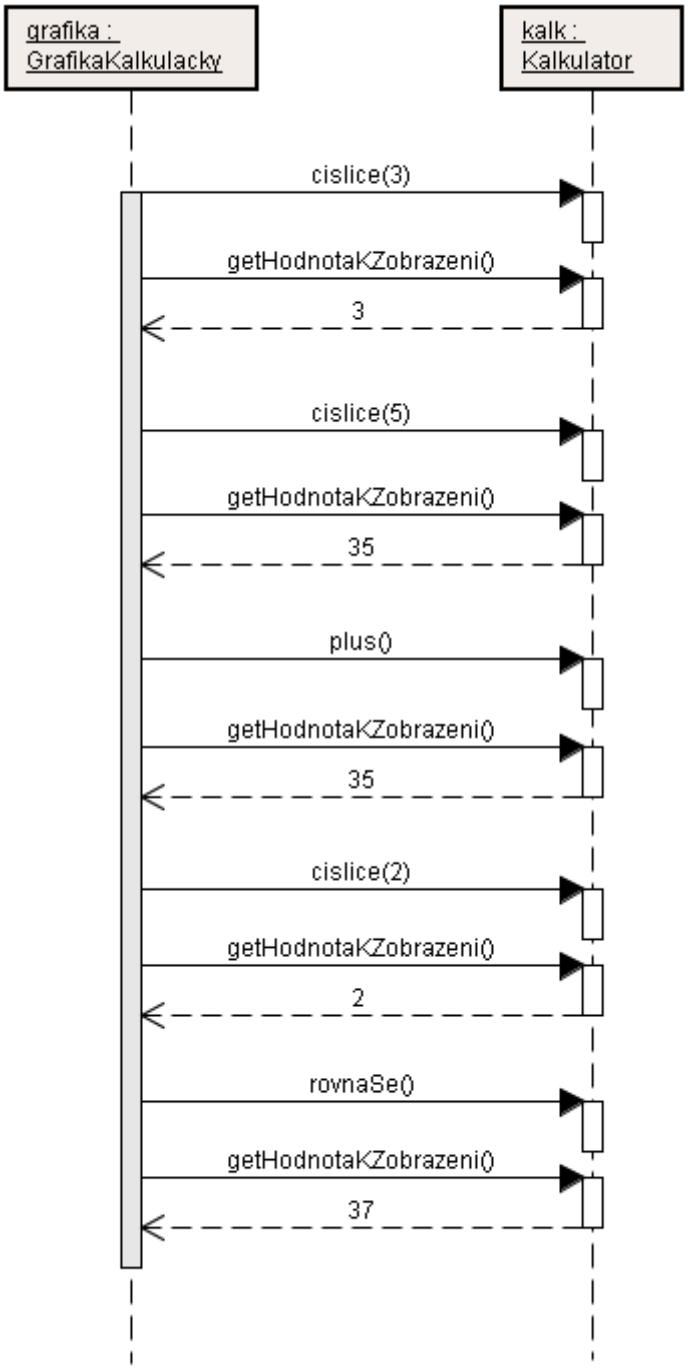


Obrázek 1.8 Zadávání hodnot pro testování návratové hodnoty metody.

Výsledkem této činnosti je potom kód testovací metody, který BlueJ vloží do třídy MujKalkulatorTest. Tento kód je uveden v následujícím výpisu:

```
public void test35()
{
     mujKalku1.cislice(3);
     assertEquals(3, mujKalku1.getHodnotaKZobrazeni());
     mujKalku1.cislice(5);
     assertEquals(35, mujKalku1.getHodnotaKZobrazeni());
}
```

Další testy se „naklikají“ obdobně. Pro ujasnění komunikace mezi instancemi tříd GrafikaKalkulacky a implementací rozhraní Kalkulator si ukážeme sekvenční diagram, který popisuje jaké metody s jakými parametry a návratovými hodnotami se volají. Sekvenční diagram zobrazuje průběh komunikace při zadávání výrazu 35 + 2 = 37. Na diagramu není znázorněn vstup od uživatele.



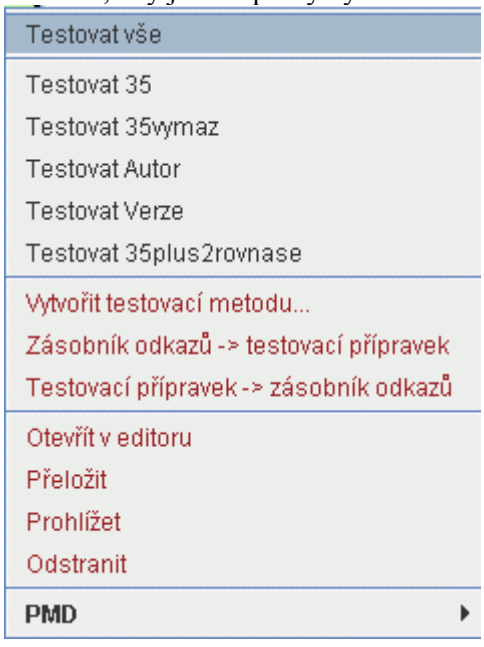
Obrázek 1.9 Diagram zobrazuje průběh komunikace mezi instancemi při zadávání 35+2=

V následujícím výpisu vidíte kód prvních tří testů z tabulky a testů na autora a verzi.

```
public void test35vymaz()
{
     mujKalku1.cislice(3);
     mujKalku1.cislice(5);
     mujKalku1.vymaz();
     assertEquals(0, mujKalku1.getHodnotaKZobrazeni());
}
```

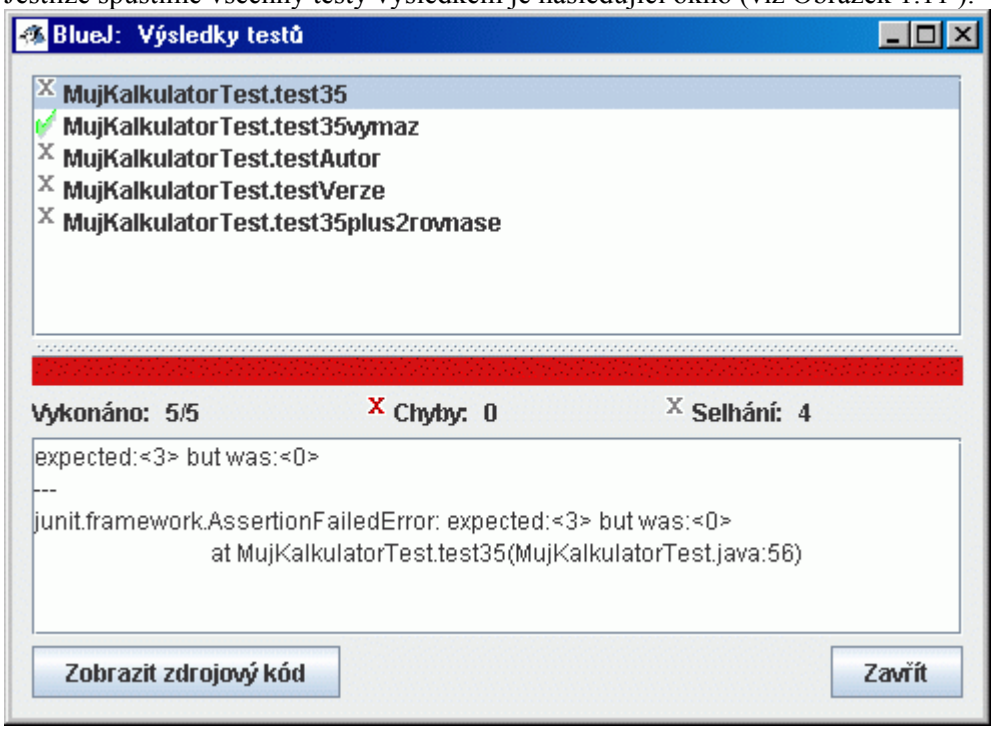
```
public void test35plus2rovnase()
{
     mujKalku1.cislice(3);
     mujKalku1.cislice(5);
     // to že kalkulačka zobrazí 35 víme z výsledku předchozího testu,
     // v testu nemusíme volat a porovnávat výsledek metody
     // getHodnotaKZobrazeni() vždy, ale jen tam, kde nás výsledek
     // zajímá v daném testu.
     mujKalku1.plus();
     assertEquals(35, mujKalku1.getHodnotaKZobrazeni());
     mujKalku1.cislice(2);
     assertEquals(2, mujKalku1.getHodnotaKZobrazeni());
     mujKalku1.rovnaSe();
     assertEquals(37, mujKalku1.getHodnotaKZobrazeni());
}

public void testAutor()
{
     assertEquals("Pavlíčková", mujKalku1.getAutor());
}

public void testVerze()
{
     assertEquals("1.0", mujKalku1.getVerze());
}
```

Testy není nutné vždy „naklikávat“, můžeme také psát přímo do kódu. Při dalších testech nemusíte již testovat každou navracenou hodnotu, stačí testovat to, že konečný výsledek odpovídá očekávané hodnotě.
Nyní si ukážeme jak testy spustit. V příručním menu třídy s testy najdete volbu Testovat vše a pak tolik voleb, pro spuštění jednotlivých testů, kolik jste vytvořili testovacích metod. Jak vypadá menu v situaci, kdy jsou napsány výše uvedené testy vidíte na Obrázek 1.10



Obrázek 1.10 Příruční menu testovací třídy s několika testy.

Jestliže spustíme všechny testy výsledkem je následující okno (viz Obrázek 1.11).



Obrázek 1.11 Okno, zobrazující výsledky testů.

V horní části okna vidíme seznam testů a je zde graficky znázorněno, jak jednotlivé testy dopadly. X znamená, že test neprošel, ✓, že prošel. To, že projde test na výmaz je dáno tím, že výsledkem má být nula a naše implementace zatím vrací nulu vždy. Jakmile začneme ladit vkládání čísel, test na výmaz neprojde, dokud tuto funkci nenaprogramujeme správně.
V prostřední části okna jsou shrnuty výsledky testů, červená lišta znamená, že alespoň jeden test neprošel. Našim cílem je dosáhnout toho, aby byla lišta zelená, což znamená, že všechny testy prošly.
V dolní části okna si můžete zobrazit detaily o každém testu. V našem případě je zde zobrazen detail testu na vkládání čísla 35, v popisu je vidět, že byla očekávána hodnota 3, ale naše implementace vrátila 0.
Testy lze spouštět i jednotlivě, pokud test neprojde, objeví se okno s výsledky. Pokud spouštíte jeden test a ten projde, okno s výsledky se nezobrazí, pouze v dolním levém rohu hlavního okna BlueJ se objeví hláška, že test prošel např. test35ymaz je OK.

1.5.3. Doplnění autora a verze

Po vytvoření všech testů můžeme přistoupit k implementaci metod v třídě `MujKalkulator`.
Budeme postupovat následovně:

- napíšeme část kódu,
- přeložíme ho,
- spustíme testy,
- zkontrolujeme výsledek testu, který testuje to, co právě řešíme
- pokud test prošel, můžeme řešit další část, pokud ne, opravíme kód a opět testujeme.

Nejjednodušší je doplnění metod `getAutor()` a `getVerze()`. Do kódu stačí pouze doplnit správné údaje. Po spuštění testů, by měly projít testy `testAutor()` a `testVerze()`.

1.5.4. Vkládání prvního čísla

Druhým krokem bude „naučit“ kalkulačku správně zobrazovat první zadávané číslo. Po každém stisku tlačítka na kalkulačce je zavolána metoda `cislice()`. V parametru hodnota dostaneme číslici, kterou uživatel zadal.
Mezi jednotlivými stisky tlačítek si musíme pamatovat, co již bylo zadáno. Budeme tedy potřebovat datový atribut pro uchovávání vkládaného čísla (a současně čísla, které se bude vracet metodou

getHodnotaKZobrazeni() k zobrazení na displeji). Toto číslo bude typu *int* (kalkulačka umí pouze celá čísla) a jeho deklarace může vypadat následovně:

```
private int hodnotaKZobrazeni;
```

V konstruktoru přiřadíme do datového atributu počáteční hodnotu[1] 0. Ve výše uvedeném příkladu by po stisknutí klávesy 3 (tj. na konci provádění metody *cislice()*) měl tento datový atribut obsahovat hodnotu 3, po následném stisknutí klávesy 5 hodnotu 35. Postupným vkládáním jednotlivých číslic se skládá hodnota čísla. Číslo je zadáváno zleva, při zadání další číslice se předchozí hodnota zvětší o jeden řád (z jednotek budou desítky, z desítek stovky atd.) a přičte se k ní hodnota naposledy vložené číslice. Kód metody *cislice()* by mohl vypadat následovně:

```
public void cislice(int hodnota) {
    this.hodnotaKZobrazeni = this.hodnotaKZobrazeni*10+hodnota;
}
```

Proměnná *hodnotaKZobrazeni* je uvozena klíčovým slovem *this*, které zdůrazňuje, že se jedná o datový atribut této instance. Pokud nedochází ke kolizi jména datového atributu s lokální proměnnou či jménem parametru, není potřeba v Javě toto klíčové slovo uvádět. Kód proto může vypadat následovně:

```
public void cislice(int hodnota) {
    hodnotaKZobrazeni = hodnotaKZobrazeni*10+hodnota;
}
```

Všimněte si též parametru metody *cislice()* – deklarace se skládá z typu (*int*) a identifikátoru (*hodnota*). Důležitý je datový typ – při volání metody se kontroluje pouze typ. Identifikátor slouží pro programátora metody – pomocí tohoto identifikátoru se programátor na parametr metody odkazuje. Není problém tento identifikátor změnit, aniž by bylo potřeba něco měnit ve třídách/metodách, které tuto metodu volají. Metoda by mohla vypadat následovně:

```
public void cislice(int cislo) {
    hodnotaKZobrazeni = hodnotaKZobrazeni*10+cislo;
}
```

Metoda *getHodnotaKZobrazeni()* vrací obsah datového atributu *hodnotaKZobrazeni*, její kód bude vypadat následovně:

```
public int getHodnotaKZobrazeni() {
    return hodnotaKZobrazeni;
}
```

Přeložte aplikaci a spusťte testy, projít by měl i *test35()*, všimněte si, že test na výmaz již neprojde.Nyní již můžete vkládat čísla do kalkulačky, narazíte však na dva problémy:

- pro zadání nového čísla je potřeba znovu spustit kalkulačku,
- při vložení většího počtu číslic dojde k přetečení rozsahu číselného typu *int* (přibližně 2 miliardy) – tento problém nebudeme řešit.

1.5.5. Operace výmaz (C)

Po stisknutí klávesy výmaz (C) se volá metoda *vymaz()* a měla by se vynulovat hodnota na displeji. Řešení této situace je poměrně jednoduché – v metodě *vymaz()* by se měla vynulovat hodnota datového atributu *hodnotaKZobrazeni*. Kód metody *vymaz()* bude vypadat následovně:

```
public void vymaz() {
    hodnotaKZobrazeni = 0;
}
```

Zda je to správně si opět ověříme pomocí testu.

[1] Toto přiřazení není nutné – Java automaticky číselným datovým atributům přiděluje počáteční nulu. V případě lokálních proměnných to ale neplatí. I z tohoto důvodu je vhodné si zvyknout vždy přiřazovat počáteční hodnotu datovým atributům a lokálním proměnným.

1.5.6. Operace plus (+)

Nyní budeme řešit situaci, kdy uživatel vloží první číslo (např. 35) a stiskne tlačítko se znaménkem plus.

Po stisknutí tlačítka plus by měla na displeji zůstat zobrazena hodnota 35. Musíme si zapamatovat první číslo momentálně uložené v datovém atributu `hodnotaKZobrazeni`. Při vkládání druhého čísla se obsah tohoto atributu přepíše, je tedy nutné uložit jeho obsah do dalšího datového atributu. Bude opět typu `int` a můžeme ho pojmenovat např. `prvniOperand`, deklarace bude vypadat takto:

```
    private int prvniOperand;
```

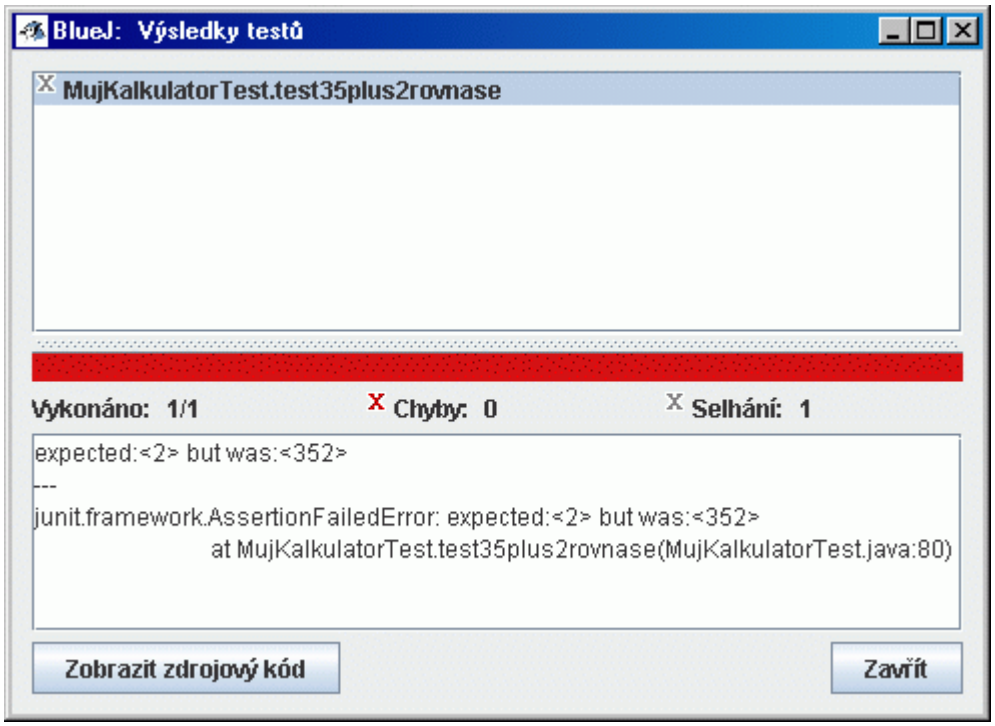
V konstruktoru mu přiřadíme jako počáteční hodnotu nulu. Metoda `plus()` vypadá nyní následovně:

```
public void plus() {
    prvniOperand = hodnotaKZobrazeni;
}
```

Po vložení dalšího čísla a stisknutí tlačítka rovná se (=) by se na displeji měl zobrazit výsledek. V metodě `rovnaSe()` se sečte první operand s aktuálně vloženým číslem a výsledek se uloží do datového atributu `hodnotaKZobrazeni`:

```
public void rovnaSe() {
    hodnotaKZobrazeni = prvniOperand + hodnotaKZobrazeni;
}
```

Nyní máme nějakým způsobem naimplementováno sčítání a je nutné ověřit si, zda je implementace správná. Spustíme tedy test 35plus2rovnase. Výsledek testů nás nepotěší, test neprojde. Okno s výsledky testu viz Obrázek 1.12.



Obrázek 1.12 Výsledek testu

Jak je vidět z popisu chyby, problém nastává již při vkládání druhého operandu. Nezačne se zobrazovat nové číslo, ale číslice se připojí na konec prvního operandu. Tj. po následující posloupnosti kláves

se na displeji zobrazí číslo 352. V proměnné *hodnotaKZobrazeni* zůstává původní číslo a dále se „prodlužuje“. Zkusíme to vyřešit tak, že tuto proměnnou vynulujeme poté, co hodnotu, která zde byla uložena přesuneme do proměnné *prvniOperand*.

```
public void plus(){
     prvniOperand = hodnotaKZobrazeni;
     hodnotaKZobrazeni = 0;
}
```

Znovu spustíme test, ten opět neprojde, jen z výpisu je patrné, že místo očekávaných 35 je po stisknutí plus zobrazena nula. My sice potřebujeme proměnnou *hodnotaKZobrazeni* vynulovat, ale až o krok později. Pro ošetření této situace potřebujeme v metodě *cislice()* odlišit situaci, kdy se začne vkládat nová hodnota. Tento datový atribut můžeme pojmenovat např. *noveCislo* a bude typu *boolean*, tj. bude nabývat hodnot *true* (uživatel začíná vkládat nové číslo) nebo *false* (uživatel pokračuje vložením stávajícího čísla). V konstruktoru zinicializujeme tento datový atribut s hodnotou *true* (po spuštění kalkulačky začíná uživatel s vkládáním nového čísla).

```
     private boolean noveCislo;
```

V metodě *plus()* tedy nebudeme nulovat datový atribut *hodnotaKZobrazeni*, ale nastavovat datový atribut *noveCislo* na *true*.

```
public void plus(){
    prvniOperand = hodnotaKZobrazeni;
    noveCislo = true;
}
```

Musíme změnit i kód metody *cislice()*. Pokud je v atributu *noveCislo* hodnota *true*, musíme do atributu *hodnotaKZobrazeni* vložit obsah parametru metody a změnit obsah atributu *noveCislo* na *false*. Metoda *cislice()* bude vypadat takto:

```
public void cislice(int hodnota) {
     if (noveCislo) {
          hodnotaKZobrazeni = hodnota;
          noveCislo = false;
     }
     else {
          hodnotaKZobrazeni = hodnotaKZobrazeni*10 + hodnota;
     }
}
```

Znovu zkusíme otestovat 35plus2rovnase a tentokrát již test projde. Máme tedy naimplementováno jednoduché sčítání a nyní budeme implementovat odčítání.

1.5.7. Operace mínus (−)

Místo plus může uživatel stisknout klávesu minus:

Obdobně jako u sčítání se výsledek počítá v metodě *rovnaSe()*. Abychom odlišili sčítání od odčítání, musíme v metodách *plus()* a *minus()* uložit nejen první operand, ale i typ operace. Pro uložení požadované operace se nabízí několik možností:

- použijeme datový atribut typu *char*,
- použijeme datový atribut typu *String*,
- použijeme datový atribut typu *int* a nadefinujeme konstanty pro jednotlivé operace,
- použijeme výčtový datový typ pro označení operací.

Ukážeme si nejprve použití prvního řešení, potom i řešení dle třetí varianty. Druhá varianta je velmi podobná první, jen si musíme uvědomit, že *String* je referenční datový typ a pro porovnávání je třeba použít metodu *equals()* a ne operátory == či !=. Čtvrtou variantu si ukazovat nebudeme, použití výčtového typu bude ukázáno v projektu Trojúhelníky.
Datový atribut typu *char* pro uložení typu operace pojmenujeme *operator* a jeho deklarace bude vypadat takto:

```
    private char operator;
```

V konstruktoru budeme atribut *operator* inicializovat hodnotou mezera (nezapomeňte, že znaky se uvozují apostrofy, ne uvozovkami). Tato hodnota bude znamenat, že není požadována žádná operace. V metodě *plus()* musíme přidat uložení typu operace do proměnné *operator*, metoda *minus()* je velmi podobná metodě *plus()*:

```
public void plus() {
    prvniOperand = hodnotaKZobrazeni;
    noveCislo = true;
    operator='+';
}
public void minus() {
    prvniOperand = hodnotaKZobrazeni;
    noveCislo = true;
    operator='-';
}
```

V kódu metody *rovnaSe()* je potřeba pomocí selekce *if* rozlišovat, zda uživatel stiskl tlačítko plus či minus. Po provedení výpočtu se do proměnné *operator* vloží mezera:

```
public void rovnaSe() {
    if (operator == '+') {
        hodnotaKZobrazeni = prvniOperand + hodnotaKZobrazeni;
    }
    else {
        if (operator == '-') {
            hodnotaKZobrazeni = prvniOperand - hodnotaKZobrazeni;
        }
    }
    operator=' ';
    noveCislo = true;
}
```

Správnost ověříme opět testy, měl by projít test na jednoduché sčítání i odčítání.

1.5.8. Operace plus – pokračování

Zatím jsme uvažovali o situaci, že uživatel sčítal/odčítal jen dvě čísla. Uživatel může však sečíst více čísel:

Na začátku metody *plus()* musíme zjistit, zda uživatel v předchozím kroku již požadoval nějakou operaci nebo ne. Pokud ano, je třeba dříve zadanou operaci (může to být i minus) nejdříve provést, její výsledek zobrazit a též uložit jako *prvniOperand*. Kód metody *plus()* by mohl vypadat takto:

```
public void plus() {
     if (operator == '+') {
          prvniOperand = prvniOperand + hodnotaKZobrazeni;
     }
     else {
          if (operator == '-') {
               prvniOperand = prvniOperand - hodnotaKZobrazeni;
          }
          else {
               prvniOperand = hodnotaKZobrazeni;
          }
     }
     operator='+';
     hodnotaKZobrazeni = prvniOperand;
     noveCislo=true;
}
```

Metoda *minus()* bude z větší části duplicitní s metodou *plus()*. Duplicit v kódu je však ještě více, velká část kódu metody *rovnaSe()* je také stejná. Z důvodu lepší udržovatelnosti a rozšiřovatelnosti kódu je vhodné shodný kód umístit do jedné privátní metody. Pokud do kalkulačky přidáme ještě násobení a dělení, budeme výpočty řešit pouze v jedné metodě. Metodu, do které přesuneme společný kód, nazveme *vypocet()*. Je to pomocná metoda pro jiné metody ze třídy *Kalkulator* a není třeba (není to ani vhodné) ji volat z jiných tříd – proto bude označena modifikátorem *private*. Kód metody *vypocet()* a upravený kód metod *plus()* a *rovnaSe()* vidíme na následujícím výpise. Kód metody *minus()* bude analogický s kódem metody *plus()*.

```
/**
 * metoda se volá při stisknutí tlačítka "+" (plus) na kalkulačce
 */
public void plus() {
     vypocet();
     hodnotaKZobrazeni=prvniOperand;
     noveCislo=true;
     operator='+';
}
/**
* metoda se volá při stisknutí tlačítka "=" (rovná se) na kalkulačce
*/
public void rovnaSe() {
     vypocet();
     hodnotaKZobrazeni = prvniOperand;
     noveCislo=true;
     operator=' ';
}

/**
 * metoda spočítá mezivýsledek a uloží ho do proměnné prvniOperand
 */
private void vypocet () {
     if (operator == '+') {
          prvniOperand = prvniOperand + hodnotaKZobrazeni;
     }
     else {
          if (operator == '-') {
               prvniOperand = prvniOperand - hodnotaKZobrazeni;
          }
          else {
               prvniOperand = hodnotaKZobrazeni;
          }
     }
}
```

1.5.9. Řešení s konstantami

Řešení, ve kterém jsou pro označení operací místo znaků (typu *char*) použity pojmenované konstanty, se bude od předchozího lišit v několika drobnostech. Konstanty pro jednotlivé operace deklarujeme jako datové atributy s modifikátorem *final*:

```
    private final int ZADNA_OPERACE = 0;
    private final int OPERACE_PLUS = 1;
    private final int OPERACE_MINUS = 2;
```

Datový atribut *operator* bude typu *int* a v konstruktoru nastavíme jeho počáteční hodnotu na *ZADNA_OPERACE*. V metodách *plus()*, *minus()* a *rovnaSe()* budeme místo znaků přiřazovat do tohoto datového atributu odpovídající konstantu. Metoda *vypocet()* se změní takto:

```
/**
 * metoda spočítá mezivýsledek a uloží ho do proměnné prvniOperand
 */
private void vypocet () {
    if (operator == OPERACE_PLUS) {
        prvniOperand = prvniOperand + hodnotaKZobrazeni;
    }
    else {
        if (operator == OPERACE_MINUS) {
            prvniOperand = prvniOperand - hodnotaKZobrazeni;
        }
        else {
            prvniOperand = hodnotaKZobrazeni;
        }
    }
}
```

1.6. Domácí úkoly

1. Dokončete implementaci třídy *MujKalkulator* tak, aby prošly všechny testy.
2. Předělejte vnořené příkazy *if* na příkaz *switch* v metodě *vypocet()*.