Krok 2: Jednoduché ošetření pomocí odchycení výjimky

Vytvoříme v projektu nový formulář zkopírováním předchozího formuláře. Provedeme v panelu Solution Exploreru. Klepneme na Form1 a stiskneme CTRL+C, potom na projekt a CTRL+V. Nová třída dostane jméno Copy of Form1, přejmenujeme na Form2. Názvy tříd v obou parciálních třídách se bohužel automaticky nepřejmenují, učiníme tak tedy ručně. Pak ve třídě Program.cs nasměrujeme na tento nový formulář spouštění.
Vložíme tělo metody btSecti_Click() do bloku try-catch. V sekci catch bude vyčištění textu zadávacích textboxů, takže po zadání chybného znaku budou smazány i případné předchozí platné znaky.

Krok 3: ošetření se zachováním platných znaků

Nevýhodou předchozího jednoduchého způsobu je to, že po zadání nějakého neplatného znaku se vymažou i případné předchozí platné znaky. Ukážeme si způsob, jak tyto znaky zachovat. Budeme muset vytvořit nějaké úložiště pro dosud zadané platné znaky. Postup je pak takový, že po ověření, že další zadaný znak je platný, tak se aktualizuje řetězec v úložišti (zápis obsahu textBoxu do úložiště). Naopak po zadání neplatného znaku se obsah textBoxu nahradí obsahem úložiště (tedy posledním platným řetězcem). Předchozí stav prvního textBoxu si bude pamatovat řetězcová proměnná posledníPlatnyText1, podobná proměnná bude i pro druhý textBox.
Jakou událost jakého aktivního prvku budeme odchytávat?
???
Tato procedura zajistí, že v textBoxu bude vždy řetězec, který lze převést na číslo. Proto je zbytečné zachytávání výjimky po klepnutí na tlačítko Sečti. Takže obsluha klepnutí na tlačítko se vlastně vrátí do stavu, ve kterém byla ve Form1.
Proto nejprve vytvoříme další formulář (Form3) zkopírováním Form1 (tedy ne Form2) a provedeme všechna nutná přejmenování. Pak vložíme událost. Poklepeme na prvním textBoxu. Vloží se:

```
private void   tBox1_TextChanged(object sender, EventArgs e) {    }
```

Ze začátku stačí, když se v případě chybného znaku smaže celý textbox. Funkčnost se sice nezmění, ale ověříme si přesun odchytávání výjimky z handleru tlačítka Secti do handleru textboxů:
Do handleru tBox1_TextChanged vložíme snippet try. V sekci try zavoláme metodu Parse. Zkopírujeme z handleru tlačítka Secti, ale výsledek metody nikam neukládáme. Jde jen o ověření, zda vlastnost Text lze převést na řetězec. A do sekce catch zkopírujeme sekci catch z handleru tlačítka Secti (pozor, mažeme jen obsah tBox1.Text, ne tBox2.Text). Tytéž změny provedeme pak pro druhý textBox. Poté již přestaneme zachytávat výjimku v handleru. Ověříme funkčnost.
Dále přepíšeme handler tBox1_TextChange. Pokud se volání int.Parse povede, pak je tento text zřejmě v pořádku. Proto další řádek zajistí vložení aktualizovaného řetězce do proměnné posledniPlatnyText. Tím končí blok try.
Pokud se převod nepovede, tak se vyhodí výjimka a v sekci catch zajistíme přepsání textového pole obsahem proměnné posledniPlatnyText (tedy opačné přiřazení).

Kde budeme proměnnou PosledniPlatnyText deklarovat? Zkuste obě možnosti a ověřte chování. Rozumíte, proč se chová tak, jak se chová?
Handler tBox2_TextChanged bychom mohli dostat mírnou úpravou handleru tBox1. Ale my jej v rámci tréningu zkusíme napsat jinak. Abychom si ověřili, že zpracování výjimek se někdy používá místo větvení, tak v tomto handleru použijeme větvení if-else, v podmínce bude volání metody int.TryParse, která vrací boolean.
Program spustíme. Vidíme, že špatně zadaný znak jen na milisekundu problikne a předchozí obsah textBoxu zůstává nezměněn.
Poté v handleru tBox2_TextChanged zakomentujeme sekci if-else a zkopírujeme sem řešení pomocí try-catch z handleru tBox1_TextChanged (a mírně upravíme). Verzi if-else nebudeme dále rozvíjet.

Krok 3b: Další krok (3): drobná nedokonalost: po zapsání špatného znaku a vyčištění textboxu skočí kurzor na začátek. A neposune se po vložení posledního platného textu. Zkuste spravit, v intellisense věnujte pozornost vlastnosti SelectionStart
Nakonec (4) drobné vylepšení: Ošetření situace, kdy v textBoxu není nic (aby to bylo chápáno jako znak nula a ne jako prázdný řetězec). Protože při prázdném řetězci by to při sčítání vyhodilo výjimku.
A s tím souvisí problém, že při mazání pomocí BackSpace nejde vymazat 1. číslice, protože po vymazání 1. číslice zůstane prázdný řetězec, který vyhodí výjimku a ta tam vždy vrátí poslední platný text, tedy první číslici.

Krok 4:

Předchozí program již pracuje tak, jak má. Ale jeho nevýhodou je, že kód pro první i druhý textBox je prakticky identický. A co kdyby takových textBoxů bylo deset!? Proto by bylo lepší, kdyby metoda byla jen jedna (nazveme ji tBox_TextChanged) a oba textBoxy by volaly tuto jedinou metodu. Což zajistíme ve skryté části Form4.Designer: událost TextChanged na obou textBoxech bude volat stejnou metodu, tedy tBox_TextChanged. Původně (po poklepání na textboxech) je:
Form4:
```
private void   tBox1_TextChanged(object sender, EventArgs e) {    }
private void   tBox2_TextChanged(object sender, EventArgs e) {    }
```
Form4.Designer:
```
this.tBox1.TextChanged += new
System.EventHandler(this.tBox1_TextChanged);
this.tBox2.TextChanged += new
System.EventHandler(this.tBox2_TextChanged);
```
Pro srovnání: V jazyku Visual Basic není roztržení do dvou částečných tříd, po poklepání na tBox1 se vloží subrutina s hlavičkou:
Private Sub tBox1_TextChanged (ByVal sender As System.Object,ByVal e As System.EventArgs) Handles tBox1.TextChanged
Po úpravě bude ve Form4:
private void   tBox_TextChanged(object sender, EventArgs e) {   }
a ve Form4.Designer:
```
this.tBox1.TextChanged += new
System.EventHandler(this.tBox_TextChanged);
```

```
this.tBox2.TextChanged += new
System.EventHandler(this.tBox_TextChanged);
```
Jak ale metoda pozná, zda byla vyvolána událostí změny textu prvního nebo naopak druhého textBoxu? Tato informace se metodě předává jako parametr sender. Jedná se o parametr typu object, ve skutečnosti se však jedná o objekt tBox1 nebo tBox2, proto je možné přetypování zpět na typ TextBox. Po přetypování uložíme do proměnné tb.
```
TextBox tb = (TextBox)sender;
```
První činností tohoto kroku tedy bude přejmenování metody tBox1_TextChanged na tBox_TextChanged. Poté provedeme totéž přejmenování i v skryté části parciální třídy Form4.Designer. A totéž i pro druhý TextBox.
Na první řádek metody vložíme deklaraci proměnné tb třídy MujTextBox (viz níže) a vložíme do ní přetypovaný sender. A do pomocné proměnné pom pak budeme ukládat ne tBox1.Text či tBox2.Text, ale tb.Text.
Funkčnost programu ověřte spuštěním, ale zadávejte jen čísla. Protože ošetření chybného znaku ještě nemáme.

Potom (2) musíme vyřešit ukládání posledního platného textu jednotlivých textBoxů. Předchozí program ukládal poslední platný text obou textboxů do dvou datových složek formuláře. Pokud by bylo textových polí hodně, pak by muselo být i hodně těchto datových složek. Což by bylo poněkud nepřehledné. Lepší by bylo, kdyby měl tuto datovou složku každý textBox. Prostě principem OOP je, že si každá třída nese všechno potřebné s sebou. Problém ale je, že třída TextBox žádnou takovou datovou složku nemá. Tak tedy vytvoříme vlastní třídu MujTextBox, která bude dědit z třídy TextBox a navíc bude mít tuto datovou složku (tedy posledniPlatnyText). Soubor vytvoříme běžným způsobem pomocí Add/Class (tedy ne Add/UserControl) z místní nabídky projektu. Na začátek přidáme
```
using System.Windows.Forms;
```
A do třídy umístíme datovou složku. Nic víc:
```
using System; using System.Windows.Forms;
namespace FormTextBox {
  public class MujTextBox : TextBox {
    public String posledniPlatnyText;
  }
}
```
Po kompilaci získá třída v panelu Project Explorer ikonku:
Nový aktivní prvek se po kompilaci také automaticky nabídne v ToolBoxu, a to úplně nahoře, ve skupině FormTextBoxComponents. Do formuláře pak umístíme dva tyto nové aktivní prvky (budou sloužit pro zadávání čísel). Nejlépe je vložit je navíc (tedy předchozí textBoxy nechat). Druhá možnost je přejmenovat stávající z TextBox na MujTextBox (zde je ale nebezpečí, že se někde zapomene přejmenovat a program přestane fungovat). TextBox pro zobrazení výsledku bude používat klasický TextBox (protože do tohoto textBoxu nezadáváme, není tedy nebezpečí špatného zadání).

Následující (3.) činností bude v metodě tBox_TextChanged ve Form4 změna datového typu dočasné proměnné tb z TextBox na MujTextBox (totéž u přetypování). A posledniPlatnyText už nebude datovou složkou formuláře, ale textboxu tb. Program odzkoušíme.

Dále (4.) aby to bylo programátorsky košer, tak datová složka posledniPlatnyText bude privátní a tedy doplníme její setter a getter.

A poté v metodě tBox_TextChanged ve Form4 místo přiřazovacího příkazu použijeme getter a setter
Poslední (5.) činností bude v rámci tréninku vytvoření v třídě MujTextBox ještě ekvivalent setteru a getteru, tedy Property:
Poté upravíme ve formuláři ještě používání Property místo getteru a setteru.

Krok 5: Jiný způsob ošetření: použití Hashtable

Předchozí program ukládal poslední platný text do příslušné datové složky každého textboxu. Pro procvičení si ukážeme i jiný způsob, kdy poslední platné texty uložíme centrálně (tedy ne zvlášť v každém textboxu, ale v datové složce formuláře). Řešení by mělo být univerzální, tedy použitelné pro libovolný počet textBoxů. Šlo by tedy použít pole řetězců. Pole má ale jako index číslo, což je problém, protože textová pole nejsou očíslována. Mohla by se v roli indexu použít jejich vlastnost TabIndex. Je ale v tom případě nutné na to při tvorbě formuláře pamatovat a umisťovat textBoxy pěkně za sebou (tedy ne na přeskáčku). Což je zbytečné omezení, co když dodatečně budeme chtít přidat ještě jedno textové pole? Lepší je Hashtable. To může mít jako index např. řetězec, tedy např. název textového pole. Nebo ještě lépe přímo celý objekt, v našem případě objekt typu sender, čili objekty typu TextBox. Je to tedy vlastně dvojrozměrné pole, kde v prvním sloupci je klíč (index), v tomto případě typu TextBo. A v druhém sloupci je hodnota, v tomto případě řetězec. Vysvětlení je v kapitole Kolekce v dílu OOP v jazyku C#. Pozor: musíme přidat using System.Collections, protože Hashtable není generická kolekce (předělání na generickou kolekci Dictionary bude v dalším kroku).
Postup: Nebudeme používat novou třídu MujTextBox. Proto vyjdeme z kroku 3. Takže pomocí CTRL+C, CTRL+V zkopírujeme Form3 a kopii přejmenujeme na Form5. Provedeme všechna další související přejmenování.
Pro srovnání: první (nejjednodušší) verze kroku 3 byla:

```
public partial class Form3 : Form {
  string posledniPlatnyText1 = "", posledniPlatnyText2 = "";
  private void tBox1_TextChanged(object sender, EventArgs e) {
    try {
      int.Parse(tBox1.Text);
      posledniPlatnyText1 = tBox1.Text;
    }
    catch (Exception) {tBox1.Text = posledniPlatnyText1; }
  }
```

Stejně jako v kroku 4 chceme mít jeden handler, který volají události tBox1.TextChanged i tBox2.TextChanged. Provedeme tedy stejné změny v Designeru, jako v kroku 4 a handler přejmenujeme na tBox_TextChanged. A z Form4 zkopírujeme první řádek handleru, tedy
`TextBox tb = (TextBox)sender;`
Pozor, ne MujTextBox, takovou třídu zde nemáme.
Takže předchozí kroky byly rutinní, totožné s postupem v kroku 4. A nyní začneme pracovat s HashTable. Základem úspěchu je, abychom nevytvářeli úplně nový program. Neboť co se osvědčilo, do toho se nevrtá. Provedeme tedy změny pouze v místech, kde se vyskytovala datová složka PosledniPlatnyText1 či 2. A nahradíme ji instancí Hashtable.

a) Na začátku třídy Form5 je deklarace datových složek PosledniPlatnyText1 a 2. Zakomentujeme a pod to umístíme deklaraci datové složky PosledniPlatneTexty typu Hashtable.
b) Když chceme do příslušné buňky kolekce zapsat řetězec pro textBox (sekce try), který byl zdrojem události (např. to byl tBox1), tak v tabulce hledáme objekt s tímto indexem (klíčem). Stejně jako u běžného pole je index (klíč) v hranaté závorce. Pokud tabulka tento klíč obsahuje (to nám řekne výsledek metody ContainsKey, doslova „obsahuje klíč“), tak do položky zapíšeme obsah textového pole. Pokud neobsahuje (protože uživatel do daného textBoxu teprve zapsal první znak), tak položku s tímto klíčem přidáme (metoda Add). Cituji ze skript OOP, z kapitoly Kolekce/Hashtable (cca str. 55):

```
if (!tabulka.ContainsKey("Karel"))
  tabulka.Add("Karel", 15000);
else
  tabulka["Karel"] = 15000;
```

Takže zakomentujeme zápis do datové složky PosledniPlatnyText1 (či 2) a pod toto místo zkopírujeme ty čtyři řádky ze skript OOP. A pak si ujasníme, co v našem programu bude místo "Karel" a co místo 15000. A pak přepíšeme.
Zakomentujeme vnitřek sekce catch (která ještě není upravená) a program odzkoušíme. Musíme ale zadávat jen platné číslice
c) Pokud naopak chceme z tabulky číst (sekce catch), tak také nejprve zjistíme, zda položku s daným klíčem obsahuje. Na začátku zápisu do textBoxu ještě položku neobsahuje (tedy není dosud žádný platný text), v tom případě do textBoxu napíšeme prázdný řetězec. Pokud už něco uloženo je, tak přiřadíme příslušnou položku tabulky do textového pole.
Takže opět zakomentujeme čtení z PosledniPlatnyText1 (či 2) a místo toho vložíme strukturu if-else, podobnou jako v sekci try. Ale přiřazovací příkaz bude opačný, tedy prohodí se levá a pravá strana.

Krok 6
předělejte na použití generické kolekce Dictionary, přidejte ošetření absence textu v textboxu (aby bylo chápáno jako nula), viz závěr kroku 3

Krok 7: Oddělení aplikační logiky do samostatného projektu

Aby se náš program podobal komerčním programům, tak jej rozdělíme do tří projektů. Vytvoříme pro ně nové solution s názvem Csharp-pCvFormTxBx, v něm projekt GUI typu Class library, v něm bude formulář Form6. Metodu secti() přesuneme do projektu Knihovna (zde by byly umístěny všechny výpočetní a jiné metody) typu Class Library, do třídy Kalkulacka. A vytvoříme rovněž spouštěč, projekt bude typu Console Application, dostane název Aplikace. V ní bude třída Program, ve kterém bude metoda Main(). Nyní tento odstavec rozepíšeme podrobně:
1) V solution vytvoříme projekt typu Class Library, formulář zkopírujeme z Form5 (tedy rozpracujeme dále variantu s hashovací tabulkou, ne MujTextBox), dáme mu název Form6. Zkopírujeme jej tak, že z místní nabídky cílového projektu vybereme Add/Existing item a "nabrouzdáme" Form5 v předchozím projektu. Tím se vloží nejen soubor Form5.cs ale i [Designer]. Jiné způsoby kopírování (kopírovat v Průzkumníku soubory nebo kopírovat obsahy souborů) by se musely dělat nadvakrát, pro Form5.cs a pro [Designer]. Potom ještě

musíme přidat do referencí projektu System.Windows.Forms, protože projekt vytvořený jako Class Library tuto referenci standardně nemá.

Po kompilaci si ověřte, zda se v dané složce objeví soubor typu exe či dll.
tak schválně, který?
Dále přidáme projekt Console app, nazveme jej Aplikace. V třídě Program je metoda Main, zkopírujeme do ní obsah metody Main z kroku 5. Je zde vytvoření instance formuláře. Třída Form6 projektu GUI ale není známa, proto doplníme do References a pak do using jmenný prostor, ve kterém třída Form6 je (jmenný prostor GUI). Jako Start-up project nastavíme projekt Aplikace. Spuštěním ověříme funkčnost.
Poté formulář zobrazíme instanční metodou ShowDialog() (dosud byl zobrazen statickou metodou run třídy Application). Kompilátor ale hlásí, že tato metoda není známa (přestože třída Form6 známa je).
???
3) Vytvoříme projekt Knihovna typu Class Library, v ní třídu Kalkulacka a přesuneme tam metodu secti().
4) Ve formuláři voláme metodu secti z třídy Kalkulacka. Zjistíme ale, že třída Kalkulacka není ve formuláři známá. Co musíme udělat?
5) Při spuštění vidíme, že se zbytečně zobrazuje i černé okno konzoly.

Podívejte se v Průzkumníku do složky Bin\Debug jednotlivých projektů. Budou tam kromě vlastního přeloženého exe nebo dll souboru i dll soubory projektů, které má projekt v referencích. Překvapivě je ale v projektu Aplikace i soubor Knihovna.dll, přestože projekt Aplikace nemá Knihovnu v referencích. Proč?