Programovací styly

- Na příkladu programu simulujícího čítač si ukážeme programovací styly:
  - naivní
  - procedurální
  - objektově orientovaný
- Příklad komunikace programu:

```
Hodnota = 10
0) Konec
1) Zvětšit
2) Zmenšit
3) Nastavit
Vase volba: 1
Hodnota = 11
0) Konec
1) Zvětšit
2) Zmenšit
3) Nastavit
...
```

Naivní styl

- Jednoduché úlohy lze řešit přímočaře:

```
package alg11;
import sugar.Sys;
public class Citac1 {
  final static int pocHodn = 0;
  static int hodn, volba;
  public static void main(String[] args) {
    hodn = pocHodn;
    do {
      Sys.pln("Hodnota = "+hodn);
      Sys.pln("0) Konec\n1) Zvětšit\n2) Zmenšit\n3) Nastavit");
      Sys.p("Vaše volba: ");
      volba = Sys.readInt();
      switch (volba) {
        case 0: break;
        case 1: hodn++; break;
        case 2: hodn--; break;
        case 3: hodn = pocHodn; break;
        default: Sys.pln("nedovolená volba");
      }
    } while (volba>0);
    Sys.pln("Konec");
  }
}
```

Procedurální styl

- Připomeňme hlavní zásady:
  - Zadaný problém se snažíme rozložit na podproblémy
  - Pro řešení podproblémů zavádíme abstraktní příkazy, které nejprve specifikujeme a pak realizujeme pomocí procedur a funkcí
- Aplikace na čítač

  Dílčí podproblémy:
  - komunikace s uživatelem, jejímž výsledkem je kód požadované operace
  - provedení požadované operace s čítačem
- Řešení:

```
public class Citac2 {
  final static int pocHodn = 0;
  static int hodn;

  static void operace(int op) {
    switch (op) {
      case 1: hodn++; break;
      case 2: hodn--; break;
      case 3: hodn = pocHodn; break;
    }
  }
```

Procedurální styl

```
static int menu() {
   int volba;
   do {
     Sys.pln("0. Konec");
     Sys.pln("1. Zvětšit");
     Sys.pln("2. Zmenšit");
     Sys.pln("3. Nastavit");
     Sys.p("Vaše volba: ");
     volba = Sys.readInt();
     if (volba<0 || volba >3) {
       Sys.pln("nedovolená volba");
       volba = -1;
     }
   } while (volba<0);
   return volba;
 }
```

Procedurální styl

```
public static void main(String[] args) {
   int volba;
   hodn = pocHodn;
   do {
     Sys.pln("hodnota = "+hodn);
     volba = menu();
     if (volba>0) operace(volba);
   } while (volba>0);
   Sys.pln("Konec");
 }
```


- Poznámky:
  - procedurální řešení čítače (jediná procedura pro všechny operace + globální proměnná pro hodnotu + globální konstanta udávající počáteční hodnotu) je nedokonalé (proč?)
  - čítač je typ, pro který jsou definovány určité operace (realizovatelné procedurami a funkcemi) a jehož implementace (proměnná pro hodnotu) by neměla být volně přístupná
  - pro realizaci čítače je vhodnější objektově orientovaný styl

Objektově orientovaný styl

- Hlavní zásady:
  - Při rozkladu problému specifikujeme nové datové typy (datové abstrakce), tzn. specifikujeme operace, které se budou s daty těchto typů provádět
  - Pro realizaci datových abstrakcí použijeme objekty a třídy
- Připomeňme si charakteristiku objektu:
  - objekt se může nacházet v různých stavech daných hodnotami datových položek (atributů) objektu
  - chování objektu je dáno metodami (operacemi), které jsou pro něj definovány
  - datové položky a metody objektu jsou dány typem objektu - třídou

Čítač jako datový typ

- Čítač zavedeme jako datový typ s operacemi
  zvetsit, *zmensit, nastavit* a *hodnota*
  a s datovými položkami
  hodn a *pocHodn*
- Grafické vyjádření:

- Poznámka: hodnota položky *pocHodn* bude stanovena při vytvoření objektu

Třída Čítač

```
public class Citac {
  private int hodn;
  private int pocHodn;

  public Citac(int ph) {
    pocHodn = ph; hodn = ph;
  }

  public void zvetsit() {
    hodn++;
  }
  public void zmensit() {
    hodn--;
  }
  public void nastavit() {
    hodn = pocHodn;
  }
  public int hodnota() {
    return hodn;
  }
}
```

položky objektu (instanční proměnné)

konstruktor

instanční metody

Použití čítače jako objektu

```
public class Citac3 {
  static int menu() { ... }

  public static void main(String[] args) {
    int volba;
    Citac citac = new Citac(0);
    do {
      Sys.pln("Hodnota ="+citac.hodnota());
      volba = menu();
      switch (volba) {
        case 1: citac.zvetsit(); break;
        case 2: citac.zmensit(); break;
        case 3: citac.nastavit(); break;
      }
    } while (volba>0);
    Sys.pln("Konec");
  }
}
```

Třídy a objekty

- Třída popisující nový datový typ obsahuje:
  - deklarace položek, ze kterých se skládají objekty daného typu
  - deklaraci konstruktoru, kterým se inicializují vytvořené objekty
  - deklarace metod, které realizují operace s objekty
- Položky se obvykle deklarují tak, aby nebyly z vnějšku objektu přístupné:
  `private` *`typ jméno`*`;`
- Konstruktor je inicializační operace, která má jméno třídy a nevrací žádnou hodnotu; schéma deklarace konstruktoru třídy *T*:

```
public T(specifikace parametrů) {
  tělo konstruktoru
}
```

- Ve třídě může být deklarováno několik konstruktorů, které se liší počtem a/nebo typy parametrů (přetěžování)
- Obvyklé schéma deklarace metody:

```
public typ jméno(specifikace parametrů) {
  tělo metody
}
```

Přístup k položkám objektu

- V metodě (konstruktoru) jsou položky objektu, na který se metoda (konstruktor) aplikuje, přímo přístupné prostřednictvím svých jmen

```
public class Citac {
  private int hodn;
  private int pocHodn;
...
public void zvetsit() {
  hodn++;  // zvětší se hodnota položky objektu, na který
           // se metody aplikuje
}
```

- Chceme-li zdůraznit, že se jedná o položku objektu, můžeme (někdy musíme) před její jméno napsat *this.*

```
public void zvetsit() { this.hodn++; }
```

- Položka objektu může být deklarována jako *public*; pak je zvenčí objektu přístupná pomocí operátoru "." a svého jména

```
public class Complex {
  public double re, im;
}
...
Complex x = new Complex();
x.re = 1.25; x.im = 5.3;
```

Třída Complex

- Podívejme se na deklaraci třídy *Complex* v balíku *suggar*

```
package sugar;

public class Complex {
  public double re;
  public double im;

  public Complex() {re=0; im=0;}
  public Complex(double r) {re=r; im=0);}
  public Complex(double r, double i) {re=r; im=i;}

  public double abs() {
    return Math.sqrt(re*re+im*im);
  }

  public Complex plus(Complex c) {
    return new Complex(re+c.re, im+c.im);
  }

  public Complex minus(Complex c) {
    return new Complex(re-c.re, im-c.im);
  }

  public String toString() {
    return "["+re+", "+im+"]";
  }
```