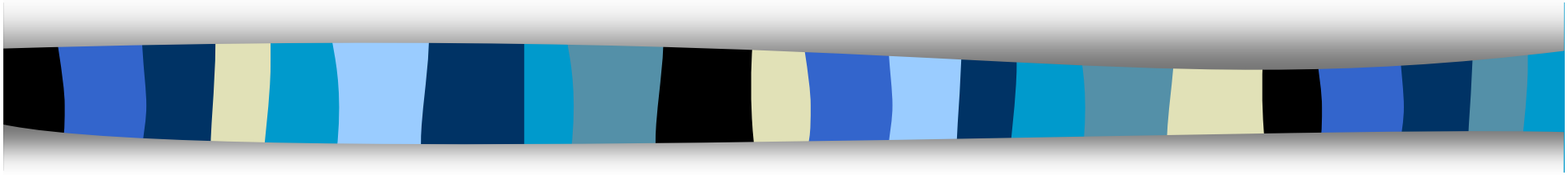
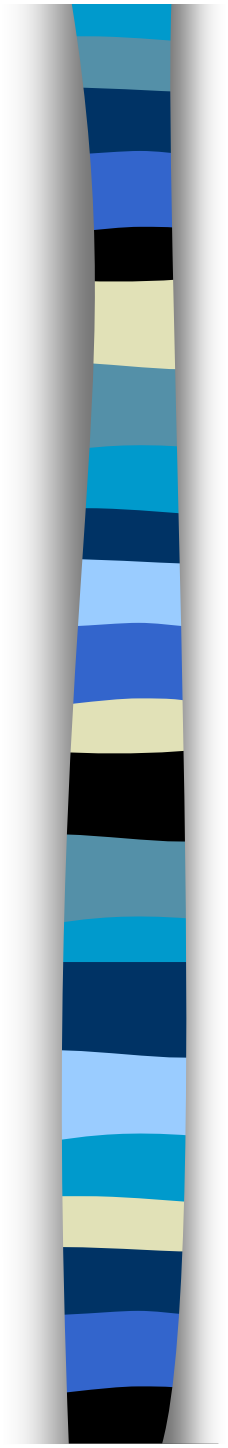
Návrh podle kontraktu - klasická metodika a moderní nástroje

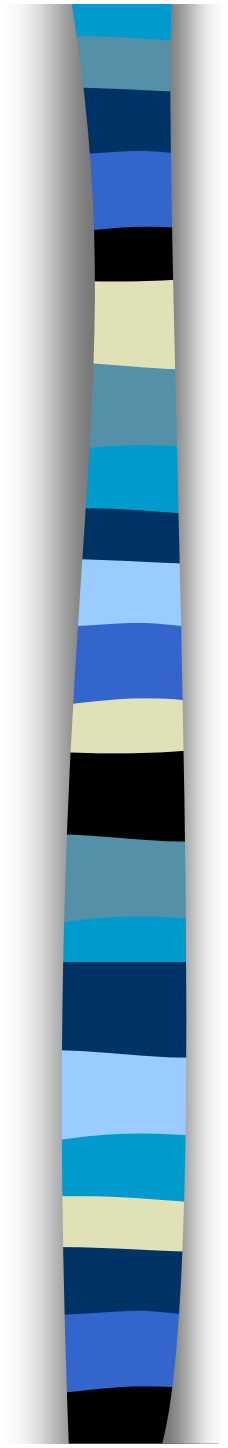


Tomáš Pitner
Masarykova univerzita v Brně
Fakulta informatiky
tomp@fi.muni.cz



Osnova

- Co je návrh podle kontaktu
- Vznik a vývoj metodiky a nástrojů
- Základní principy a podmínky
- Dostupné nástroje a příklad použití
- Srovnání s nástroji testování jednotek
- Shrnutí, informační zdroje



Co je návrh podle kontaktu

- **Návrh podle kontraktu** (Design by Contract, DbC) je delší dobu známý, ale stále nedoceněný princip tvorby spolehlivých programů.
- Abstraktní **specifikace chování** komponent SW pomocí **podmínek** a
- buďto nástroje na **běhové ověřování** jejich platnosti nebo
- jejich **formální důkaz**

Co je „dobrý program“

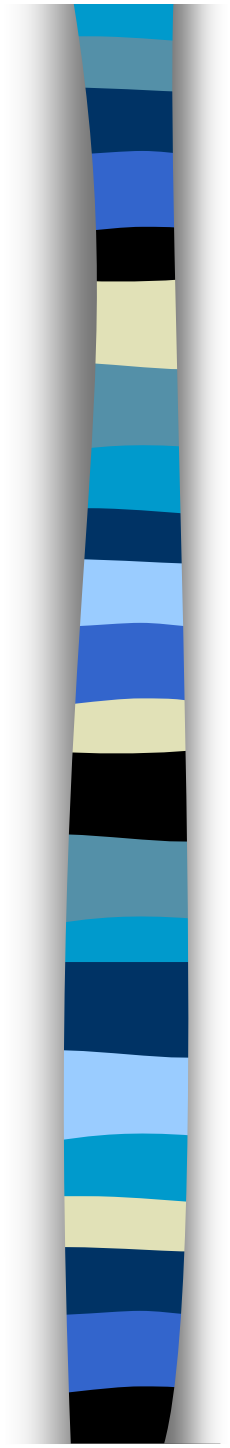
- Požadavky na SW lze členit na funkční a mimofunkční.
- K základním požadavkům patří:
  - **korektnost** (splnění požadavků, tj. specifikace programem)
  - **robustnost** (odolnost vůči nesplnění požadavků zvnějšku - uživatelem)

Vznik a vývoj návrhu podle kontraktu

- *Tony Hoare* (1969) zavedl tzv. **Hoareho kalkul** na zachycení chování programu pomocí **vstupních a výstupních podmínek**.
- *Barbara Liskow* v 70. letech vytvořila jazyk **CLU**, který je nativně podporuje.
- *Bertrand Meyer* (ETH Zürich) zavedl metodiku **DbC** a použil ji v jazyce **Eiffel**

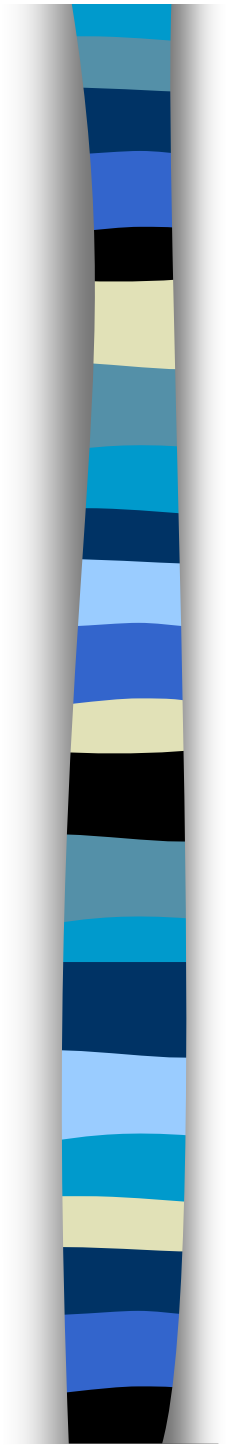
Základní princip návrhu podle kontraktu

- „Smlouva“ mezi programem a jeho uživatelem (tím může být i jiný program):
  - jestliže uživatel dodrží **vstupní** podmínku
  - program pak dodrží **výstupní** podmínku
- V objektovém prostředí lze podmínky zjemňovat, přidávat další.



Podmínky v kontraktu - vnější

- Vstupní podmínka (`precondition`)
  - tu musí splnit uživatel, resp. volající kód
- Výstupní podmínka (`postcondition`)
  - tu musí volaný kód, pokud je splněna vstupní podmínka

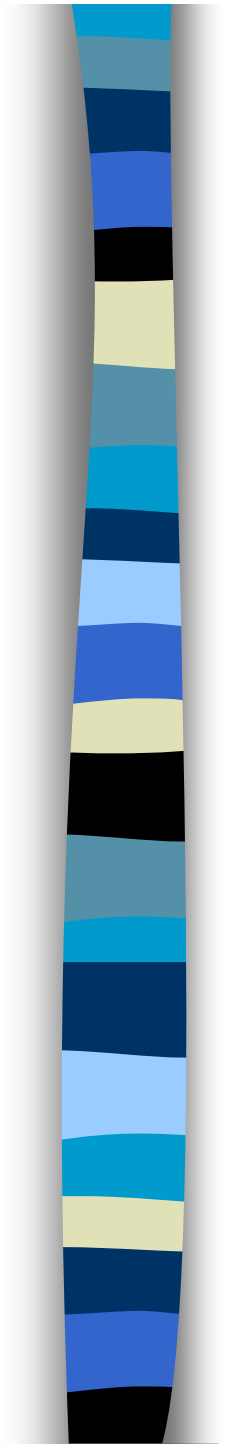


Podmínky v kontraktu - vnitřní

- Invariant objektu (`invariant`)
  - podmínka platící na objektu stále, přesněji mezi jednotlivými voláními metod objektů
- Invariant cyklu (`invariant`)
  - podmínka platící v daném místě cyklu stále
- Invariant cyklu (`variant`)
  - podmínka zaručující ukončení cyklu

Podmínky v OO prostředí

- Podmínky definované až na **třídách** (implementacích)
- Podmínky definované už na **rozhraních**
  - nástroje je pak uplatní i ve třídách implementujících tato rozhraní

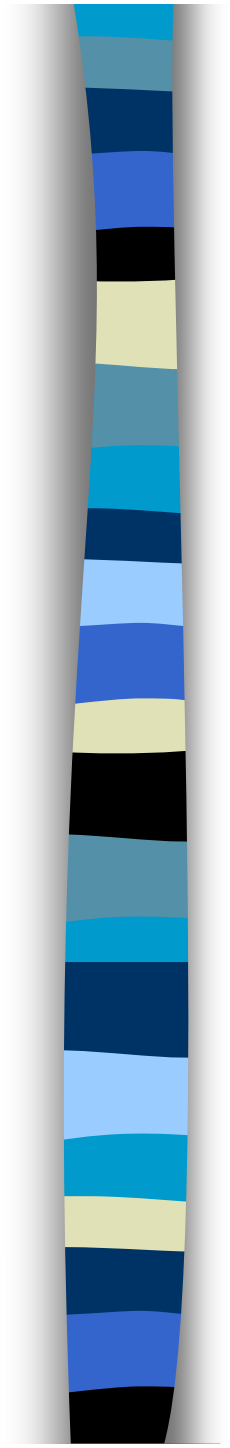


Nástroje pro návrh podle kontraktu

- Omezíme se na prostředí Java:
  - profesionálně používané
  - velmi rozšířené (na rozdíl od jazyka Eiffel...)
  - Java slouží i jako referenční jazyk

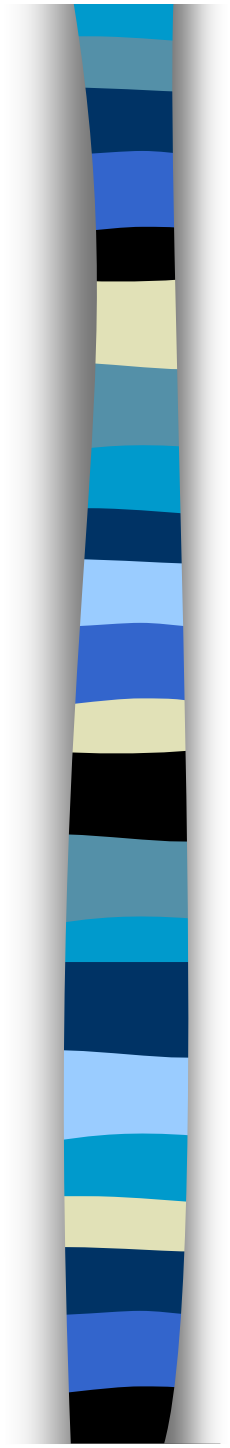
Nástroje pro návrh podle kontraktu - v Javě

- Volně (zdarma) dostupné:
  - **jass**
  - iContract
  - JContract
  - Contract4J (Java 5)
- Komerční:
  - JMSAssert™



JASS - Java with ASSertions

- Volně dostupný nástroj pro podporu DbC v Javě, tj. specifikaci a běhové hlídání:
  - vstupních a výstupních podmínek
  - invariantů objektů a cyklů, variantů cyklů



Příklad DbC - ADT zásobník

- Ukážeme použití *jass* při implementaci abstraktního datového typu zásobník (třída `Stack`), využijeme:
  - `require` - vstupní podmínka
  - `ensure` - výstupní podmínka
  - `invariant` - invariant cyklu
  - `invariant` - invariant objektu

Podmínky *require, ensure*

```
public Stack(int capacity) {
/** require capacity > 0; **/
  storage = new Object[capacity];
/** ensure [created_empty] isEmpty(); **/
}
```

- `require` - vstupní podmínka
- `ensure` - výstupní podmínka

Podmínka *ensure changeonly*

```
public Stack push(Object o) {
/** require [valid_object] o!=null;
            [not_full] !isFull(); **/
  top++;
  storage[top] = o;
  return this;
/** ensure changeonly{top,storage};
    [nonempty_after_push] !isEmpty();
    [increments_top] Old.top==top-1; **/
}
```

- `ensure changeonly{top,storage}`
  - zajistí, že metoda změnila jen uvedené proměnné

V cyklu - *invariant, variant*

```
public boolean contains(Object o) {
   ...
  while (i<=top &&!storage[i].equals(o))
/** invariant 0 <= i && i <= top + 1; **/
/** variant top + 1 - i **/
  {
      i++;
  }
  return i <= top;
}
```

- **`invariant`** - musí v tomto místě platit stále
- **`variant`** - hodnota výrazu se musí snižovat

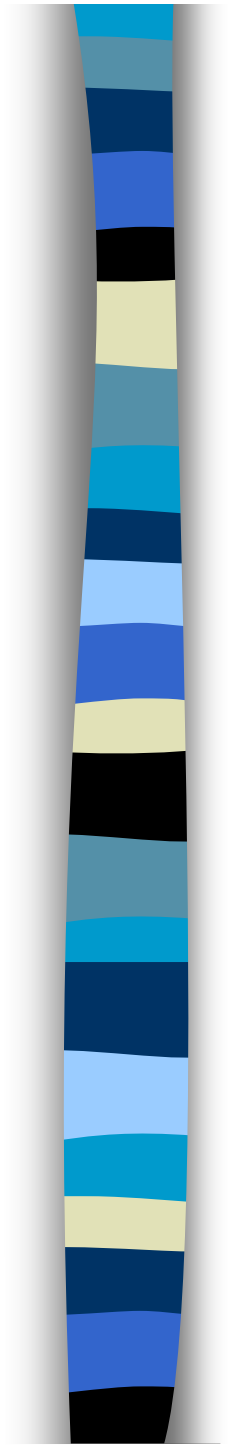
Na objektu - *invariant*

```
/** invariant
[range] -1<=top && top<storage.length;

[valid_content] storage.length == 0||
  (forall i:{0 .. top}#storage[i]!=null);

[LIFO] isFull() ||
  push("neco").pop().equals("neco");
**/
```

- **`invariant`** - musí na objektu platit stále
  - zajímavost: ve výrazu je použit *obecný kvantifikátor* (`forall`)



Fáze práce s jass

- Předkompilace zdroje nástrojem *jass*:

```
java -cp jass.jar;. jass.Jass Stack.jass
```

- Kompilace výsledku pomocí *javac*:

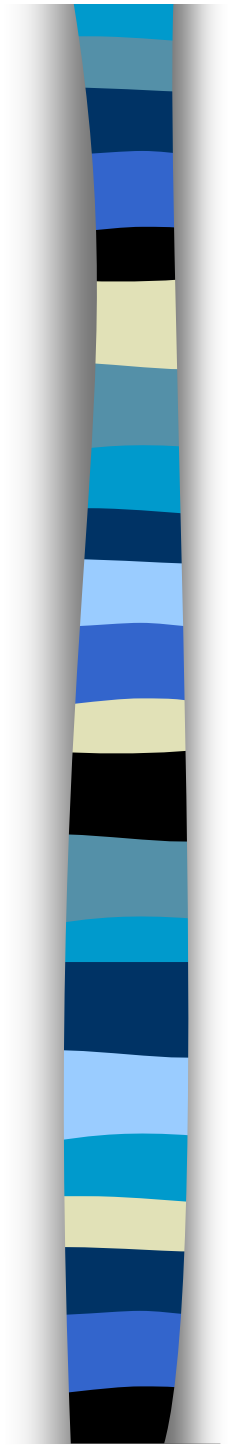
```
javac -classpath jass.jar;. Stack.java
```

- Spuštění třídy demotřídy `StackDemo`:

```
java -cp jass.jar;. StackDemo
```

Testování jednotek

- Dnes velmi populární - zejména v Javě
- a to v souvislosti s metodikou „Extrémní programování“.
- Umožňují prověřit chování jednotek programu (tříd, balíků) „*zvnějšku*“ - „blackbox testing“.
- **Doplňuje se tedy s DbC** - ten hlídá korektnost „*zevnitř*“ - „whitebox testing“.



Testování jednotek - nástroj *junit*

- `junit`: javové prostředí pro testování jednotek, tj.:
  - psaní a spouštění testů jednotek v Javě
- Volně (zdarma) dostupné na
  - `http://junit.org`

junit test zásobníku

```
public class StackTest extends
                junit.framework.TestCase {
  public void setUp() {
      ...
  }
  public void testEmptyAfterCreation() {
    assertTrue("Stack should be empty.",
                  st.isEmpty());
  }
  public void tearDown() {
      ...
  }
}
```

Metody v testech *junit*

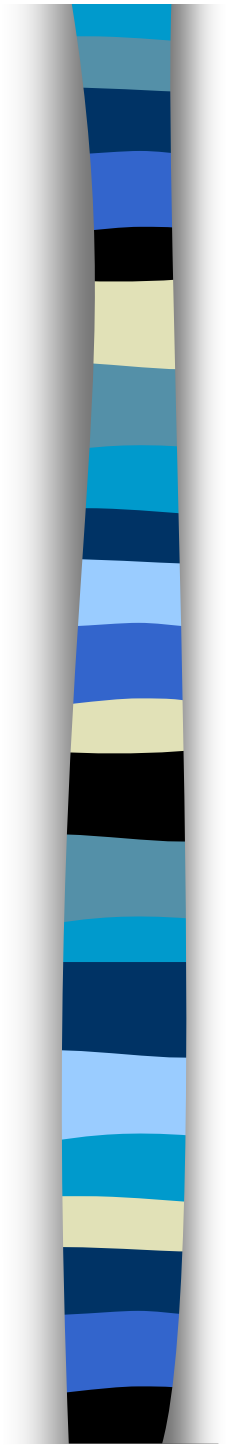
- `setUp()`: nastavení prostředí testu
- `test`*`NazevTestu`*`()`:
  - vlastní testovací metody
- `tearDown()`: úklid prostředí po testu

Test zásobníku - spuštění

```
java -cp jass.jar;. jass.Jass Stack.jass

javac -classpath junit.jar;jass.jar;.
  Stack.java StackTest.java

java -cp junit.jar;jass.jar;.
  junit.swingui.TestRunner StackTest
```



Shrnutí: DbC + testy jednotek

- Nesrovnatelně populárnější je testování jednotek - podporuje i řada IDE
- Lze kombinovat s DbC
- Ani jedno nezpůsobí zhoršení běhových vlastností (rychlost, náročnost)
- ...
- **Tak proč to nepoužívat...!?**