Dobře fungující objekty

Založeno na originální prezentace ke Kapitole 7
„Well-behaved objects“ z učebnice
Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Hlavní pojmy

- Testování
- Ladění
- Automatizace testování
- Psaní udržovatelných programů

Kousek kódu na tento den

```
public void test()
{
	int sum = 1;

	for (int i = 0; i <= 4; i++);
	{
		sum = sum + 1;
	}

	System.out.println("The result is: " + sum);
	System.out.println("Double result: " + sum+sum);
}
```

Jaký bude výstup?

Možné výsledky

```
The result is: 5
The result is: 6
The result is: 11
The result is: 2

Double result: 12
Double result: 4
Double result: 22
Double result: 66
```

Který z nich bude vytištěn?

The result is: 2
Double result: 22

Kousek kódu na tento den

```
public void test()
{
	int sum = 1;

	for (int i = 0; i <= 4; i++);
	{
		sum = sum + 1;
	}

	System.out.println("The result is: " + sum);
	System.out.println("Double result: " + sum + sum);
}
```

Musíme se zabývat chybami

- Časné chyby jsou obvykle *syntaktické chyby*.
  - Tyto chyby objeví kompilátor.
- Pozdější chyby jsou obvykle *logické chyby*.
  - S těmi nám nemůže pomoci kompilátor.
  - Rovněž jsou známy jako „bugs“ (štěnice, obtížný hmyz).
- Některé logické chyby nemají okamžitý zřejmý projev.
  - Komerční software je zřídka bez chyb.

Prevence versus detekce (Vývojář versus udržovatel)

- Můžeme snížit pravděpodobnost chyb.
  - Používejte techniky softwarového inženýrství jako je zapouzdření.
- Můžeme zvýšit šance odhalení chyb.
  - Používejte obvyklé postupy softwarového inženýrství, například modularizaci a dokumentaci.
- Můžeme rozvíjet schopnosti vyhledávat chyby.
  - Komplexní pohled na problém.

Testování a ladění

- Toto jsou rozhodující dovednosti.
- Testováním hledáme přítomnost chyb.
- Laděním hledáme příčinu chyb.
  - Chyba se může plně projevit v určité „vzdálenosti“ od svého zdroje.

Techniky testování a ladění

- Testování jednotek (v NetBeans)
- Automatizace testování
- Ruční procházení
- Ladící tisky
- Debuggery

Testování jednotek

- Každá jednotka aplikace může být testována.
  - Metoda, třída, modul(package v jazyce Java).
- Může (mělo by) se to provádět během vývoje.
  - Včasné nalezení a opravení chyb snižuje náklady na vývoj (např. programátorský čas).
- Je vytvářena sada testů.

Základy testování

- Je třeba porozumět tomu, co by měla jednotka dělat – porozumět jejímu *kontraktu*.
  - Hledáme porušení kontraktu.
  - Použití pozitivních i negativních testů.
- Testujte *okrajové případy*.
  - Prohledávání prázdné kolekce.
  - Přidání do plné kolekce.
  - Zahrnuje i případy nejen těsně *před*, ale i *za hranicemi*.

Dobře fungující objekty

Automatizace testování

Hlavní pojmy

- Testování jednotek
- Knihovna JUnit
- Opakované testování
- Testovací třídy
- Testové případy
- Aserce (tvrzení)
- Testovací přípravky

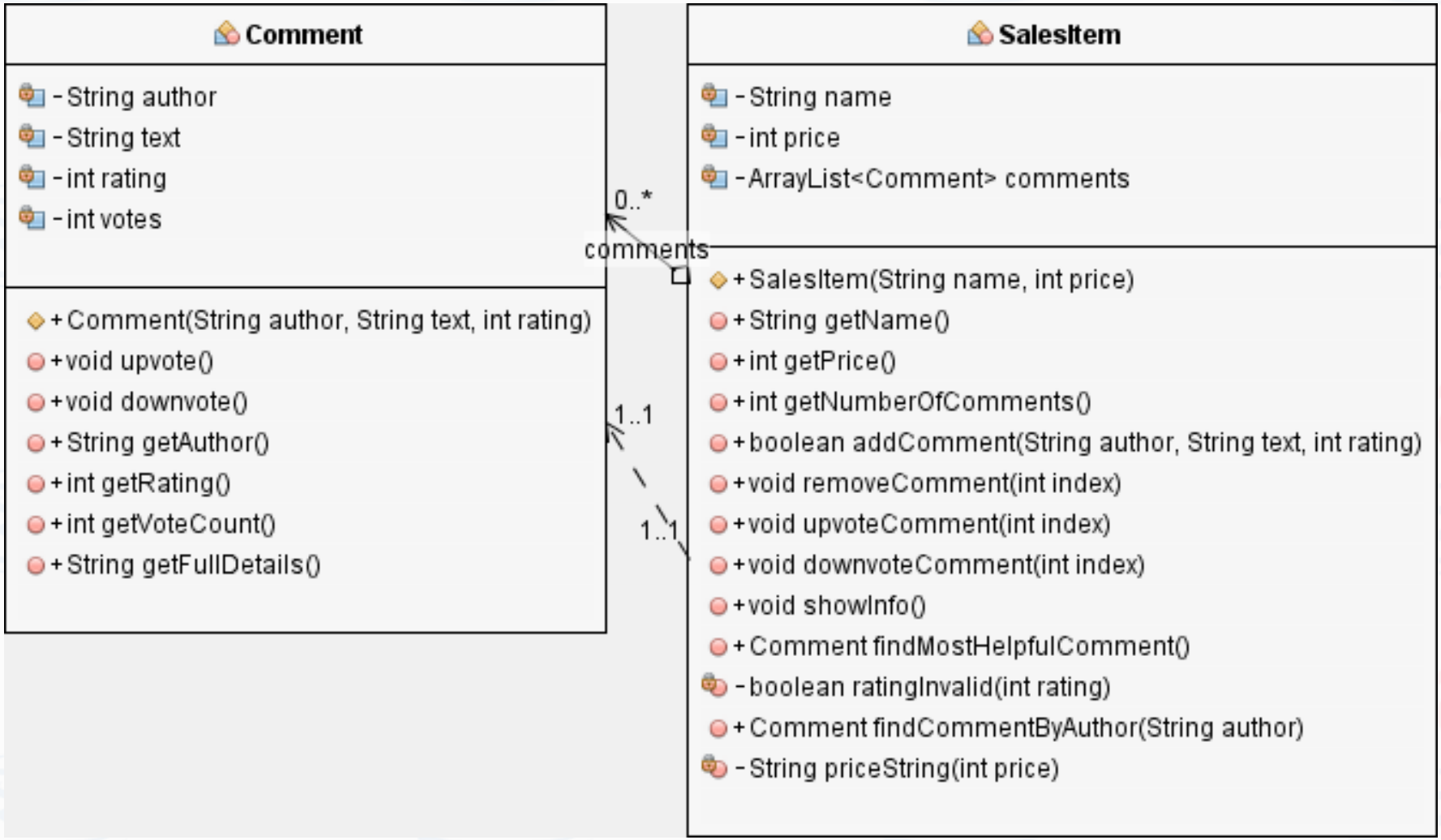
Testování jednotek

- Je možné vytvořit instance jednotlivých tříd.
- Jednotlivé metody mohou být zavolány.
- Inspektoři umožňují pohled na aktuální stav objektů.
- Seznámení s testováním prostřednictvím projektu *online-shop*.

Projekt *online-shop*

- Projekt *online-shop* představuje úvodní fázi vývoje softwaru pro online shop.
- Projekt obsahuje pouze dvě třídy:
  - SalesItem
  - Comment
- Funkcionalita se týká pouze komentářů zákazníků k nabízeným položkám.

Projekt *online-shop*



Projekt *online-shop*

- Položky lze vytvářet s uvedením popisu a ceny.
- Komentáře zákazníků lze přidat či odebrat z prodávané položky.
- Komentář obsahuje text komentáře, jméno autora komentáře a rating v intervalu <1;5>.
- Zákazník může vložit pouze jeden komentář.

Projekt *online-shop*

- Budoucí GUI aplikace bude obsahovat dotaz „Pomohl vám tento komentář?“ s možností se vyjádřit pomocí tlačítek „Ano“ a „Ne“.
- Informace o hodnocení komentářů bude uchovávána s tím, že komentář s nejvyšším hodnocením bude zobrazován na prvním místě.

Testování projektu *online-shop*

- Pro testování se zaměříme na třídu SalesItem a pokusíme se ověřit její funkcionalitu.
  - Je možné přidat a odebrat komentář?
  - Zobrazuje metoda showInfo správně všechny informace?
  - Jsou omezení (rating v intervalu<1;5> , pouze jeden komentář) správně prosazována?
  - Je správně vyhledáván komentář s nejvyšším hodnocením?

Automatizace testování

- Důkladné testování je *kreativní proces*, ale ...
- ... průběžné testování je časově náročné a opakující se.
- *Regresivní testování* zahrnuje opakovaný běh testů.
- Použití skupiny počítačů (*test rig)* nebo automatizovaného testovacího frameworku (*test harness)* může snížit některé náklady.

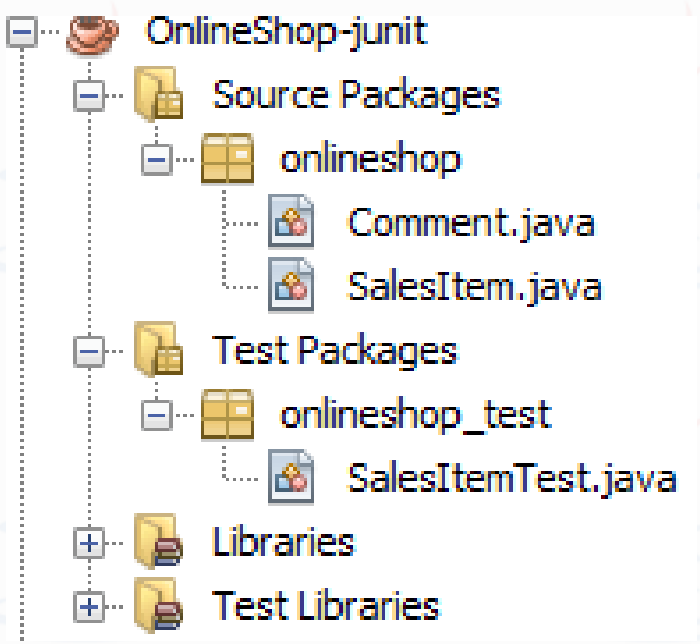
Test harness

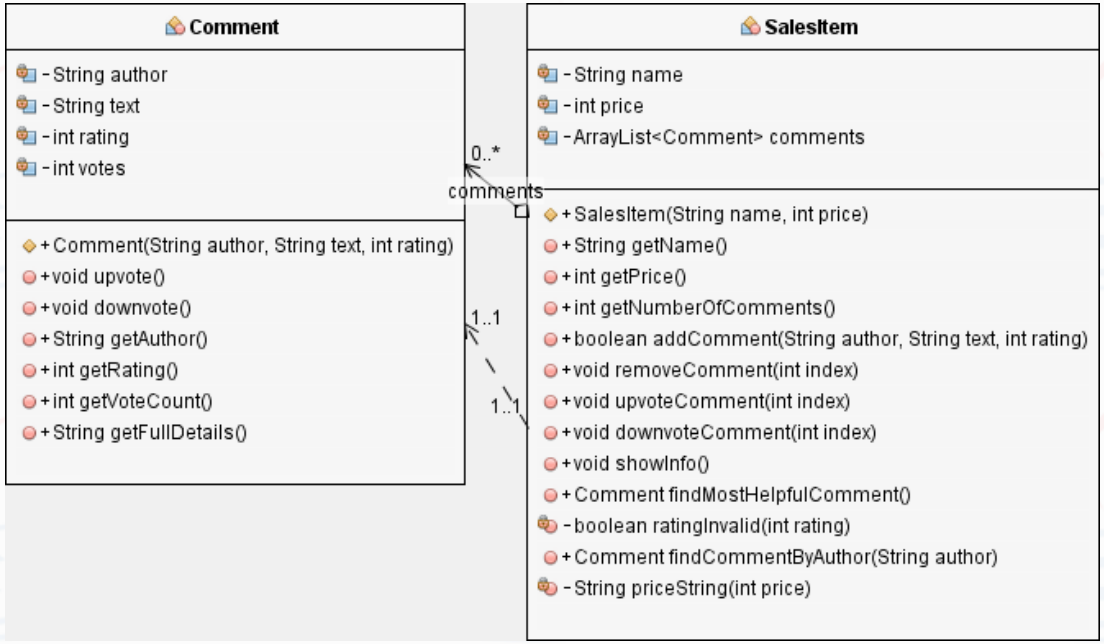
- Pro automatizaci testování se píší další testovací třídy.
- Objekty těchto tříd nahrazují lidskou interaktivitu.
- Pro vytvoření těchto testovacích tříd je zapotřebí tvořivosti a představivosti.
- Při přidávání funkcionality musí být testovací třídy *udržovány aktuální*.

Automatizace testování

- Pro podporu automatizace testování existují testovací frameworky.
- Automatizaci lze zkoumat prostřednictvím projektu *online-shop-junit*.
  - Intervence je požadovaná pouze pokud je ohlášeno selhání.

Projekt *online-shop-junit*





JUnit

- JUnit je testovací framework pro programy v jazyce Java.
- Testovací případy jsou metody, které obsahují testy.
- Testovací třídy obsahují testovací metody.
- Aserce (assert…) se používají k formulaci tvrzení o očekávaných výsledcích metod.
- Pro podporu opakovaného testování lze využít další metody a proměnné testovacích tříd.
- JUnit4 vs. JUnit5

Výběr testovací strategie

- Uvědomte si dostupné strategie.
- Vybírejte strategie odpovídající fázi vývoje.
- Automatizujte kdykoliv je to možné, protože automatizace
  - redukuje únavu.
  - redukuje lidské chyby.
  - umožňuje a podporuje opakované testování.

Dobře fungující objekty

Ladění

Testování a ladění

- Testováním hledáme přítomnost chyb.
- Laděním hledáme příčinu chyb.
  - Chyba se může plně projevit „dále“ od svého zdroje.

- Podpora ladění
  - Snížit pravděpodobnost chyb.
    - Použití technik SW inženýrství, např. zapouzdření.
    - Použití kritérií kvality návrhu - koheze a vazby.
  - Zvýšit šance vyhledání chyb.
    - Rozvíjet schopnosti vyhledávat chyby.
    - Použití technik SW inženýrství, např. modularizace a dobré dokumentace.

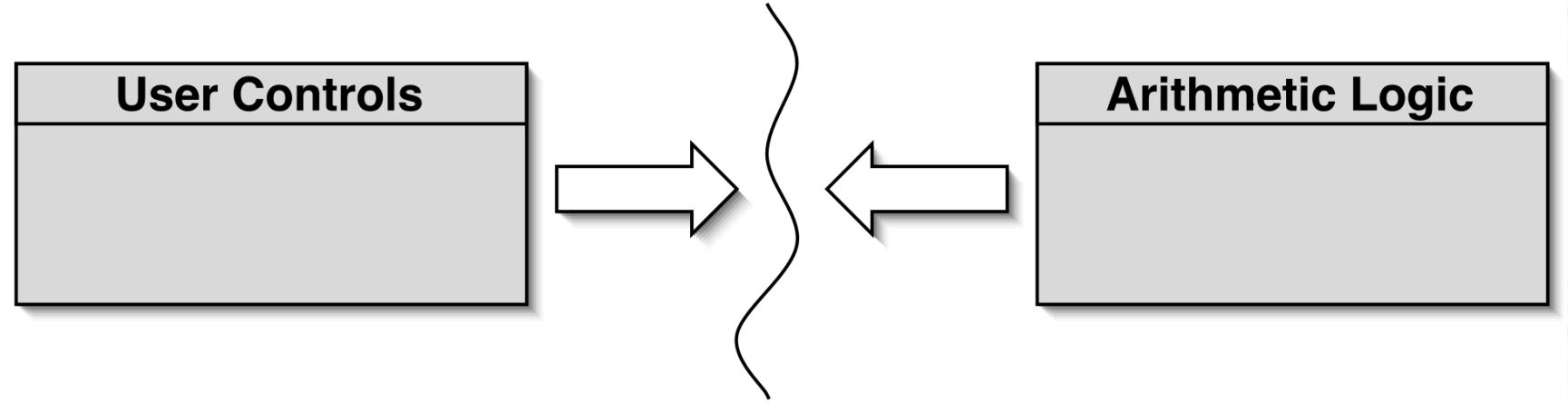
Ladění

- Je důležité rozvíjet schopnost čtení kódu.
  - Často ladění kódu, který tvořil někdo jiný.
- Techniky a nástroje pro podporu procesu ladění.
  - Ruční průchod
  - Ladicí tisky
  - Debuggery
- Možno zkoumat prostřednictvím projektu *calculator-engine*.

Modularizace a rozhraní

- Aplikace se často skládají z různých modulů.
  - Například tak, že na modulech mohou pracovat různé týmy.
- *Interfejs* mezi moduly musí být jasně specifikován.
  - Podporuje nezávislý paralelní vývoj.
  - Zvyšuje pravděpodobnost úspěšné integrace.

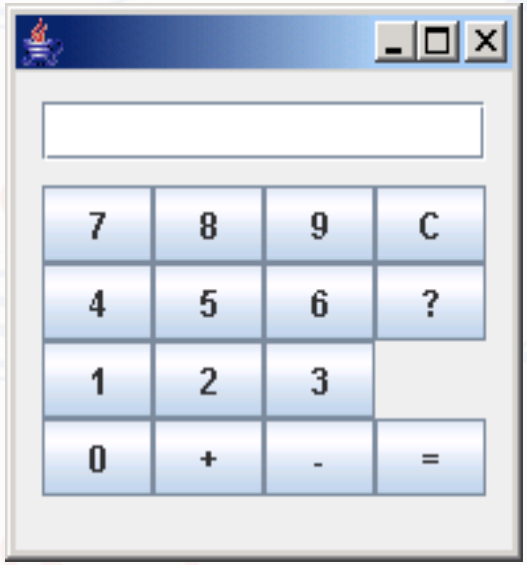
Modularizace u kalkulátoru



- Modul nepotřebuje znát implementační detaily druhého modulu.
  - Uživatelské ovládací prvky by mohly mít podobu GUI nebo hardware.
  - Logika by mohla být hardwarová nebo softwarová.

Hlavičky metod jako interfejs

```
// Vrací hodnotu pro zobrazení na displeji.
public int getDisplayValue();

// Volá se, když je stiknuto tlačítko s cifrou.
public void numberPressed(int number);

// Volá se, když je stisknut operátor +.
public void plus();

// Volá se, když je stisknut operátor -.
public void minus();

// Volá se pro ukončení výpočtu.
public void equals();

// Volá se pro resetování kalkulátoru.
public void clear();
```



Public versus private

Veřejné (public) prvky třídy (instanční proměnné, konstruktory, metody) jsou přístupné pro objekty ostatních tříd.
Instanční proměnné by neměly být veřejné (public).
Privátní (private) prvky jsou přístupné pouze objektům téže třídy.
Pouze metody, které jsou určeny pro ostatní třídy by měly být veřejné (public).

Ukrývání informací

- Data náležející jednomu objektu jsou skryta pro ostatní objekty.
- Víme, co objekt může dělat, ale nikoliv jak to dělá.
- Ukrývání informací zvyšuje úroveň nezávislosti.
- Nezávislost modulů je důležitá pro velké systémy a pro údržbu.

Techniky ladění

- Ruční provádění
- Tabelování stavu objektů
- Verbální průchod
- Ladící tisky
- Debuggery

Ruční provádění

- Relativně nevyužívané.
  - Tradiční přístup.
  - *Mocnější než jak je hodnoceno!*
- Odejděte od počítače!
- „Provádějte“ program ručně.
- Vysokoúrovňové (Krok) nebo nízkoúrovňové pohledy (Krok do).

Tabelování stavu objektu

- Chování objektu je velkou měrou určeno jeho stavem….
- …takže nesprávné chování je často důsledkem nesprávného stavu.
- Tabelujte hodnoty klíčových položek (fields).
- Dokumentujte změny stavu po každém volání metody.

Verbální průchod

- Vysvětlete někomu jinému, co kód dělá.
  - Může poukázat na chybu.
  - Proces vysvětlování vám může pomoci tak, že chybu odhalíte sami.
- Existují skupinové procesy pro řízení formálního provádění či *inspekcí*.

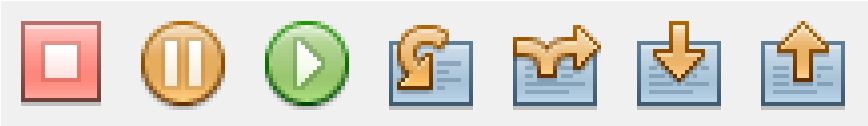
Ladicí tisky

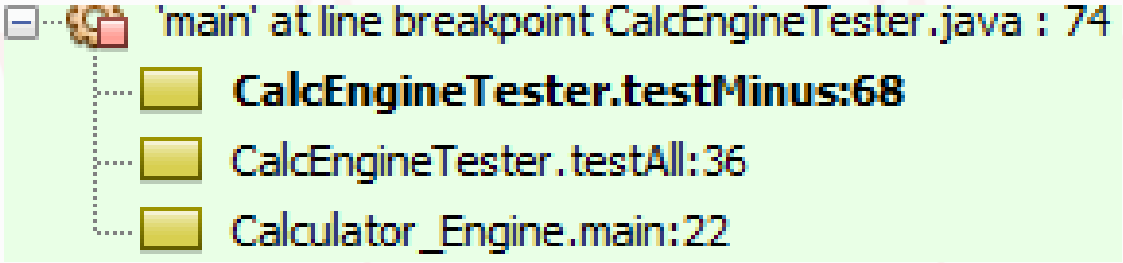
- Nejpopulárnější technika.
- Nevyžaduje speciální nástroje.
- Všechny programovací jazyky je podporují.
- Efektivní pouze v případě, že jsou použity ve „správných“ metodách.
- Výstup může být objemný!
- Vypínání a zapínání vyžaduje předběžnou rozvahu.

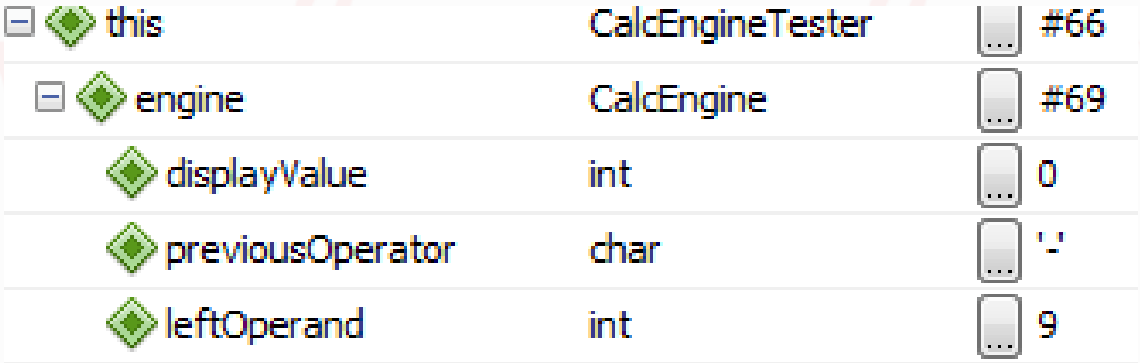
Debuggery

- Debugery jsou závislé na programovacích jazycích a na IDE.
  - NetBeans má integrovaný debugger.
- Podporuje body zastavení.
- Řízené provádění pomocí akcí Step a Step-into.
- Zobrazuje sekvence volání metod (zásobník).
- Zobrazuje stav objektů.

Netbeans




'main' at line breakpoint CalcEngineTester.java : 74
CalcEngineTester.testMinus:68
CalcEngineTester.testAll:36
Calculator_Engine.main:22


this
CalcEngineTester
#66
engine
CalcEngine
#69
displayValue
int
0
previousOperator
char
'-'
leftOperand
int
9

Přehled

- Chyby v programech jsou životní realita.
- Dobré techniky vývoje softwaru mohou snížit jejich výskyt.
- Dovednosti testovat a ladit jsou podstatné.
- Testování se musí státi zvykem.
- Automatizujte testování všude tam, kde je to možné.
- Neustále opakujte testy.
- Procvičujte řadu dovedností jak ladit programy.