# Testování jednotek, *JUnit*

## Testování software

- Účelem testování je obecně vzato zajistit bezchybný a spolehlivý software.
- Testování je naprosto *klíčová součást* SW vývoje.
- Je to rozsáhlá disciplína softwarového inženýrství sama o sobě.
- Některé postupy vývoje jsou přímo *řízené testy* (Test-driven Development).
- Zde v PB162 se zatím budeme věnovat jen *testování jednotek* programu.
- Každopádně se jedná o case-based testování, tj. testování na příkladech.
- Nejde tedy o formální ověření korektnosti (verifikaci).
- Bližší info v řadě předmětů na FI, např. PV260 Software Quality

## Typy testování

**Testování jednotek**

malé, ale ucelené kusy, např. třídy, samostatně testované — dělá vývojář nebo tester

**Integrační testování**

testování, jak se kus chová po zabudování do celku — dělá vývojář nebo tester často ve spolupráci s architektem řešení

**Akceptační testování**

při přijetí kódu zákazníkem — dělá přebírající

**Testování použitelnosti**

celá aplikace obsluhovaná uživatelem — dělá uživatel či specialista na UX

**Bezpečnostní testování**

zda neobsahuje bezpečnostní díry, odolnost proti útokům, robustnost — dělá specialista na bezpečnost

## Testování jednotek

- Testování jednotek (*unit testing*) testuje jednotlivé elementární části kódu, kde elementární části jsou *třídy a metody*.
- V Javě se nejčastěji používá volně dostupný balík JUnit.
- V nových produktech se používají verze JUnit 4.x nebo JUnit 5.
- Alternativně lze použít například AssertJ.
- Elementárním *testem* v JUnit je *testovací metoda* opatřena anotací `@Test`.

# Jednoduchý příklad `JUnit` testu

- `@Test` před metodou označí tuto metodu za *testovací*.
- Metoda se nemusí nijak speciálně jmenovat (žádné `testXXX` jako dříve není nutné).

```
@Test
public void minimalValueIs5() {
  Assert.assertEquals(5, Math.min(7, 5));
}
```

- Metoda `assertEquals` bere 2 parametry
  - očekávanou (expected) hodnotu — v příkladu `5` a
  - skutečnou (actual) hodnotu — v příkladu `Math.min(7, 5)`.
- Pokud hodnoty nejsou stejné, test selže.
- Může přitom vydat hlášku, co se vlastně stalo.

# Příklad Calculator — testovaná třída

- Testovaná třída `Calculator`, jednoduchá sčítačka čísel:

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}

new Calculator().evaluate("1+2"); // returns 3
```

Řetězec `"\\+"` je pouhý regulární výraz reprezentující znak `+`.

# Příklad Calculator — testovací třída

- Třída testující `Calculator`:

```
public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
```

```
    Assert.assertEquals(6, sum);
  }
}
```

Zdroj: junit wiki.

# Repl.it demo k JUnit testování

- https://repl.it/@tpitner/PB162-Java-Lecture-03-JUnit

# `assert` metody

- Jak bylo vidět, pro ověření, zda během provádějí testu platí různé podmínky, jsem používali volání `Assert.assertXXX`.
- Z jejich názvů je intuitivně patrné, co vlastně ověřují.
- Jsou realizovány jako statické metody třídy `Assert` z JUnit:
  - `assertTrue()`
  - `assertFalse()`
  - `assertNull()`
  - a další

# Import statických metod `Assert`

- Abychom si ušetřili psaní názvu třídy `Assert`, můžeme potřebné statické metody importovat

```
import static org.junit.Assert.assertEquals;
public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

Javové testování bude podrobně probíráno v dalším kurzu — PV168.