JAVA

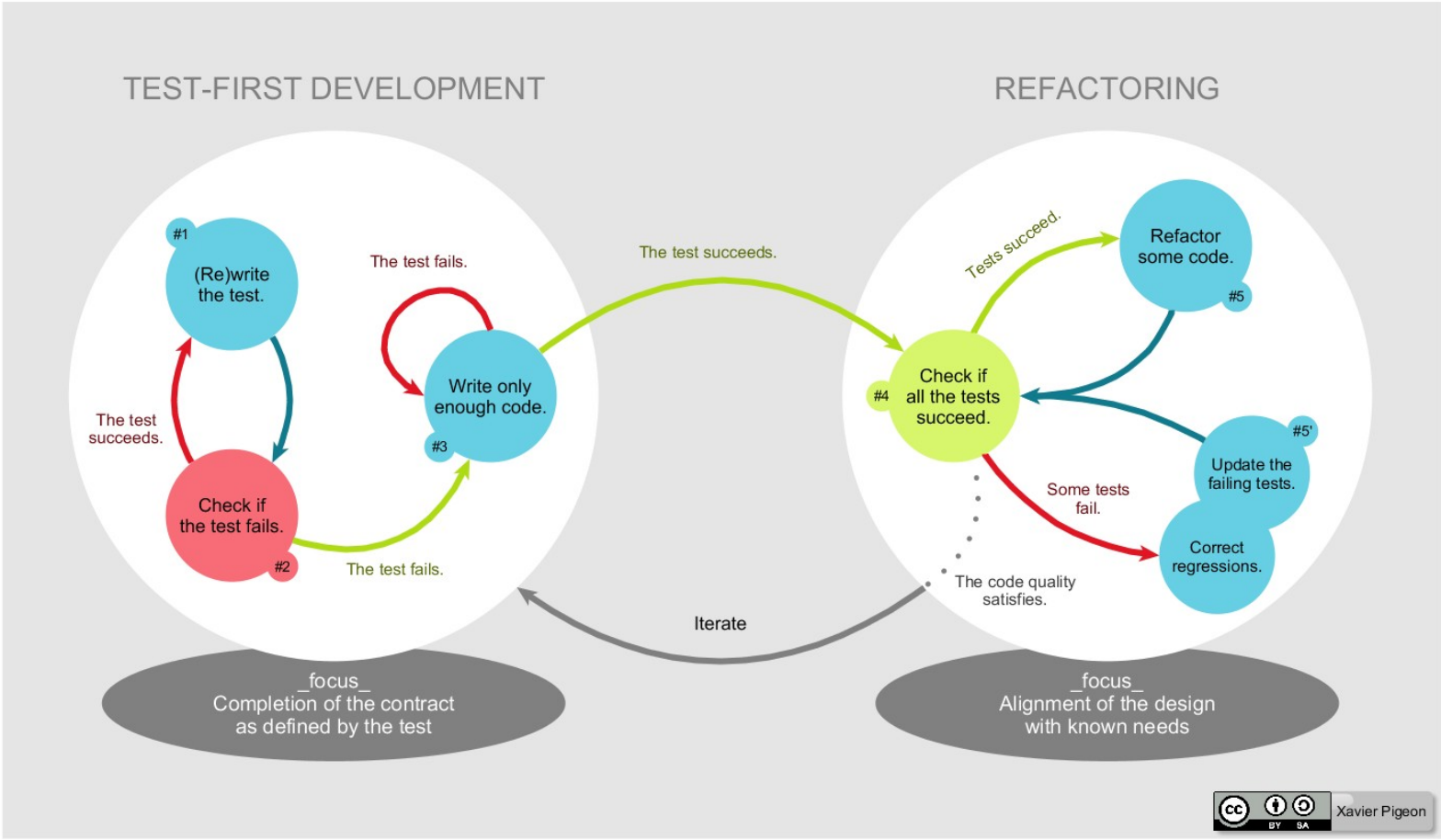
Unit testing

Úvod

- unit testing
  - testování „malý“ jednotek funkčnosti
  - jednotka – nezávislá na ostatních
    - testování zcela oddělené
    - vytvářejí se pomocné objekty pro testování
      - kontext
  - „typicky“ v OO jazycích
    - jednotka ~ metoda
  - ideálně – unit testy pro všechny jednotky v programu
    - „typicky“ v OO jazycích
      - pro všechny public metody

Test-driven development

- „začít s testy“



Zdroj: https://commons.wikimedia.org/wiki/File:TDD_Global_Lifecycle.png#/media/File:TDD_Global_Lifecycle.png

JUnit

- podpora pro unit testing v Javě
- http://www.junit.org/
- používání založeno na anotacích
  - starší verze založeny na dědičnosti a jmenných konvencích
- mírně odlišné použití u různých verzí
  - 5, 4, 3

Používání

- testovací metody označeny anotací @Test
- JUnit se spustí na množinu tříd
  - najde v nich metody označené @Test
  - ty provede

- další anotace
  - @BeforeEach (@Before)
    - metoda prováděná před každým testem
    - určeno pro přípravu „prostředí“
  - @AfterEach (@After)
    - metoda provádění po každém testu
    - určeno pro „úklid“
  - @BeforeAll (@BeforeClass)
    - metoda prováděná před všemi testy na dané třídě
  - @AfterAll (@AfterClass)
    - metoda prováděná po všech testech na dané třídě

Příklad

```
public class SimpleTest {

   private Collection collection;

   @BeforeAll
   public static void oneTimeSetUp() {
      // one-time initialization code
   }

   @AfterAll
   public static void oneTimeTearDown() {
     // one-time cleanup code
   }

   @BeforeEach
   public void setUp() {
     collection = new ArrayList();
   }

   @AfterEach
   public void tearDown() {
     collection.clear();
   }

   @Test
   public void testEmptyCollection() {
       assertTrue(collection.isEmpty());
   }

   @Test
   public void testOneItemCollection() {
       collection.add("itemA");
       assertEquals(1, collection.size());
   }
}
```

Assert

- assertTrue
- assertFalse
- assertEquals
- assert...
  - statické metody na org.junit.jupiter.api.Assertions (org.junit.Assert)
  - ověřování podmínek v testech
  - test selže pokud assert... selže
    - assert...() vyhodí AssertionError

- obecně
  - test je splněn, pokud metoda skončí normálně
  - test není splněn, pokud metoda vyhodí výjimku

Testování výjimek

- jak otestovat „správné“ vyhazování výjimek?

```
assertThrows(IndexOutOfBoundsException.class, () -> {
    new ArrayList<Object>().get(0);
});
```

- ve starších verzích

```
@Test(expected= IndexOutOfBoundsException.class) public void empty() {
    new ArrayList<Object>().get(0);
}
```

Spouštění testů

- z kódu

```
org.junit.runner.JUnitCore.runClasses(TestClass1.class,...);
```

- z příkazové řádky

```
java -jar junit.jar -select-class TestClass1
```

- z Antu
  - task junit

```
<junit printsummary="yes" fork="yes" haltonfailure="yes">
  <formatter type="plain"/>
  <test name="my.test.TestCase"/>
</junit>
```

- z Mavenu
  - mvn test
- z IDE

TestNG

- http://testng.org/
- inspirováno JUnit frameworkem
- mírně jiné vlastnosti
  - původně
  - dnes skoro stejné
- základní použití je stejné

Java

Reactive programming

Reactive programming (RP)

- datové toky a šíření změn v programu
  - změny v datech se automaticky propagují

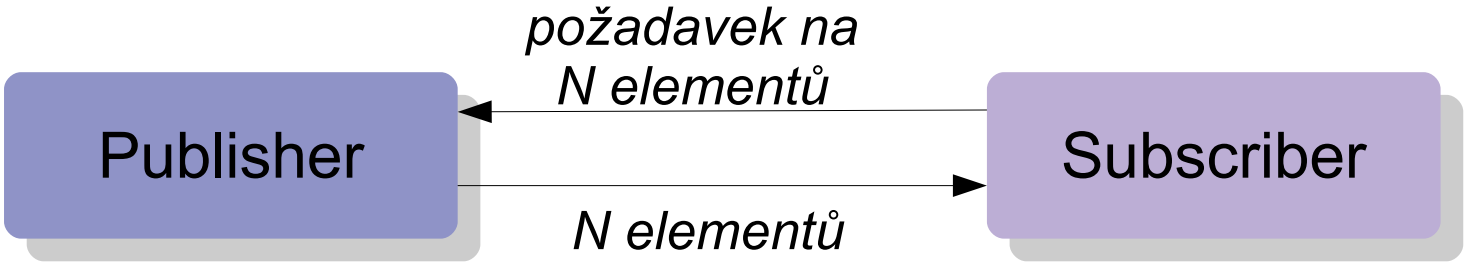
- publisher-subcriber
  - architektonický vzor
  - jeden z konkrétních modelů pro RP

  - publisher produkuje data
  - subsciber asynchronně data zpracovává
  - mezi P a S můžou být procesory transformující data

- proč RP
  - jednodušší kód, efektivnější, …
  - „rozšíření“ stream API

Publisher-Subsriber v Javě

- Flow API (Reactive streams)
- java.util.concurrent.Flow
  - od Java 9



- „kombinace iterátoru a Observer vzoru“

Flow API

```
@FunctionalInterface
public static interface Flow.Publisher<T> {
  public void subscribe(Flow.Subscriber<? super T> subscriber);
}

public static interface Flow.Subscriber<T> {
    public void onSubscribe(Flow.Subscription subscription);
    public void onNext(T item) ;
    public void onError(Throwable throwable) ;
    public void onComplete() ;
}

public static interface Flow.Subscription {
    public void request(long n);
    public void cancel() ;
}

public static interface Flow.Processor<T,R> extends
                        Flow.Subscriber<T>, Flow.Publisher<R> {
}
```

Flow API

- SubmissionPublisher
  - implementuje Publisher interface
  - asynchronně publikuje dodaná data

  - konstruktor bez parametrů
    - používá ForkJoinPool.commonPool()
  - další konstruktory – parametr pro exekutor

  - metody
    - subscribe(Flow.Subscriber<? super T> subscriber)
    - submit(T item)
    - ...

Observer pattern

- objekt (observer) „sleduje“ druhý objekt (observable) – pokud se druhý objekt změní, upozorní všechny „sledovatele“
  - java.util.Observer
  - java.util.Observable
    - pozor – od Java 9 jsou Deprecated (nahrazeny Flow)

- použití
  - UI
    - Observable – UI komponenty
    - Observer – reakce na UI události

Ještě k vláknům

ThreadLocal

- vlastní kopie pro každé vlákno
- obvykle se používají jako statické atributy

- metody

```
T get()
protected T initialValue()
void remove()
void set(T value)
static <S> ThreadLocal<S> withInitial(Supplier<?
                                        extends S> supplier)
```

JAVA

GUI
(in the std library)

Overview

- Java 1.0 – AWT
  - Abstract Window Toolkit
  - goal – the same good-looking GUI on all platforms
    - many limitations (e.g. 4 fonts only)
    - hard to use it
      - "non-object-based" approach
- Java 1.1
  - new event model
    - object-based approach
- Java 1.2
  - new GUI – Swing
    - a part of JFC (Java Foundation Classes)
- Java 8
  - JavaFX – new UI, exists since 2009 (had to be installed separately)
- Java 11
  - JavaFX removed from std lib
  - https://openjfx.io/

Swing

- packages
  - javax.swing....
  - uses also classes from java.awt...
  - many classes extends classes from java.awt...
- fully implemented in Java
  - the same look-and-feel on all platforms
    - look-and-feel can be modified – adjusted to a platform
- support for 2D graphics, printing, drag-and-drop, localization,...

Hello World

```
import javax.swing.*;

public class HelloWorldSwing {
  private static void createAndShowGUI() {
    JFrame.setDefaultLookAndFeelDecorated(true);
    JFrame frame = new JFrame("HelloWorldSwing");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JLabel label = new JLabel("Hello World");
    frame.getContentPane().add(label);
    frame.pack();
    frame.setVisible(true);
  }
  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
      public void run() {
        createAndShowGUI();
      }
    });
  }
}
```

Události

Observer pattern

- GUI se ovládá pomocí událostí (*events*)
  - př. stisknutí tlačítka → událost
- zpracování událostí – *listener*
  - objekt si zaregistruje *listener* → dostává informace o události
- mnoho druhů událostí (a jim odpovídající *listenery*)
  - př. stisk tlačítka, zavření okna, pohyb myši,...

```
public class ButtonAndLabel implements ActionListener {
  ...
  JButton button = new JButton("Click here");
  button.addActionListener(this);
  ...
  public void actionPerformed(ActionEvent e) {
    clicks++;
    label.setText("Hello World: "+clicks);
  }
```

Události

- jeden *listener* lze zaregistovat pro více událostí

```
public class TempConvert implements ActionListener {
   ...
     input = new JTextField();
     convertButton = new JButton("Preved");
     convertButton.addActionListener(this);
     input.addActionListener(this);
   ...
   public void actionPerformed(ActionEvent e) {
     int temp = (int)
   ((Double.parseDouble(input.getText())-32)*5/9);
     celLabel.setText(temp+" Celsius");
   }
}
```

Vlákna

- obsluha událostí a vykreslování GUI
  - **jedno** vlákno (event-dipatching thread)
  - zajišťuje postupné obsluhování událostí
    - každá událost se obslouží až po skončení předchozí obsluhy
    - události nepřerušují vykreslování
- SwingUtilities.invokeLater(Runnable doRun)
  - statická metoda
  - provede kód v doRun.run() pomocí vlákna obsluhujícího GUI
    - počká se, až budou všechny události obslouženy
  - metoda okamžitě skončí
    - nečeká, až se kód provede
  - používá se, pokud program upravuje GUI
- SwingUtilities.invokeAndWait(Runnable doRun)
  - jako invokeLater(), ale skončí, až se kód provede

GUI

- více v LS

Java

Pokračování...

Kam dál...

- **NPRG021 Pokročilé programování na platformě Java**
  - LS 2/2
  - obsah
    - GUI (Swing, JavaFX)
    - Reflection API, Classloaders, Security
    - Generické typy, anotace
    - RMI
    - JavaBeans
    - Java Enterprise Edition: EJB, Servlets, Java Server Pages, Spring,...
    - Java Micro Edition: Java pro mobilní a Embedded systémy, CLDC, MIDP, MEEP
    - RTSJ, Java APIs for XML, JDBC, JMX,…
    - Kotlin a další „Java“ jazyky
    - Android
  - částečně povinné pro **NPRG059 Praktikum z pokročilého objektového programování**
    - povinný předmět pro některé Mgr. okruhy