JAVA

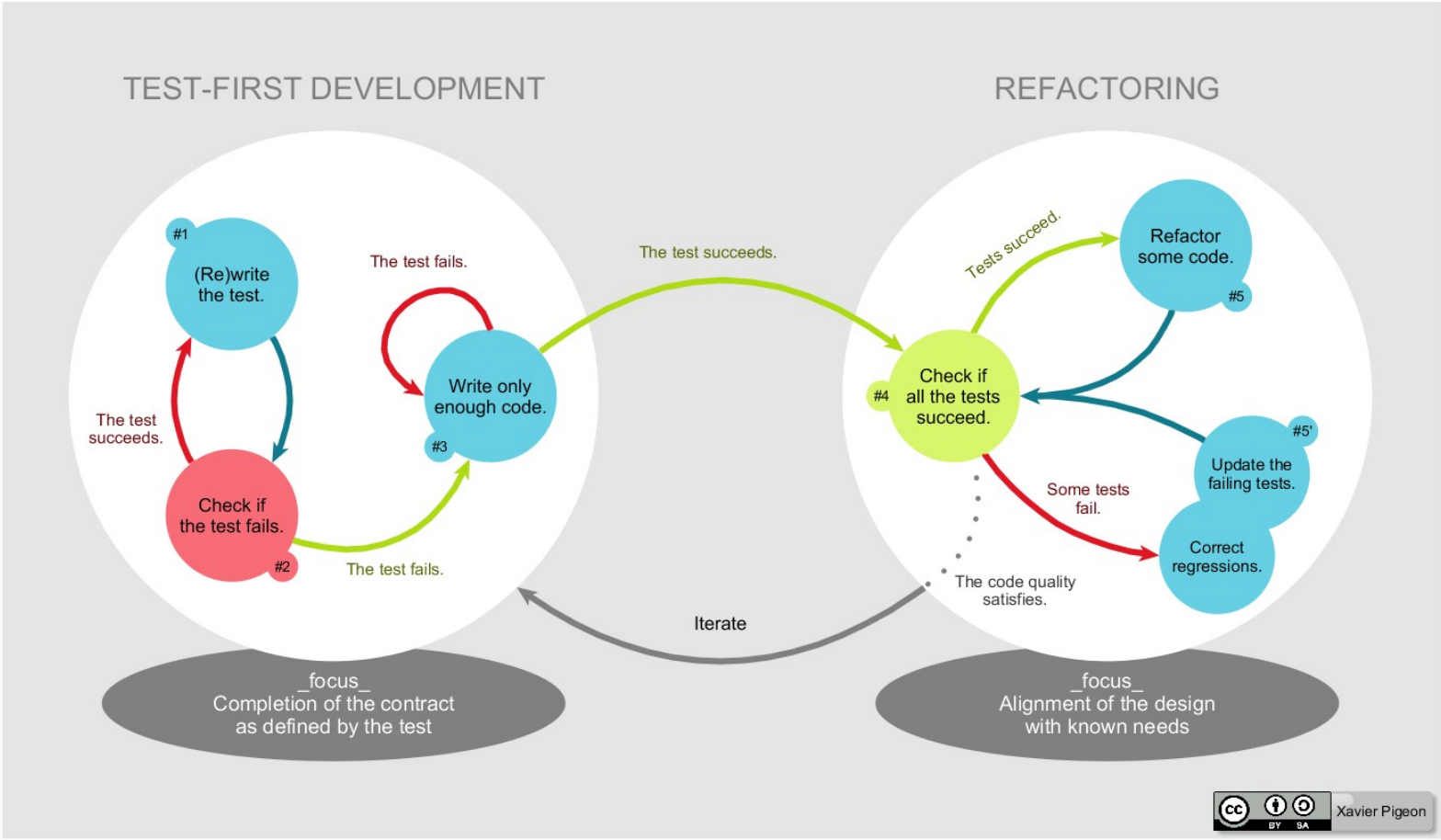
Unit testing

Introduction

- unit testing
  - testing “small” units of functionality
  - a unit – independent on other ones
    - tests are separated
    - creating helper objects for tests
      - context
  - typically in OO languages
    - unit ~ method
  - ideally – unit tests for all units in a program
    - typically in OO languages
      - for all public methods

Test-driven development

- tests first



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

JUnit

- support for unit testing in Java
- http://www.junit.org/
- usage based on annotations
  - older versions based on inheritance and naming conventions
- slightly different usage in different versions
  - 5, 4, 3

Usage

- test methods marked by the @Test annotation
- JUnit is run on a set of classes
  - searches in them all @Test methods
  - executes them

- other annotations
  - @BeforeEach (@Before)
    - a method run before each test
    - intended for “environment” preparation
  - @AfterEach (@After)
    - a method run after each test
    - intended for “cleaning”
  - @BeforeAll (@BeforeClass)
    - a method run before all tests in the given class
  - @AfterAll (@AfterClass)
    - a method run after all tests in the given class

Example

```
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...
  - static methods of org.junit.jupiter.api.Assertions (org.junit.Assert)
  - testing conditions in tests
  - test fails if assert... fails
    - assert...() throws AssertionError

- in general
  - test is successful if the method terminates regularly
  - test fails if the method throws an exception

Testing exceptions

- how to test “correctly” thrown exceptions?

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

- in older versions

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

Running tests

- from code

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

- from command line

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

- from Ant
  - the task junit

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

- from Maven
  - mvn test
- from IDE

TestNG

- http://testng.org/
- inspired by JUnit
- slightly different set of features
  - originally
  - now, more-or-less the same
- basic usage is the same

Java

Reactive programming

Reactive programming (RP)

- data streams and propagating of changes in a program
  - data changes are automatically propagated

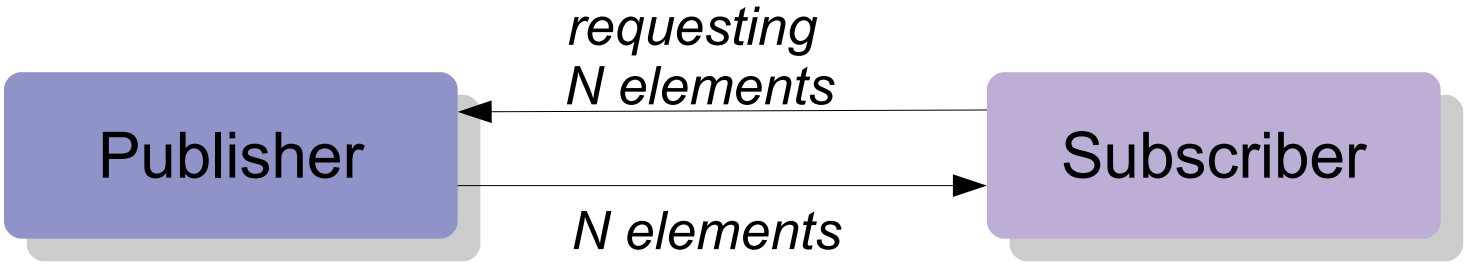
- publisher-subscriber
  - architectural pattern
  - one of particular models for RP

  - publisher publishes data
  - subscriber asynchronously data consumes
  - there can be processor between P and S transforming data

- why RP
  - simpler code, more efficient, …
  - “an extension” of the stream API

Publisher-Subsriber in Java

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



- „a combination of iterator and observer patterns“

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
  - implements the Publisher interface
  - asynchronously publishes given data
  - the constructor without parameters
    - uses ForkJoinPool.commonPool()
  - other constructors – an argument for an executor
  - methods
    - subscribe(Flow.Subscriber<? super T> subscriber)
    - submit(T item)
    - ...

Observer pattern

- an object (observer) „observes“ another object (observable) – if the other object changes, it notifies all its observers
  - java.util.Observer
  - java.util.Observable
    - warning – Deprecated since Java 9 (replaced by Flow)

- usage
  - UI
    - Observable – UI components
    - Observer – reactions to UI events

Java

More about threads

ThreadLocal

- own copy for each thread
- typically used as static fields

- methods

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

GUI
(v std knihovně)

Přehled

- Java 1.0 – AWT
  - Abstract Window Toolkit
  - cíl – na všech platformách dobře vypadající GUI
    - různá omezení (např. jen 4 fonty)
    - špatně se používalo
      - "ne-objektový" přístup
- Java 1.1
  - nový "event model"
    - objektový přístup
- Java 1.2
  - nové GUI – Swing
- Java 8
  - JavaFX – nové UI, existuje už od 2009 (bylo nutno nainstalovat samostatně)
- Java 11
  - JavaFX odstraněna z std. knihovny
  - https://openjfx.io/

Swing

- balíky
  - javax.swing....
  - používají se třídy z java.awt...
  - mnoho tříd je potomky tříd z java.awt...
- implementace plně v Javě
  - na všech platformách vypadá a chová se stejně
    - vzhled i chovaní lze upravovat – přizpůsobovat platformě
- podpora pro 2D grafiku, tisk, drag-and-drop, lokalizace, ...

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();
      }
    });
  }
}
```

Events

Observer pattern

- GUI is controlled through *events*
  - e.g. click on a button → event
- event processing – *listener*
  - an object registers a *listener* → receives info about events
- many types of events (and of corresponding *listeners*)
  - e.g. button click, window closing, mouse move,...

```
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);
  }
```

Events

- a single *listener* can be registered for multiple events

```
public class TempConvert implements ActionListener {
   ...
     input = new JTextField();
     convertButton = new JButton("Convert");
     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");
   }
}
```

Threads

- event processing and GUI painting
  - a **single** thread (event-dipatching thread)
  - ensures subsequent event processing
    - each event is processed after the previous one is finished
      - events do not interrupt painting
- SwingUtilities.invokeLater(Runnable doRun)
  - static method
  - runs code in doRun.run() using the event-processing thread
    - waits until all events are processed
  - the method ends immediately
    - does not wait till the code is run
  - used for GUI modifications
- SwingUtilities.invokeAndWait(Runnable doRun)
  - as invokeLater(), but ends after the code is run

GUI

- more in summer semester

Java

What next...

What next

- **NPRG021 Advanced programming for Java platform**
  - summer 2/2
  - synopsis
    - GUI (Swing, JavaFX)
    - Reflection API, Classloaders, Security
    - Generics, annotations
    - RMI
    - JavaBeans
    - Java Enterprise Edition: EJB, Servlets, Java Server Pages, Spring,...
    - Java Micro Edition: Java for mobile and embedded systems, CLDC, MIDP, MEEP
    - RTSJ, Java APIs for XML, JDBC, JMX,…
    - Kotlin and other “Java” languages
    - Android
  - partially mandatory for NPRG059 Advanced Programming Praxis
    - a mandatory course for several Master study branches