Politechnika Wrocławska

Programowanie poprzez testy z wykorzystaniem JUnit

Programowanie ekstremalne (XP)

- XP zaproponowano w 1999 (K. Beck: *„Extreme Programming Explained”)*
- XP dedykowane jest do projektów:
  - O małym lub średnim rozmiarze
  - O wysokim poziomie ryzyka

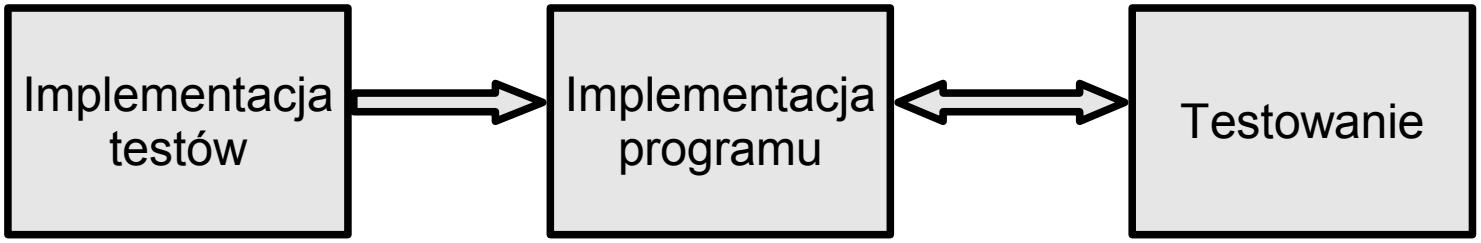
XP - podstawowe zasady

- Iteracyjność
- Sukcesywne projektowanie
- Dynamiczna architektura
- Programowanie parami
- Stały kontakt z klientem
- Dobra komunikacja w zespole
- Programowanie poprzez testy

Programowanie poprzez testy

- Testy akceptacyjne
  - Tworzone dla PU.
  - Wykonywane przez testera.
- Testy jednostkowe
  - Tworzone dla klas.
  - Wykonywane przez programistę.

Programowanie poprzez testy



JUnit

- Twórcy: K.Beck, E.Gamma
- http://junit.org
- Wsparcie dla wielu języków:
  - SUnit (Smalltalk)
  - NUnit (C#)
  - PyUnit (Python)
  - CPPUnit (C++)
  - fUnit (Fortran)
  - JSUnit (JavaScript)

Najprostsze testy

```
public class Largest{
   public static int largest( int[] list ){
      int i, max=Integer.MAX_VALUE;
      for( i=0; i<list.length-1; i++ ){
         if( list[i]>max ) {
            max = list[i];
         }
      }
      return max;
   }
}
```

Najprostsze testy

```
import junit.framework.*;
public class TestLargest extends TestCase{
   public TestLargest(String name) {
      super(name);
   }
   public void testOrder() {
      assertEquals(9,Largest.largest(new int[]
        {8,9,7}));
   }
}
```

Najprostsze testy

```
There was 1 failure:
1)testOrder(TestLargest)junit.framework.Assert
  ionFailed Error: expected<9> but
  was:<2147483647> at
  TestLargest.testOrder(TestLargest.java:7)
```

Najprostsze testy

```
public class Largest{
   public static int largest( int[] list ){
      int i, max=Integer.MAX_VALUE;//max=0
      for( i=0; i<list.length-1; i++ ){
         if( list[i]>max ) {
            max = list[i];
         }
      }
      return max;
   }
}
```

Najprostsze testy

```
public void testOrder() {
   assertEquals(9,Largest.largest(new int[]
     {9,8,7}));
   assertEquals(9,Largest.largest(new int[]
     {8,9,7}));
   assertEquals(9,Largest.largest(new int[]
     {7,8,9}));
}
```

Najprostsze testy

```
There was 1 failure:
1) testOrder(TestLargest)junit.framework.Assert
   ionFailed Error: expected<9> but was:<8> at
   TestLargest.testOrder(TestLargest.java:9)
```

Najprostsze testy

```
public class Largest{
   public static int largest( int[] list ){
      int i, max=0;
      for( i=0; i<list.length-1; i++ ){
            //i<list.length
         if( list[i]>max ) {
            max = list[i];
         }
      }
      return max;
   }
}
```

Najprostsze testy

```
public void testOrder() {
   assertEquals(9, Largest.largest(new int[]
     {9,8,9,7}));
   assertEquals(1, Largest.largest(new int[]
     {1}));
   assertEquals(-7, Largest.largest(new int[]
     {-7,-8,-9}));
}
```

Najprostsze testy

```
There was 1 failure:
1) testOrder(TestLargest)junit.framework.Assert
   ionFailed Error: expected<-7> but was:<0> at
   TestLargest.testOrder(TestLargest.java:12)
```

Najprostsze testy

```
public class Largest{
   public static int largest( int[] list ){
      int i, max=0; //max=Integer.MIN_VALUE
      for( i=0; i<list.length; i++ ){
         if( list[i]>max ) {
            max = list[i];
         }
      }
      return max;
   }
}
```

Wykonywanie testów (konsola)

Kompilacja programu i testów:

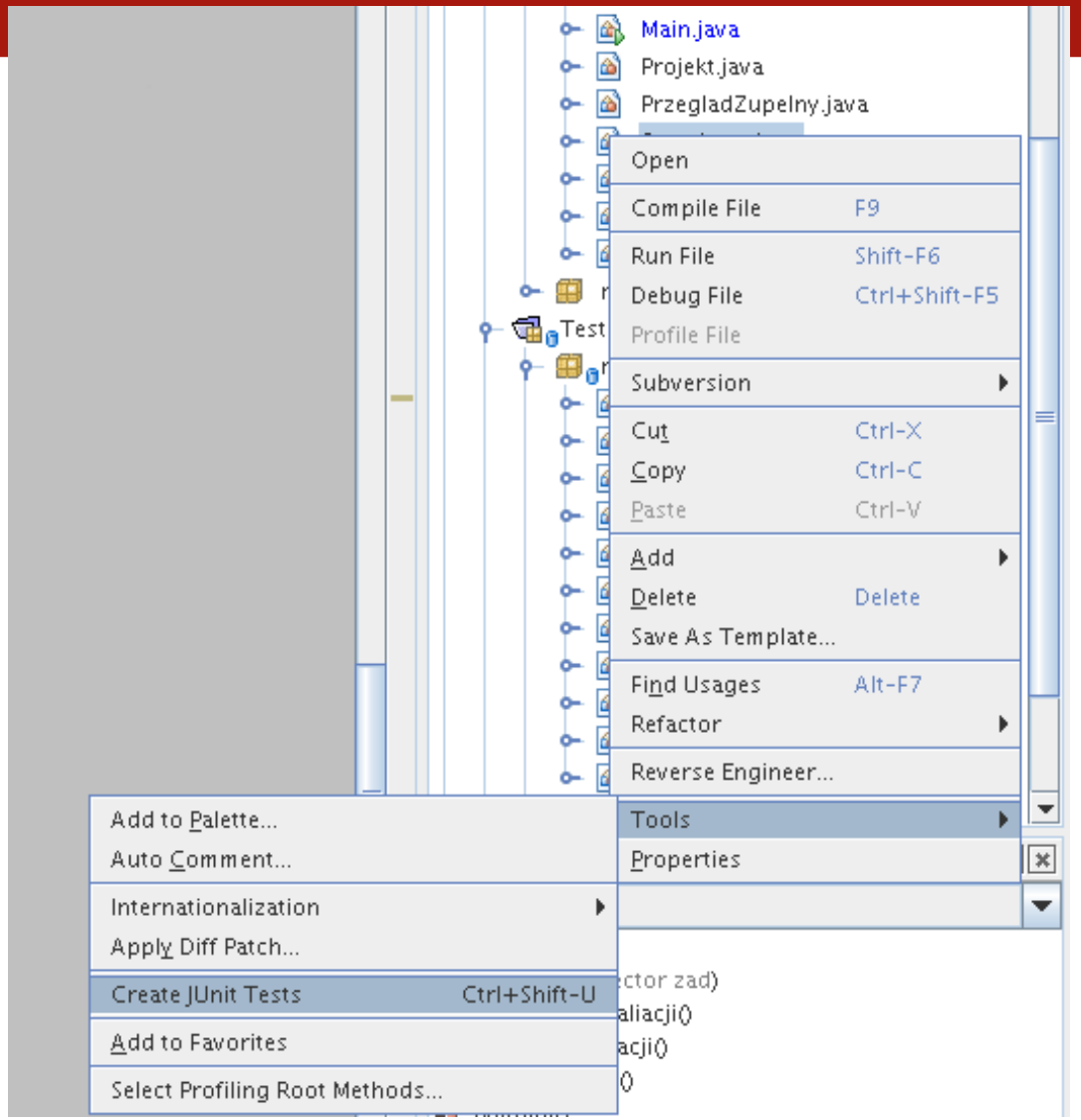
```
javac Largest.java TestLargest.java
```

Uruchomienie testów:

```
java junit.textui.TestRunner TestLargest
```

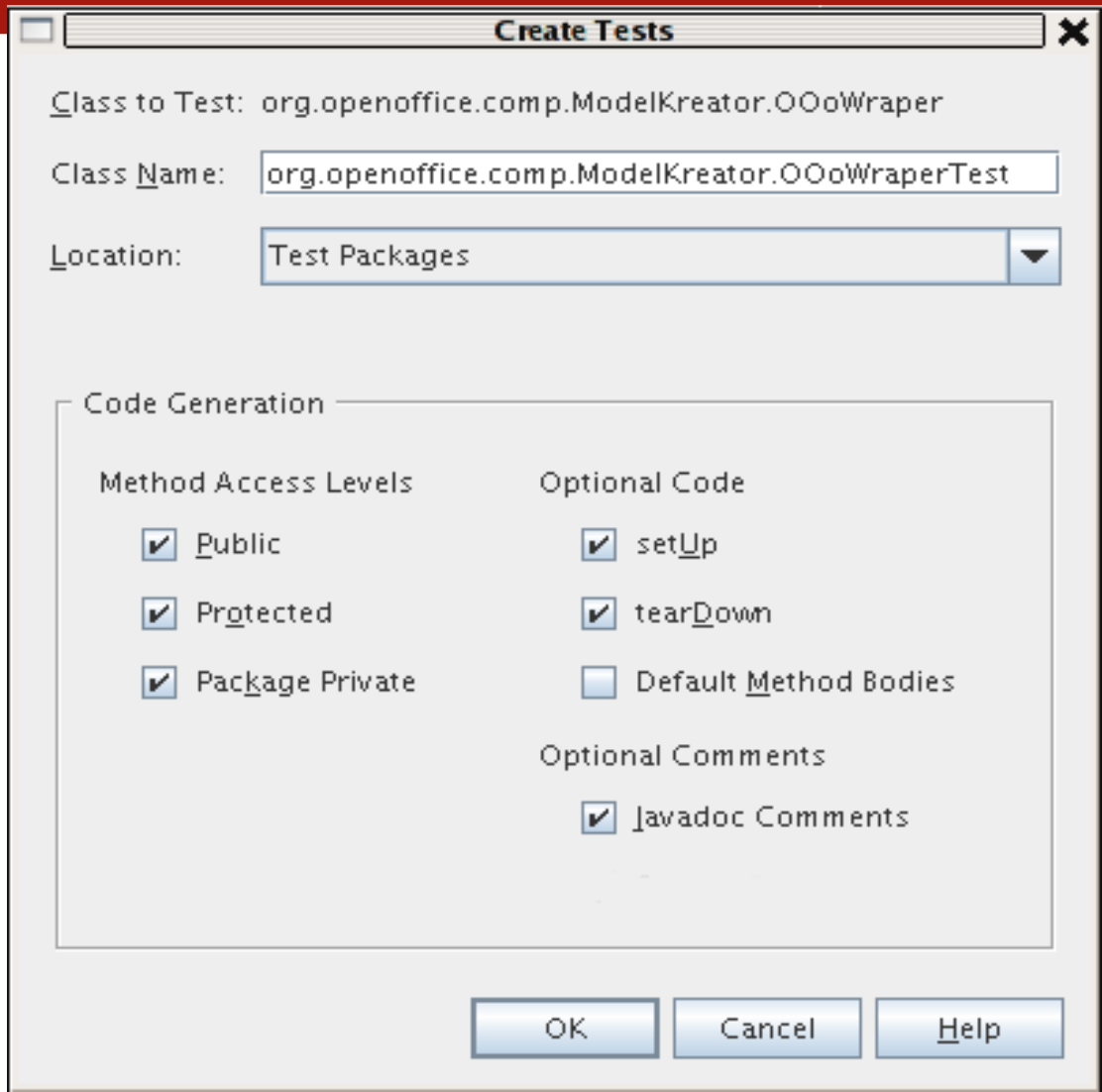
Wykonywanie testów (NetBeans)

- Tworzenie nowego testu



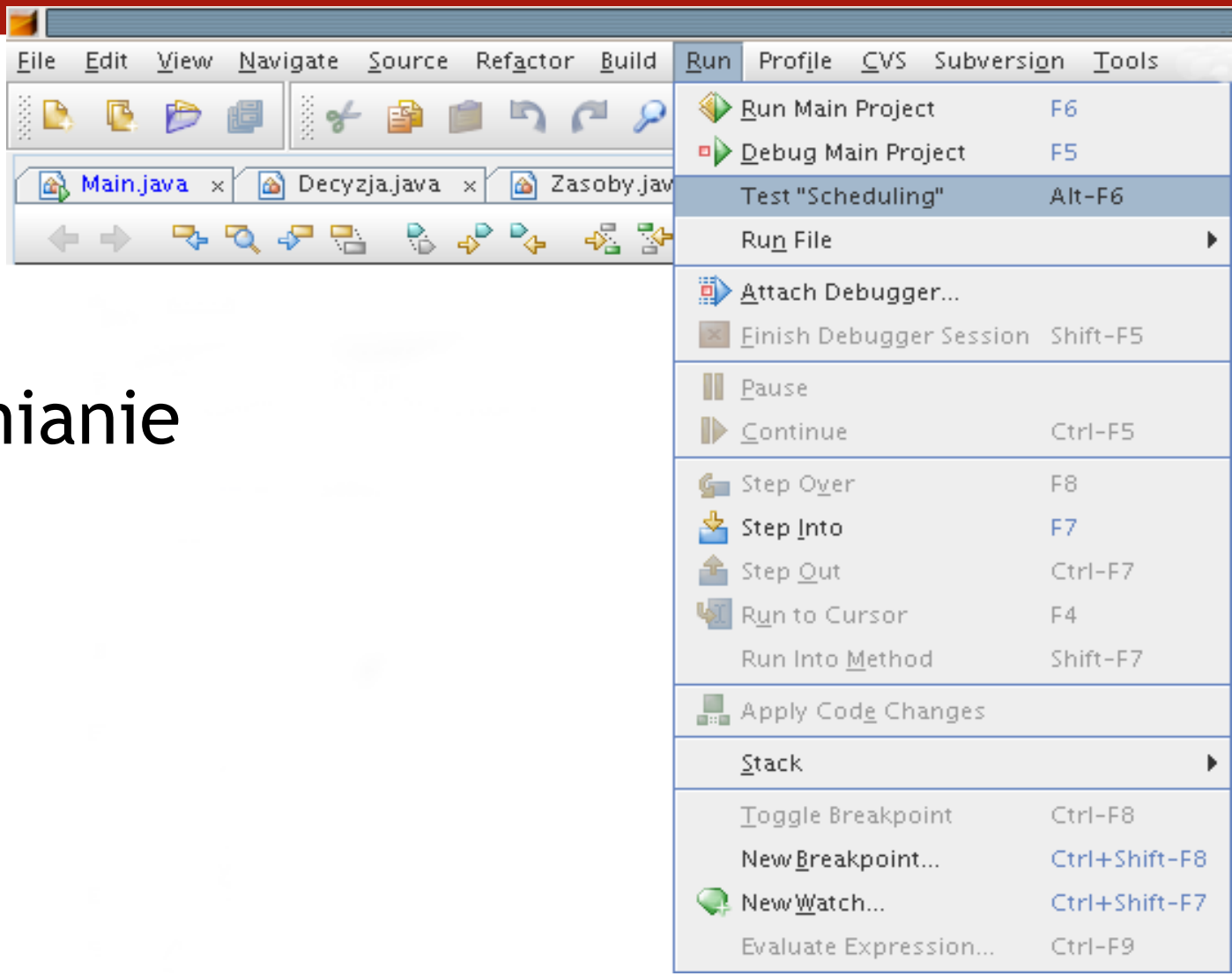
Wykonywanie testów (NetBeans)

- Parametry nowego testu



Wykonywanie testów (NetBeans)

- Uruchamianie testów



Struktura testów jednostkowych

- Dziedziczenie po klasie TestCase

- Adnotacja @Test

```
@Test public void method() {...}
```

- Konwencja nazewnictwa

```
public void testMethod() {...}
```

Asercje JUnit

- `assertEquals([message], expected, actual)`
- `assertEquals([message], expected, actual, tolerance)`
- `assertNull([message], Object object)`
- `assertNotNull([message], Object object)`
- `assertSame([message], expected, actual)`
- `assertTrue([message], condition)`
- `fail([message])`

Konfiguracja testu

```
public class TestDB extends TestCase{
   private Connection dbConn;
   protected void setUp() {
      dbCon = new Connection(...);
      dbConn.connect();
   }
   protected void tearDown() {
      dbConn.disconnect();      }
   public void test1() {...}
   public void test2() {...}
}
```

Wyjątki

```
public void testException() {
   try{
      sortMyList(null);
      fail("Metoda powinna wygenerować
       wyjątek");
   }catch (RuntimeException e) {
      assertTrue(true);
   }
}
```

Co testować?

- Czy wyniki są poprawne?
- Czy warunki brzegowe zostały prawidłowo określone?
- Czy można sprawdzić relacje zachodzące w odwrotnym kierunku?
- Czy można sprawdzić wyniki w alternatywny sposób?
- Czy można wymusić błędy?
- Czy efektywność jest zadowalająca?

Efektywność

```
public void testSym()
{
   long start,end;
   Symulator s=new Symulator();
   start=Calendar.getInstance().getTimeInMilli
     s();
   s.sym();
   end=Calendar.getInstance().getTimeInMillis(
     );
   assertTrue( end-start < 1000 );
}
```

Obiekty imitacji

Obiekty imitacji zastępują rzeczywisty obiekt na czas uruchamiania i testowania kodu.

Zastosowania:

- Obiekt rz. zachowuje się niedeterministycznie
- Obiekt rz. jest trudny do skonfigurowania
- Trudno jest wywołać interesujące nas zachowanie obiektu (np. błąd sieci)
- Obiekt rz. działa powoli
- Obiekt rz. ma interfejs użytkownika
- Obiekt rz. Jeszcze nie istnieje

Obiekty imitacji (mockobjects) - testowanie sevletu

```
public void doGet(HttpServletRequest req,
  HttpServletResponse res)
{
   String s = req.getParameter("cal");
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   double cal = Double.parseDouble(s);
   double joule = 4.1868*cal;
   out.println( String.valueOf(joule) );
}
```

Obiekty imitacji (mockobjects) - testowanie sevletu

```
import junit.framework.*;
import com.mockobjects.servlet.*;

public class TestServlet extends TestCase {
   public void test1(){
      Cal2JServlet s = new Cal2JServlet();
      MockHttpServletRequest req = new
       MockHttpServletRequest();
      MockHttpServletResponse res = new
       MockHttpServletResponse();
```

Obiekty imitacji (mockobjects) - testowanie sevletu

```
        req.setupAddParameter("cal", "1");
        res.setExpectedContentType("text/html");
        s.doGet( req, res );
        double j = Double.parseDouble(res.
           getOutputgetOutputStreamContents());
        assertEquals( 4.1868, j, 0.01 );
    }
}
```

Obiekty imitacji - Easy Mock

- Tryb nagrywania
  - Wołamy wymagane metody
  - Konfigurujemy zwracane wartości
- Tryb odtwarzania
  - Można wołać „nagrane” wcześniej metody
  - Otrzymuje się skonfigurowane wcześniej wartości

Cechy poprawnych testów jednostkowych

- **Automatyzacja** – uruchamianie testów musi być łatwe.
- **Kompletność** – należy testować wszystko co może zawieść.
- **Powtarzalność** – wielokrotne wykonanie testu daje te same wyniki.
- **Niezależność -** od środowiska i innych testów.
- **Profesjonalizm** – kod testujący jest tak samo ważny jak kod dostarczany klientowi.

Log4j

- Zalety
  - Łatwa i szybka implementacja.
  - Możliwość testowania w dowolnym miejscu kodu.
  - Przyspiesza (zastępuje) debugowanie.
- Wady
  - Kod logowania wymieszany z kodem dostarczanym klientowi.

Log4j

```
import org.apache.log4j.*
...
BasicConfigurator.configure();
Logger log=Logger.getLogger(„name”);
log.setLevel( Level.WARN );
log.debug(„...”);
log.info(„...”);
log.warn(„...”);
log.error(„...”);
log.fatal(„...”);
```

Politechnika Wrocławska

Dziękuję za uwagę