Oprogramowanie Systemów Medycznych

Informacje

Strona przedmiotu:

http://osm.ire.pw.edu.pl

Prowadzący:

Robert Kurjata (r.kurjata@ire.pw.edu.pl) pokój 61

Wojciech Gradkowski (w.gradkowski@ire.pw.edu.pl) pokój 68

Tymon Rubel (t.rubel@ire.pw.edu.pl)

Zasady oceniania

5 zadań projektowych x 10 punktów = 50 punktów

Liczba punktów	Ocena
0-25	2
25-30	3
30-35	3.5
35-40	4
40-45	4.5
45-50	5

ocena = punkty otrzymane z zadań projektowych && zaliczenie

Zadania projektowe

- **Krótkie** zadania programistyczne
- Będą prezentowane rozwiązania
- Nieprzekraczalny termin oddania
- Ocenie podlega m.in.: czytelność, stosowanie dobrych praktyk programistycznych, dokumentacja
- Uwagi:
  - skupić się na treści zadania
  - nie komplikować problemu bardziej niż jest to konieczne

Czego można się nauczyć?

- Programowania obiektowego (w JAVIE)
- Projektowania aplikacji posiadających graficzny interfejs użytkownika
- Projektowania aplikacji, które przechowują dane w bazie danych
- Standardów projektowania oprogramowania urządzeń medycznych
- Standardów wykorzystywanych w informatyce medycznej, m.in. DICOM
- Praktycznego przetwarzania obrazów medycznych
  - wyświetlanie i wizualizacja (2D, 3D)
  - analiza danych obrazowych

Środowisko programistyczne

Java

http://www.oracle.com/us/technologies/java/

Zintegrowane Środowisko Programistyczne

http://www.eclipse.org/

System Kontroli Wersji

http://subversion.apache.org/
http://www.eclipse.org/subversive/

Przydatne rzeczy

- dokumentacja API

  http://download.oracle.com/javase/6/docs/api/
- przygotowane przez ORACLE materiały pomocnicze

  http://download.oracle.com/javase/tutorial/
- Thinking in Java – Bruce Eckel

  http://www.mindview.net/Books/TIJ/
- Thinking in Patterns - Bruce Eckel

  http://www.mindview.net/Books/TIPatterns/

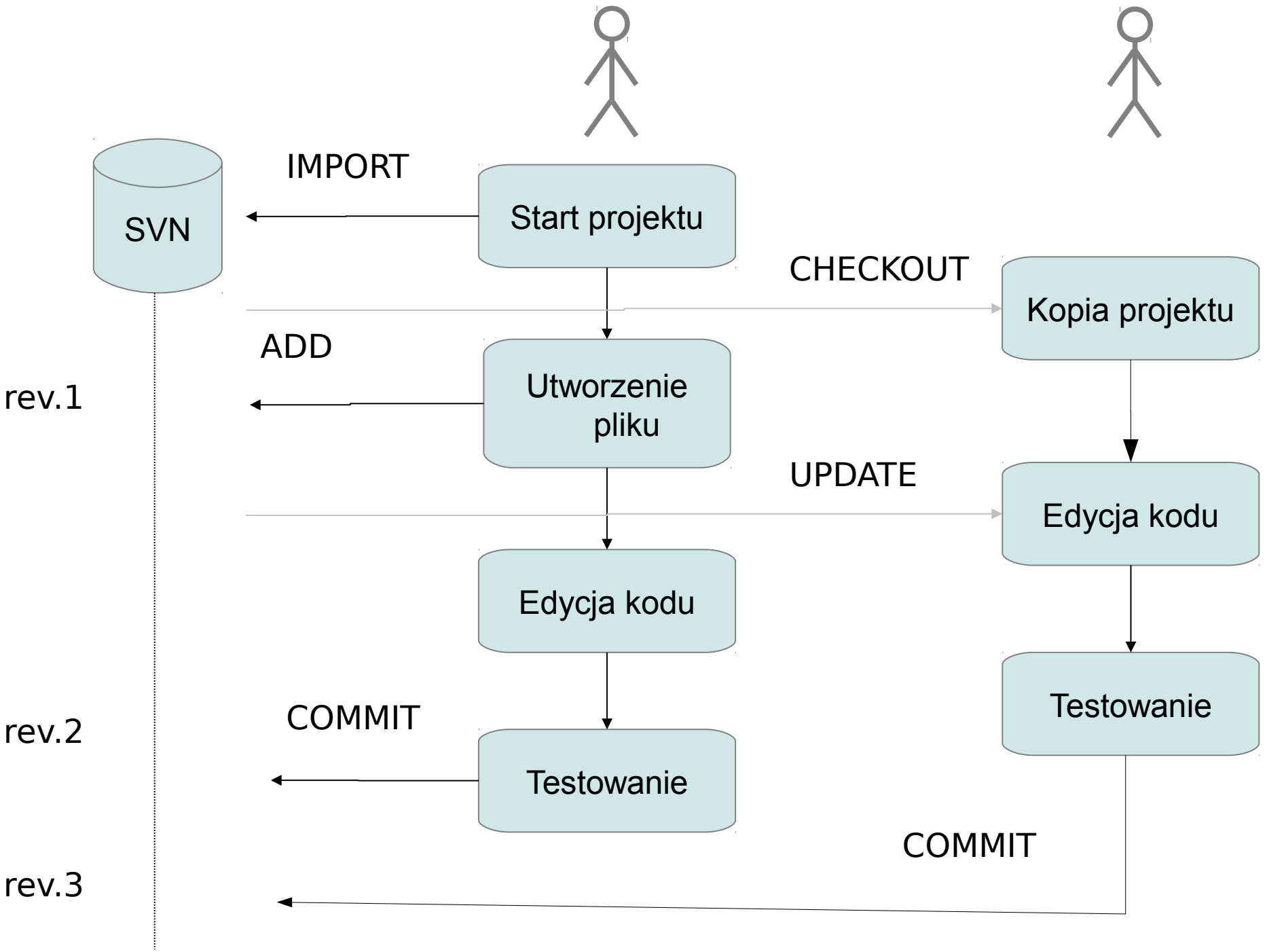
System kontroli wersji

- śledzenie zmian w kodzie źródłowym
- niezbędny przy pracy zespołowej
- obecny w każdej firmie zajmującej się produkcją oprogramowania
- konieczność stosowania ze względu na standardy, m.in. oprogramowania urządzeń medycznych (ISO 13485)

SVN (Subversion)

- **IMPORT**
  - pierwsze dodanie projektu do repozytorium
- **CHECKOUT**
  - pierwsze pobranie projektu z repozytorium
- **ADD/DELETE**
  - dodanie/usunięcie elementu spod kontroli wersji
- **UPDATE**
  - pobranie z serwera wersji kodu (HEAD najnowsza wersja)
- **COMMIT**
  - przesłanie zmodyfikowanego kodu na serwer
- **BRANCH/MERGE**
  - rozgałęzianie kodu

```
svn import ./projekt http://student01@osm.ire.pw.edu.pl/student01 -m „Start”
svn checkout http://student01@osm.ire.pw.edu.pl/student01 ./projekt_svn
cd projekt_svn
svn add Hello.java
svn commit Hello.java -m „Adding hello world!”
```


SVN
IMPORT
Start projektu
CHECKOUT
Kopia projektu
ADD
Utworzenie pliku
rev.1
UPDATE
Edycja kodu
Edycja kodu
Testowanie
COMMIT
rev.2
Testowanie
COMMIT
rev.3

SVN i Eclipse

Tutorial

http://www.eclipse.org/subversive/documentation/index.php

SVN i Eclipse

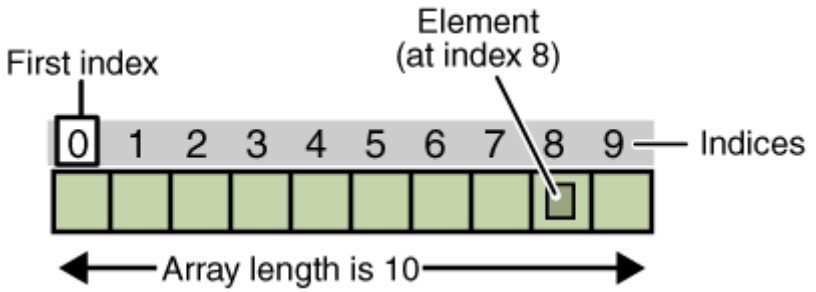
- Stworzenie nowego projektu
  - *File->New...->Java project*
- Udostępnienie projektu
  - *Team->Share Project...*
- Po dokonaniu zmian w projekcie, przesłanie kodu na serwer (commit)
  - *Team->Commit*
- Aktualizacja kodu do (najnowszej) wersji
  - *Team->Update*
- ...
- Importowanie projektu do nowej lokalizacji
  - *Import->SVN->Checkout Projects from SVN*

JAVA nie jest straszna

Typy podstawowe

```
boolean b = true;
byte b = (byte) 127; // <-128, 127>
short s = (short) 1234; // <-32768,327687>
int i = 1; // <-2 147 483 648, 2,147,483,647>
long l = 1234l; // 64 bitowy signed int
float f = 0.4f;
double d = 3.14;
char ch = 'ą'; // 16 bitowy Unicode
```

Tablice




```
int[] arrayOfInts1 = new int[5];

int[] arrayOfInts2 = {1, 2, 3, 4, 5};

// kopiowanie

System.arraycopy( arrayOfInts2, 0, arrayOfInts1, 0, 5 );

// wielowymiarowe

String[][] names = {
        {"Mr. ", "Mrs. ", "Ms. "},
        {"Smith", "Jones"}
    };
```

Tablice

```
char[] kot = { 'k', 'o', 't' };
char[] kat = kot;
kat[1] = 'a';
System.out.println(kat);
System.out.pribtln(kot);
```

Operatory

```
int a = 1;
System.out.println( a++ );
System.out.println( a );
System.out.println( ++a );
System.out.println( -a );
```

Proste	Arytmetyczne	Jednostkowe	Porównawcze	Warunkowe	Porównanie typów	Bitowe
=	**+** **-** ***** **/** **%**	**+** **-** **++** **--** **!**	**==** **!=** **>** **<** **>=** **<=**	**&&** **||** **?:**	**instanceof**	**~** **<<** **>>** **>>>** **&** **^** **|**

Kontrola

```
int a = 1;
if ( a == 1 ) {
    System.out.pribtln("is 1");
} else {
    System.out.println("not 1");
}
char[] kot1 = { 'k', 'o', 't' };
char[] kot2 = { 'k', 'o', 't' };
if ( kot1 == kot2 ) {
    System.out.println( "???" );
}
if ( Arrays.equals( kot1, kot2 ) ) {
    System.out.println("ten sam kot");
}
String isKot = Arrays.equals( kot1, kot2 ) ? "yes" : "no";
```

Kontrola

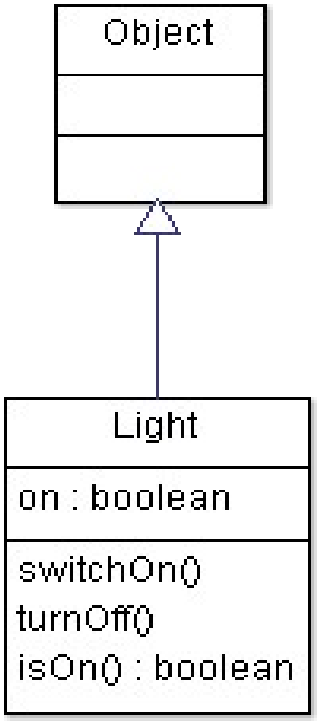
```
switch ( type ) {
    case 1:
        System.out.println("Type 1");
        break;
    case 2:
    case 3:
        System.out.println("Type 2 or 3");
        break;
    default:
       System.out.println("Type unknown");
}
```

Kontrola pętle

```
while ( true ) {
    System.out.println("...from here to eternity");
}
int i = 0;
do {
     i++;
} while ( i < 10 );
for ( int i = 10; i >= 0; i-- ) {
    System.out.println( i + " green bottles standing on the wall");
}
for ( int i = 0;; i++ ) {
    if ( i == 10 ) {
        beak;
    } else {
        continue;
    }
    System.out.pribtln("???");
}
```

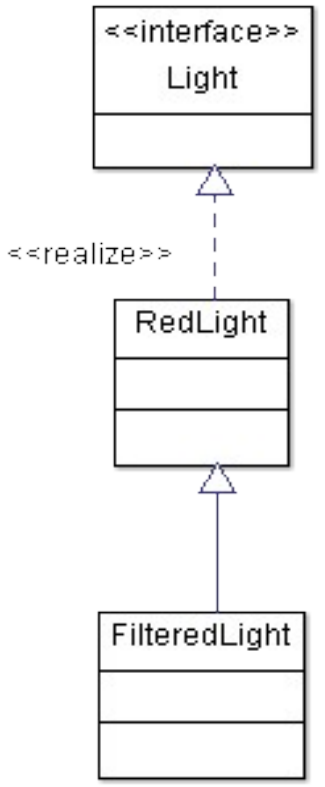
Obiekty

```
class Light {
    // pola (private, public, protected, default)
    private boolean on;
    // konstruktory
    public Light() {
        on = false;
    }
    // metody
    public void switchOn() { on = true; }
    public void turnOff() { on = false; }
    public boolean isOn() { return on; }
}
```

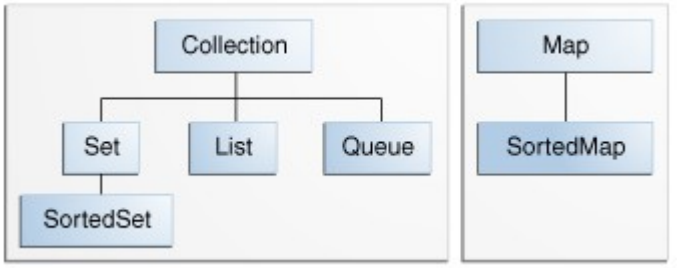


Programowanie obiektowe

```
interface Light {
    public void switchOn();
    public int emit();
}
public RedLight implements Light {
    protected on = false;
    public switchOn() { on = true; }
    public int emit() {
        return on ? 650 : 0;
    }
}
public FilteredLight extends RedLight {
    private int filter( int light ) { return on ? light - 140 : 0; }
    public int emit() {
        return filter ( super.emit() );
    }
}
```



Kolekcje (kontenery)



- **Collection** - grupa elementów
- **Set** - kolekcja, która nie może zawierać duplikatów (!equals())
- **List** - uporządkowana kolekcja (sekwencja)
- **Queue** - kolejka
- **Map** - kolekcja, która trzyma pary klucz -> wartość

kolekcje korzystają z typów generycznych, tzn. można wymusić przechowywanie obiektów tylko określonego typu, np.

List<Light> lightList; // lista na obiektu typu Light

Przykład

```
Collection<Light> lights = new ArrayList<Light>();
lights.add( new RedLight() );
lights.add( new FilteredLight() );

// for-each
for ( Light light : lights ) {
    light.switchOn();
    System.out.println("I can see " + light.emit() );
}
```

Jak uruchomić program?

MainClass.java

```
package pl.edu.pw.ire.osm.test
import [...]
class MainClass() {
    public static void main( String[] args ) {
        //
    }
}
```

Zadanie

Wyświetlić na standardowym wyjściu wynik działania 10!

$$n! = 1*2*3* .... *n$$