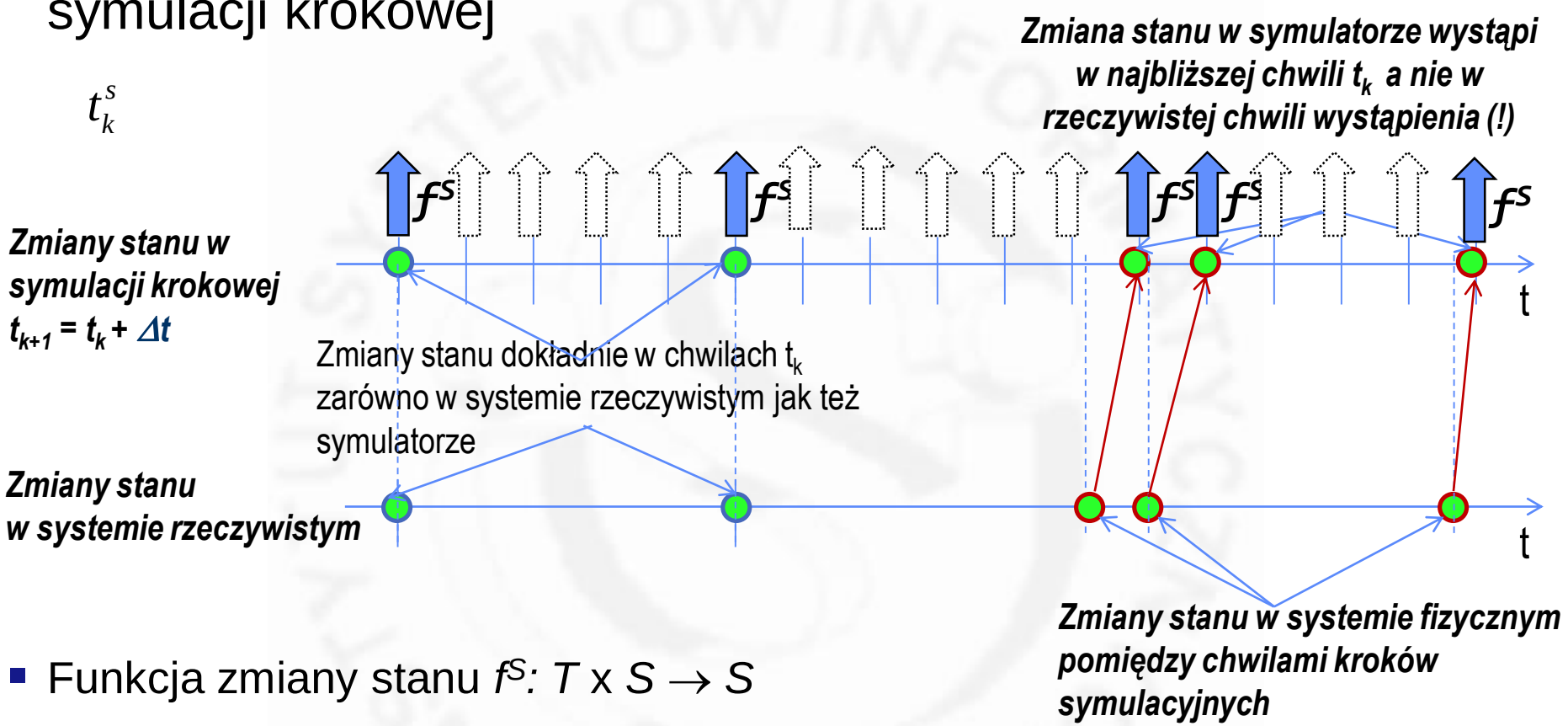
Laboratorium 3
Symulacja krokowa

dr inż. Jarosław Rulka
jaroslaw.rulka@wat.edu.pl

Symulacja krokowa - koncepcja

- Zmiany stany w systemie rzeczywistym vs. zmiany stanu w symulacji krokowej



- Funkcja zmiany stanu f^S: $T \times S \rightarrow S$
- Wyznacza stan, w jakim znajdzie się system w chwili t_k
- Różne zmiany opisane są różnymi funkcjami zmiany stanu
- W każdej chwili t_k zachodzi próba wykonania wszystkich funkcji

Symulacja krokowa - algorytm

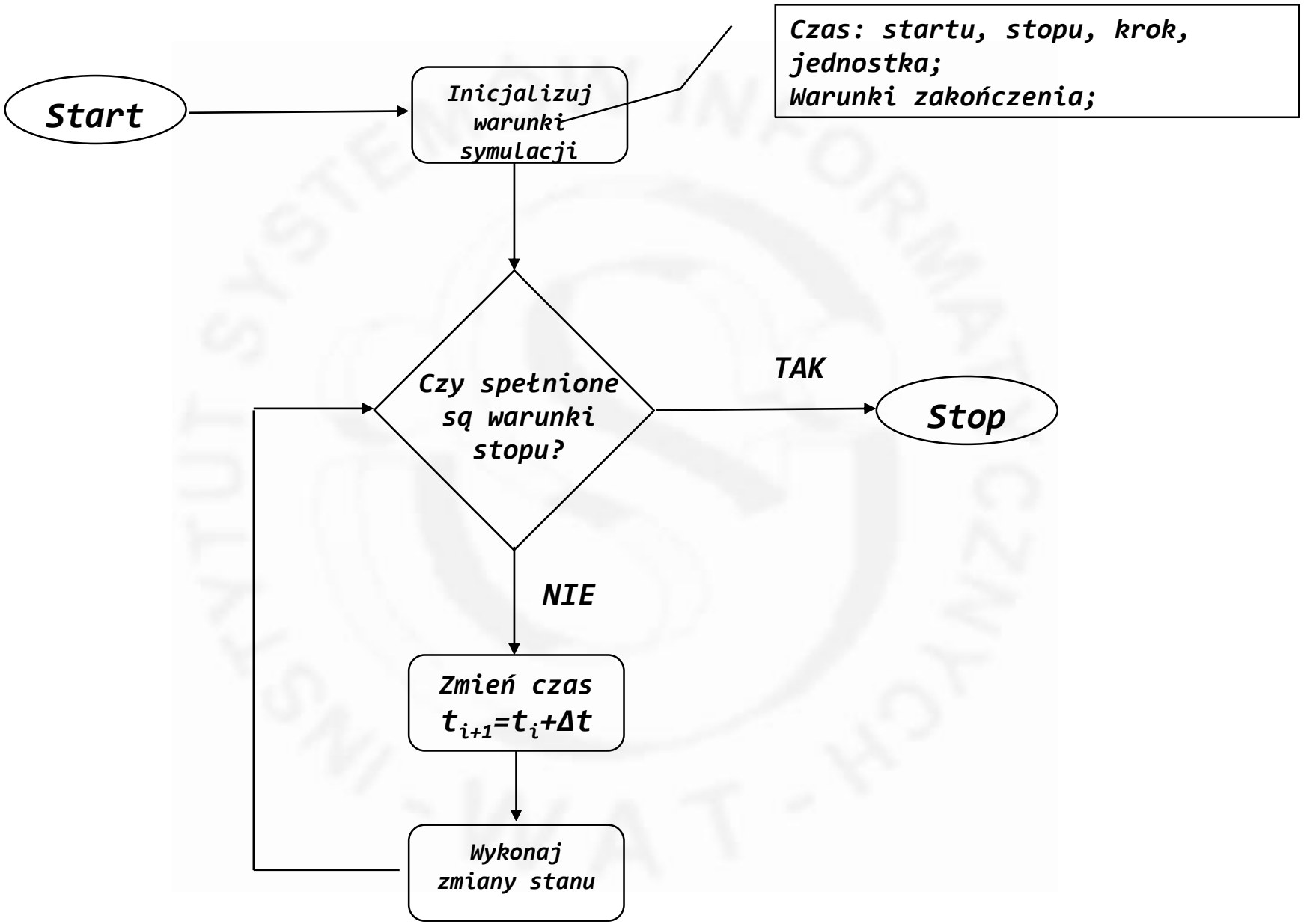
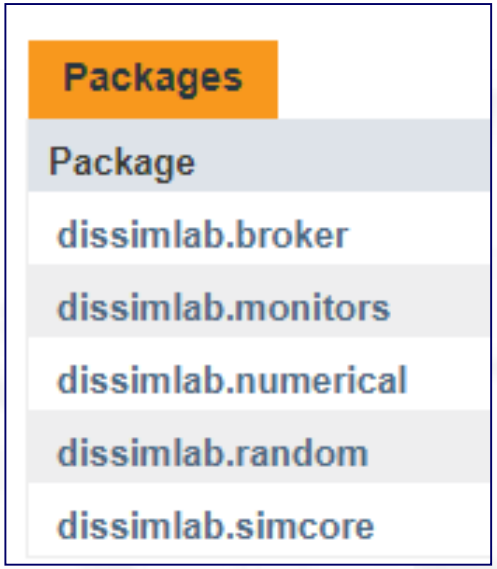
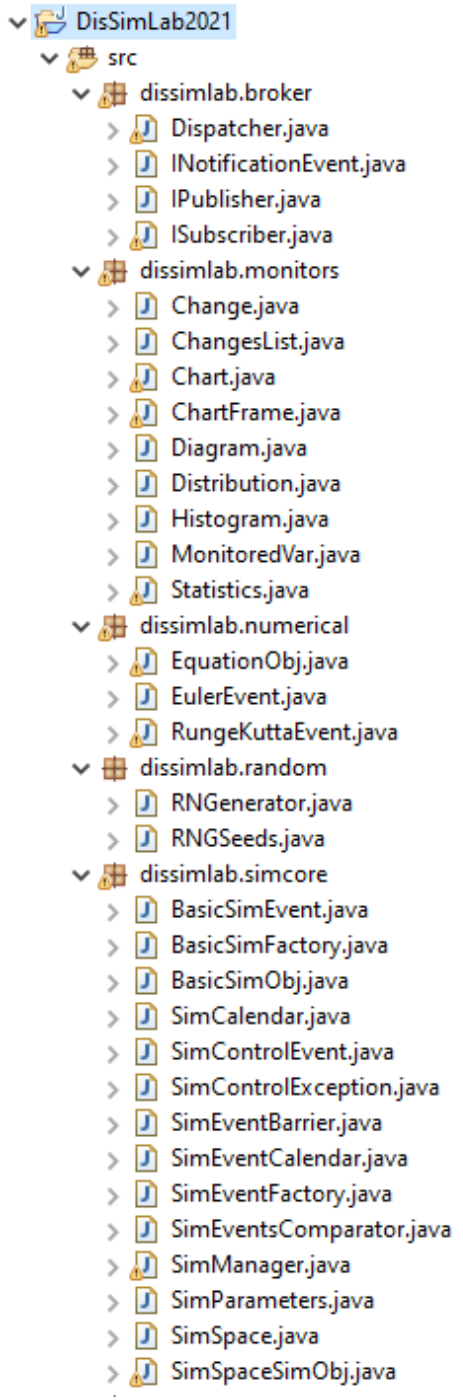


Diagram pakietów

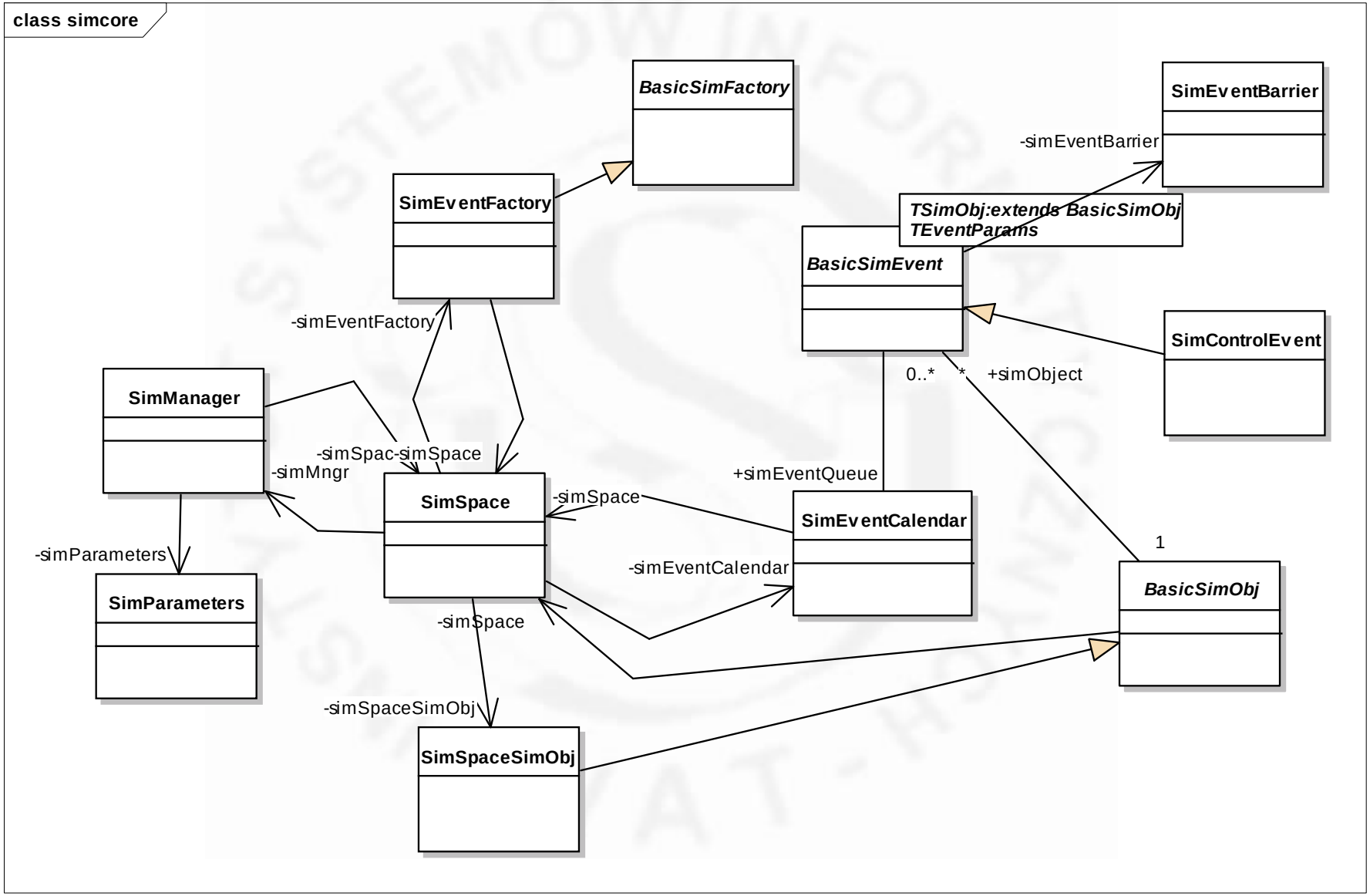




simcore – pakiet grupujący klasy odpowiedzialne za zarządzanie eksperymentem, upływ czasu symulacyjnego, zdarzenia i zmiany stanu;
monitors – klasy odpowiedzialne za monitorowanie, gromadzenie i udostępnienie do analizy statystycznej szeregów czasowych pochodzących ze wskazanych zmiennych programowych;
random – pakiet klas generatora liczb (pseudo)losowych;
broker – klasy umożliwiające przesyłanie komunikatów pomiędzy obiektami symulacyjnymi (agentami, środowiskiem);
numerical – pakiet klas (jedno-wielo)krokowych metod numerycznych do iteracyjnego przybliżonego rozwiązania równań różniczkowych zwyczajnych, spełniających założenia o istnieniu oraz jednoznaczności rozwiązania.

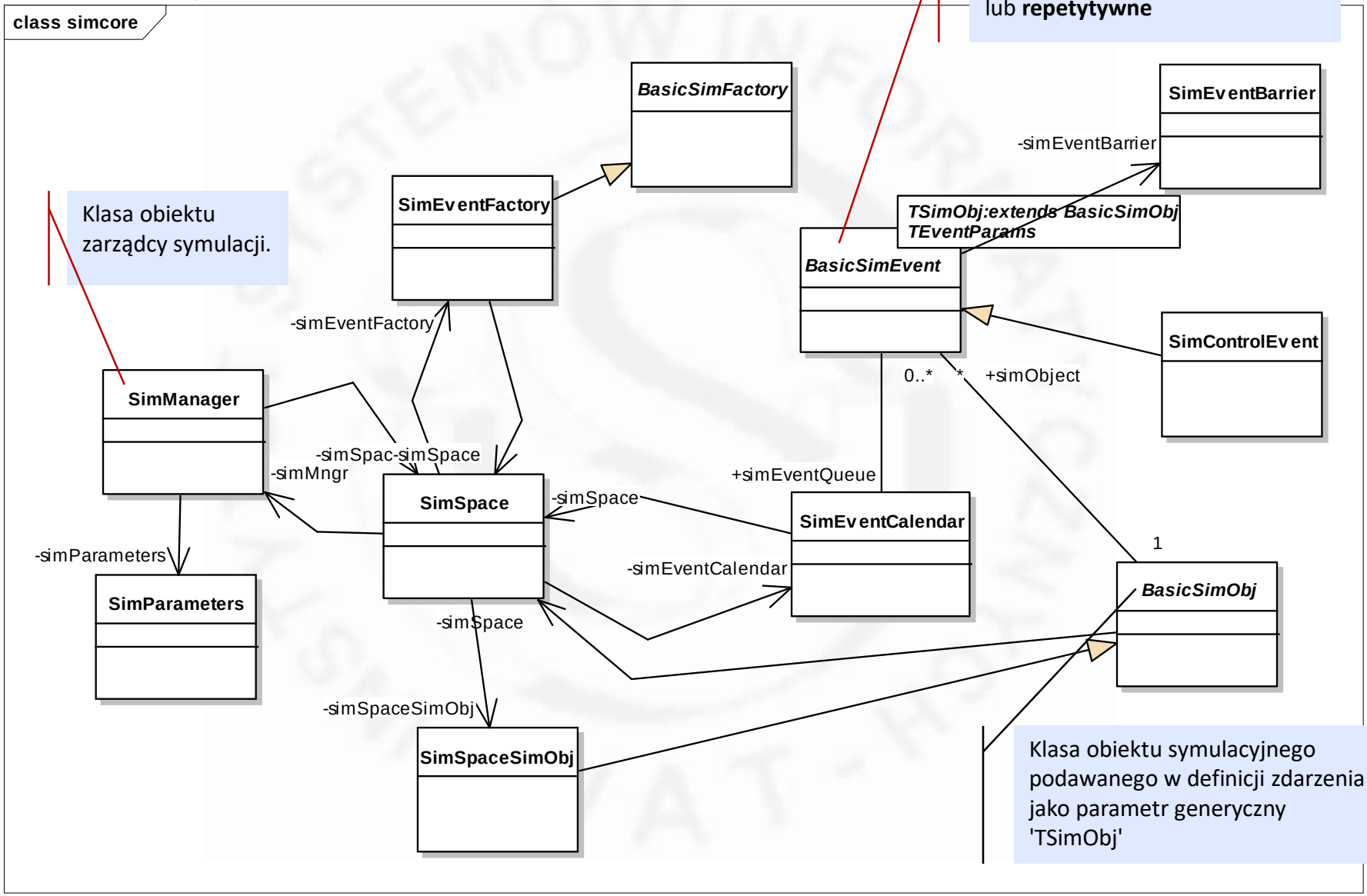
DisSimLab

Główne klasy pakietu „dissimlab.core”



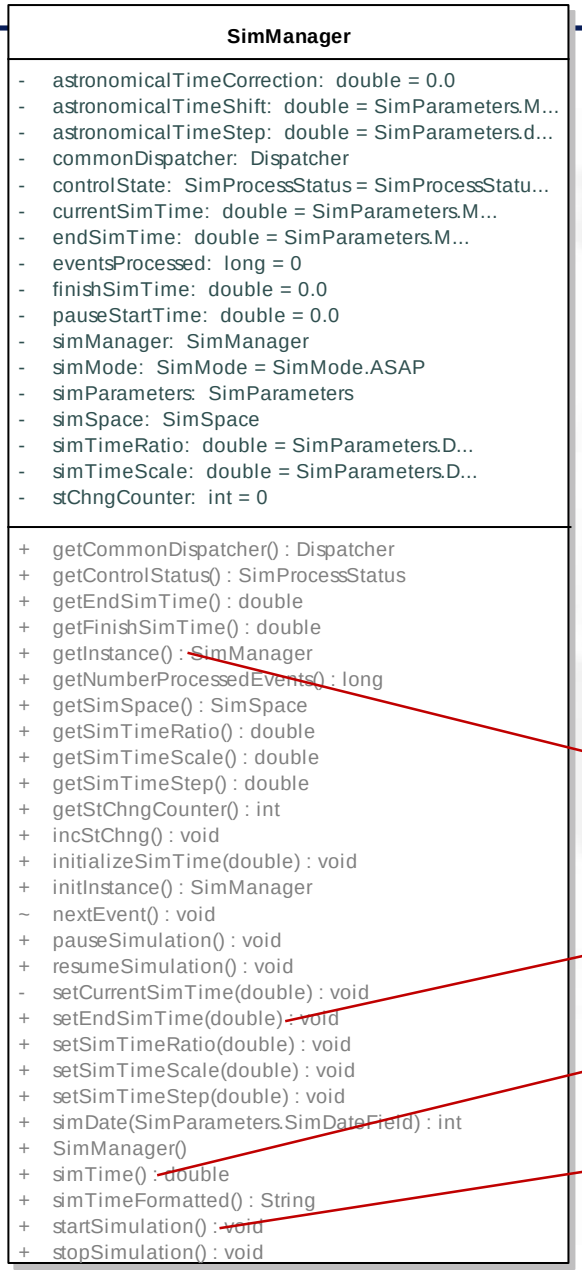
DisSimLab

Główne klasy pakietu „dissimlab.core”



DisSimLab

public class SimManager



Klasa obiektu zarządcy symulacji. Występuje w symulacji jako singleton. Steruje przebiegiem symulacji: uruchomieniem, pauzowaniem i zakończeniem. Tworzy i utrzymuje referencje do obiektów: wspólnej przestrzeni symulacji (simSpace), wspólnego w symulacji pośrednika komunikatów (commonDispatcher), parametrów symulacji (simParameters). Przechowuje dane niezbędne do sterowanie przebiegiem symulacji oraz wyznaczania aktualnego czasu symulacyjnego i kolejności obsługi zdarzeń.

Pobranie referencji do obiektu singletona zarządcy.

Metoda ustawia wartość czasu planowanego końca symulacji.

Metoda podaje aktualną wartość czasu symulacyjnego.

Metoda służy do natychmiastowego uruchomienia symulacji, pod warunkiem, że symulacja nie została już uruchomiona lub zakończona.

DisSimLab

```
public abstract class BasicSimObj
implements IPublisher, ISubscriber
```

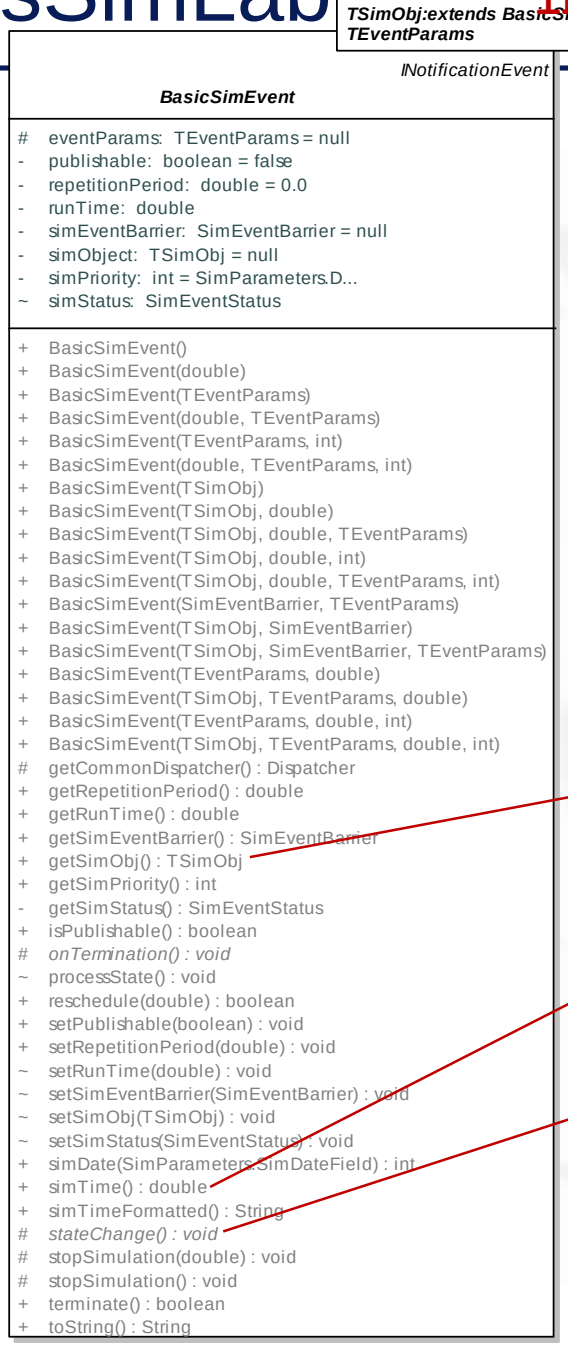
Klasa obiektu symulacyjnego podawanego w definicji zdarzenia jako parametr generyczny 'TSimObj'. Przyjmuje się, że każde zdarzenie musi mieć wskazany obiekt symulacyjny.

Metoda zwraca referencję do wspólnego w symulacji pośrednika komunikatów pozostającego pod kontrolą zarządcy symulacji 'SimManager'.

Metoda podaje aktualną wartość czasu symulacyjnego.

Metoda podaje aktualną wartość czasu symulacyjnego w postaci sformatowanej.

Class BasicSimEvent<TSimObj extends BasicSimObj,TEventParams> implements INotificationEvent<TEventParams>



Klasa zdarzenia symulacyjnego, realizowanego jako jednorazowe lub repetytywne. Po powołaniu zdarzenie jest wstrzymywane do zadanego czasu lub na barierze. W definicji zdarzenia podawane są dwa parametry generyczne: 'TSimObj' - klasa obiektu, którego dotyczy zdarzenie oraz 'TEventParams' - klasa obiektu z dodatkowymi danymi, które m.in. można wykorzystać podczas realizacji zdarzenia w metodzie stateChange().

Pobranie referencji do obiektu symulacyjnego powiązanego z tym obiektem zdarzenia.

Metoda podaje aktualną wartość czasu symulacyjnego.

Metoda abstrakcyjna, która powinna być nadpisana dla zdefiniowania zmiany stanu w danym kroku w ramach danego zdarzenia.

DisSimLab

BasicSimEvent(TSimObj entity, TEventParams params, double period)

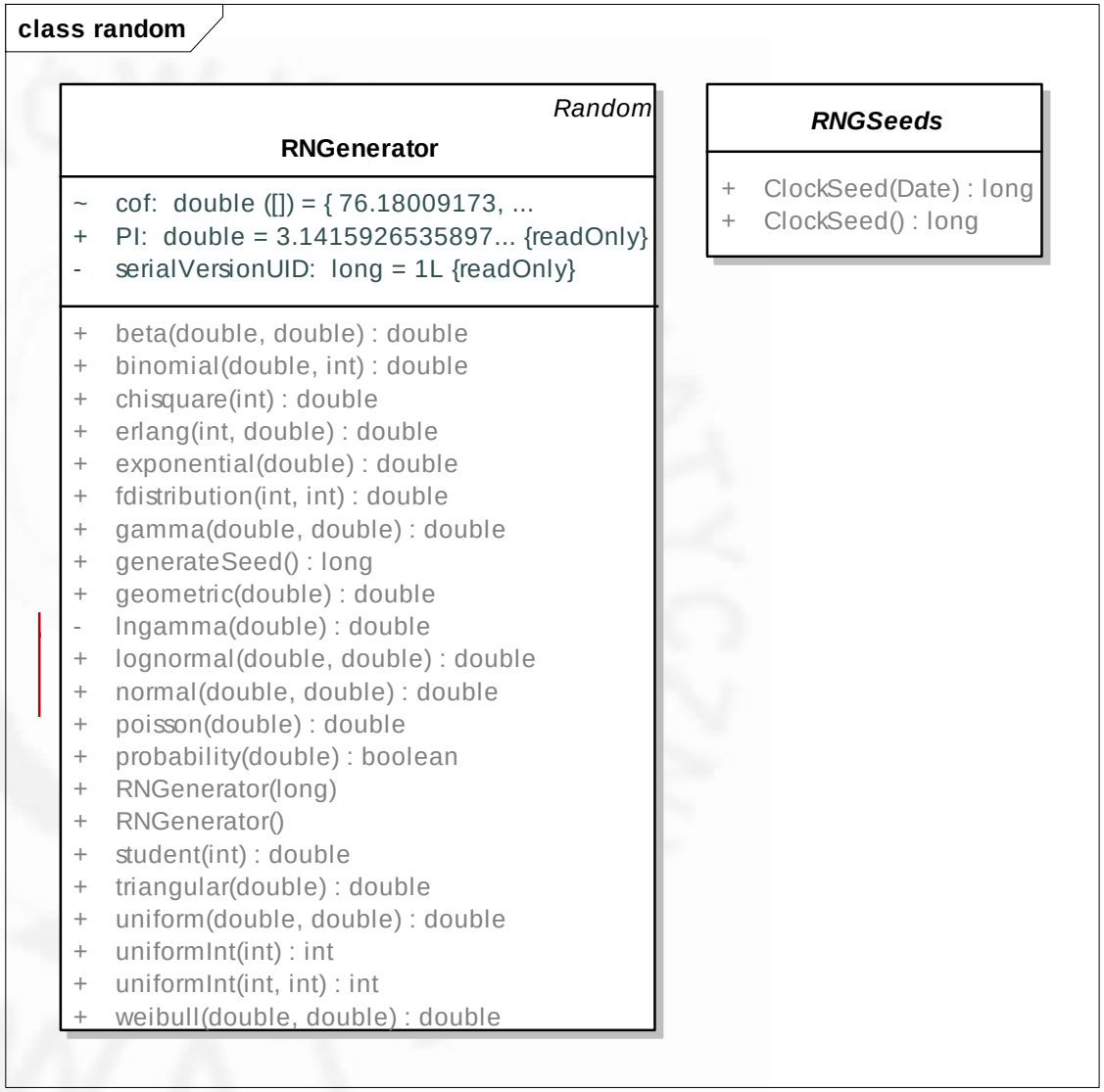
Konstruktor tworzący repetytywne zdarzenie dla obiektu **'entity'** (obiekt musi istnieć) z krokiem symulacyjnym równym **'period'**.

DisSimLab

`public class RNGenerator extends java.util.Random`

Klasa generatora liczb pseudolosowych wykorzystująca metody odwracania dystrubuanty, odrzucania oraz przyblizone na bazie szeregu Taylora.

Metody generujące liczbę pseudolosową jako realizację różnych rozkładów

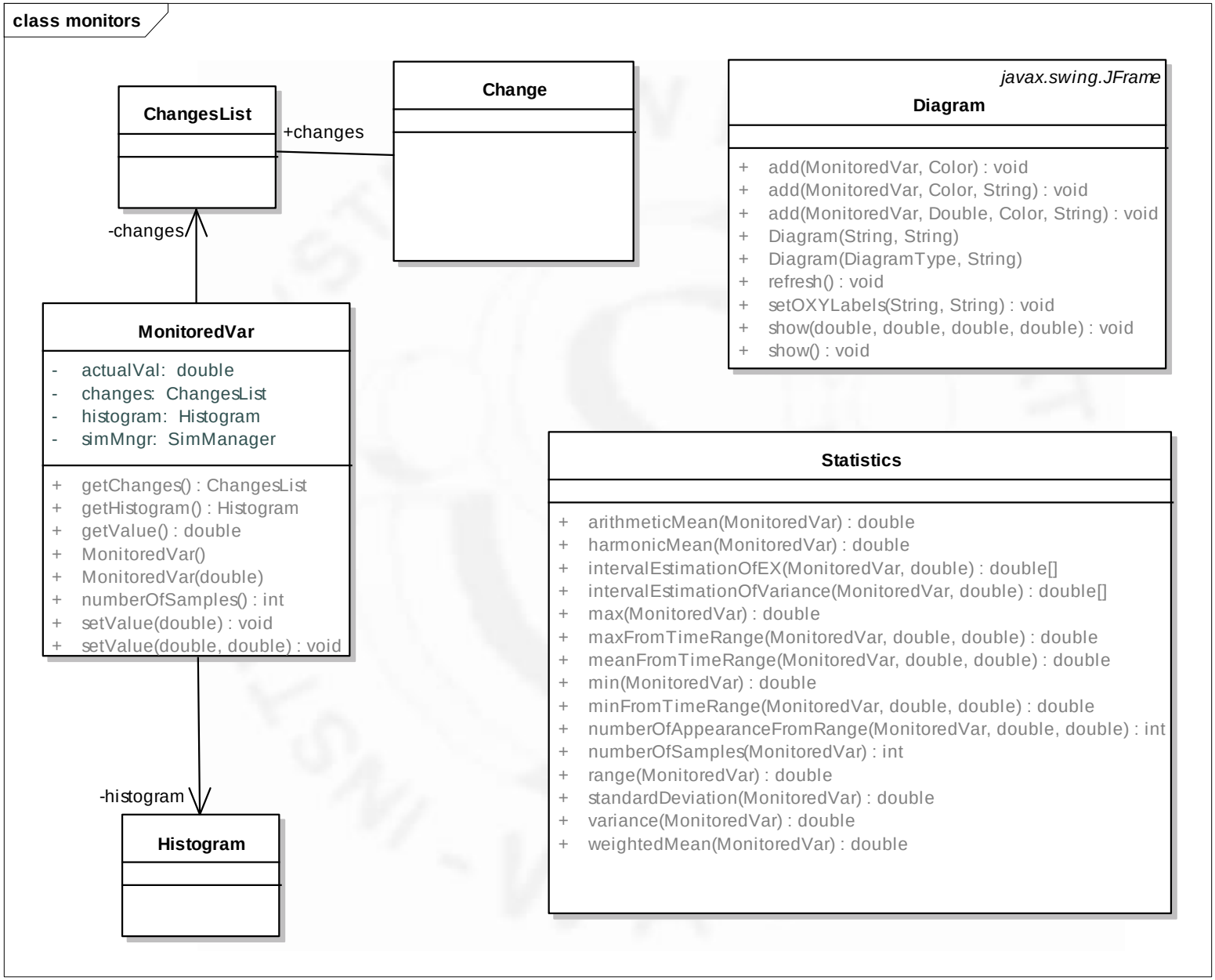


DisSimLab

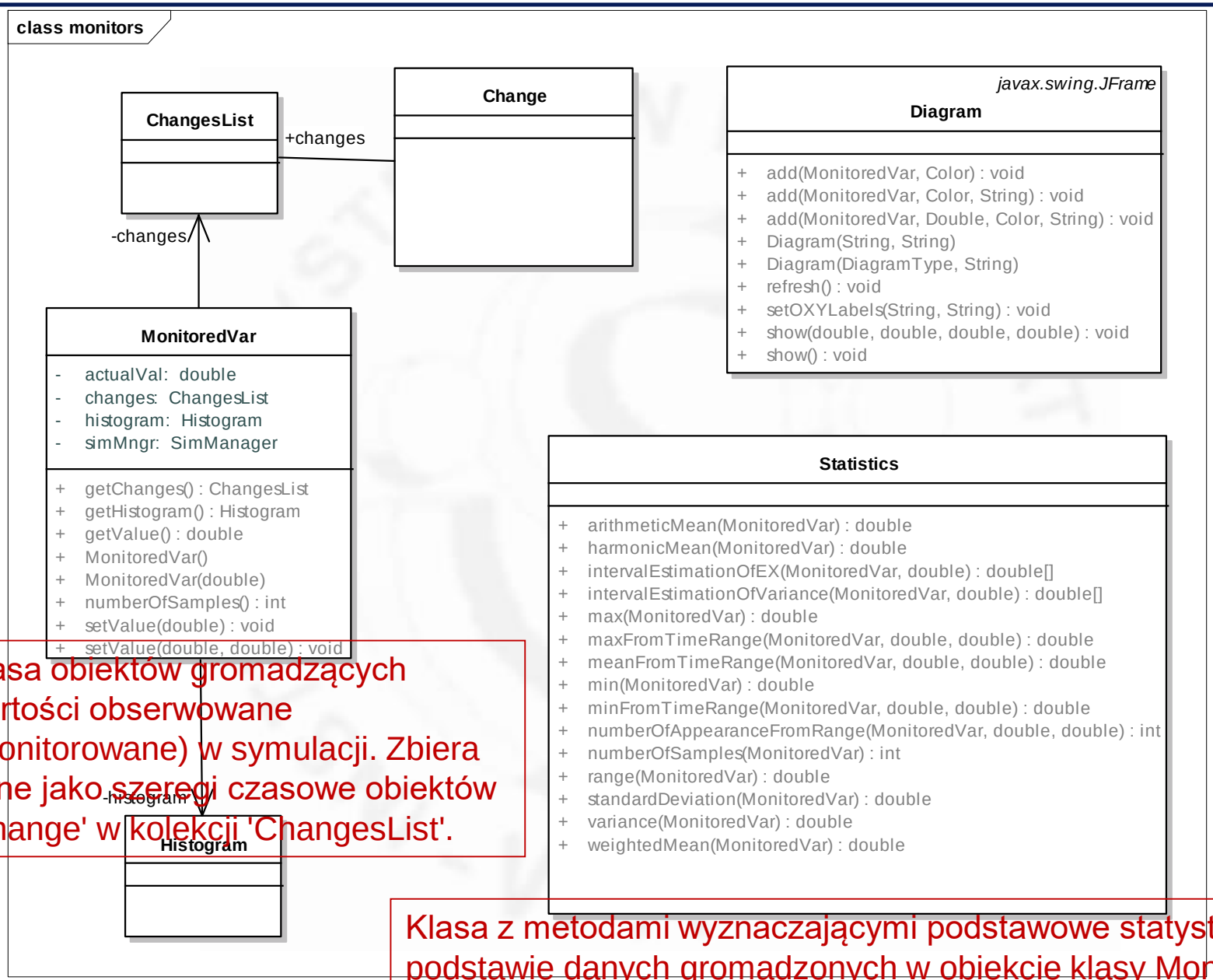
- Przykład generowania liczb pseudolosowych:

```
RNGenerator sg = new RNGenerator();      /* utworzenie nowego
generatora bez podania ziarna */

long seed = RNGenerator.generateSeed(); /* wygenerowanie
nowego ziarna na podstawie aktualnego czasu systemu
operacyjnego */

RNGenerator sg2 = new RNGenerator(seed);/* utworzenie nowego
generatora liczb pseudolosowych z podaniem ziarna jako
parametru */

double d = sg.normal(0,1); /* wygenerowanie nowej liczby o
zadanym rozkładzie, w tym przypadku z rozkładu normalnego
(0,1) */
```

DisSimLab


class monitors
ChangesList
+changes
Change
-changes
MonitoredVar
- actualVal: double
- changes: ChangesList
- histogram: Histogram
- simMngr: SimManager
+ getChanges() : ChangesList
+ getHistogram() : Histogram
+ getValue() : double
+ MonitoredVar()
+ MonitoredVar(double)
+ numberOfSamples() : int
+ setValue(double) : void
+ setValue(double, double) : void
-histogram
Histogram
javax.swing.JFrame
Diagram
+ add(MonitoredVar, Color) : void
+ add(MonitoredVar, Color, String) : void
+ add(MonitoredVar, Double, Color, String) : void
+ Diagram(String, String)
+ Diagram(DiagramType, String)
+ refresh() : void
+ setOXYLabels(String, String) : void
+ show(double, double, double, double) : void
+ show() : void
Statistics
+ arithmeticMean(MonitoredVar) : double
+ harmonicMean(MonitoredVar) : double
+ intervalEstimationOfEX(MonitoredVar, double) : double[]
+ intervalEstimationOfVariance(MonitoredVar, double) : double[]
+ max(MonitoredVar) : double
+ maxFromTimeRange(MonitoredVar, double, double) : double
+ meanFromTimeRange(MonitoredVar, double, double) : double
+ min(MonitoredVar) : double
+ minFromTimeRange(MonitoredVar, double, double) : double
+ numberOfAppearanceFromRange(MonitoredVar, double, double) : int
+ numberOfSamples(MonitoredVar) : int
+ range(MonitoredVar) : double
+ standardDeviation(MonitoredVar) : double
+ variance(MonitoredVar) : double
+ weightedMean(MonitoredVar) : double

DisSimLab

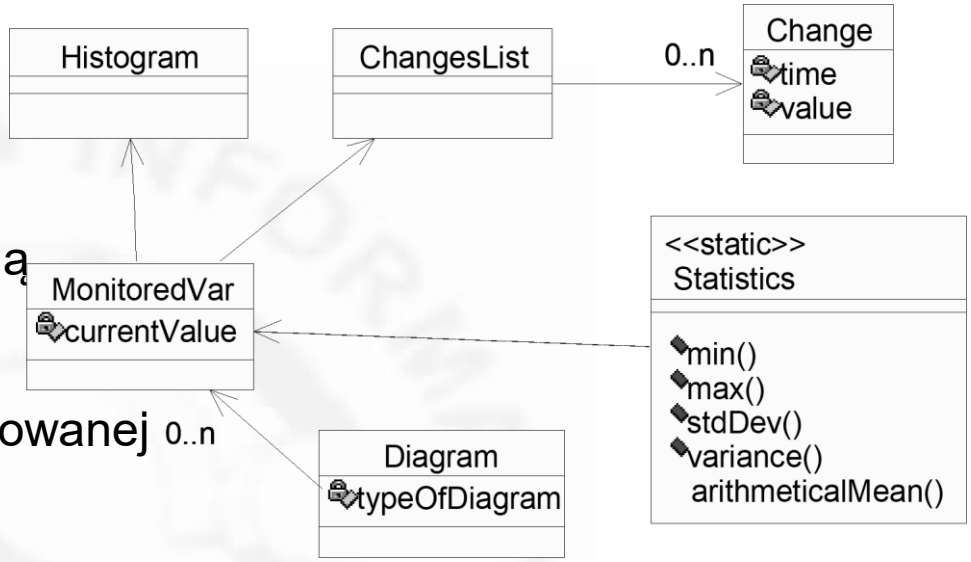


Klasa obiektów gromadzących wartości obserwowane (monitorowane) w symulacji. Zbiera dane jako szeregi czasowe obiektów 'Change' w kolekcji 'ChangesList'.

Klasa z metodami wyznaczającymi podstawowe statystyki na podstawie danych gromadzonych w obiekcie klasy MonitoredVar.

DisSimLab

Diagram klas pakietu Monitors:



- MonitoredVar
  - reprezentująca zmienną monitorowaną
- ChangeList
  - listę zmian wartości zmiennej monitorowanej
- Change
  - zapamiętuje wartość zmiennej monitorowanej
- Histogram
  - posortowaną listę wszystkich wartości zmiennej monitorowanej
- Statistics
  - statyczne metody do wyliczania podstawowych statystyk dla przekazanej jako parametr zmiennej monitorowanej;
- Diagram
  - wykresy: dystrybuanta, histogram lub przebieg w czasie;

DisSimLab

- Przykład użycia ’monitora’ obserwacji:

```
MonitoredVar mv = new MonitoredVar();  /* powołanie nowej
zmiennej monitorowanej */

mv.setValue(5); /* przypisanie nowej wartości zmiennej
monitorowanej */

double d = mv.getValue();  /* odczytanie aktualnej wartości
zmiennej monitorowanej */

Histogram h = mv.getHistogram();  /* dostęp do histogramu
zmiennej monitorowanej np. w celu liczenia statystyk

ChangeList chl = mv.getChanges();/* dostęp do listy zmian np.
w celu liczenia statystyk */
```

- Przykład użycia klasy Statistics :

```
double d =  Statistics.range(mv); /* wyliczenie rozstępu dla
zmiennej mv, (mv – zmienna monitorowana) */

double[] dr  = Statistics.intervalEstimationOfEX(mv, 0.9);

/* estymacja przedziałowa dla wartości oczekiwanej, wyliczenie
przedziału ufności dla zmiennej EX i gamma równego 0.9, jako
wynik zwracane są dwie liczby w tablicy dwuelementowej, z
których pierwsza jest lewym krańcem przedziału ufności, druga
natomiast prawym krańcem przedziału ufności*/

int n = Statistics.numberOfSamples(mv); /* zwraca liczbę
całkowitą, która określa ile razy została wywołana metoda
setValue dla zmiennej monitorowanej*/
```

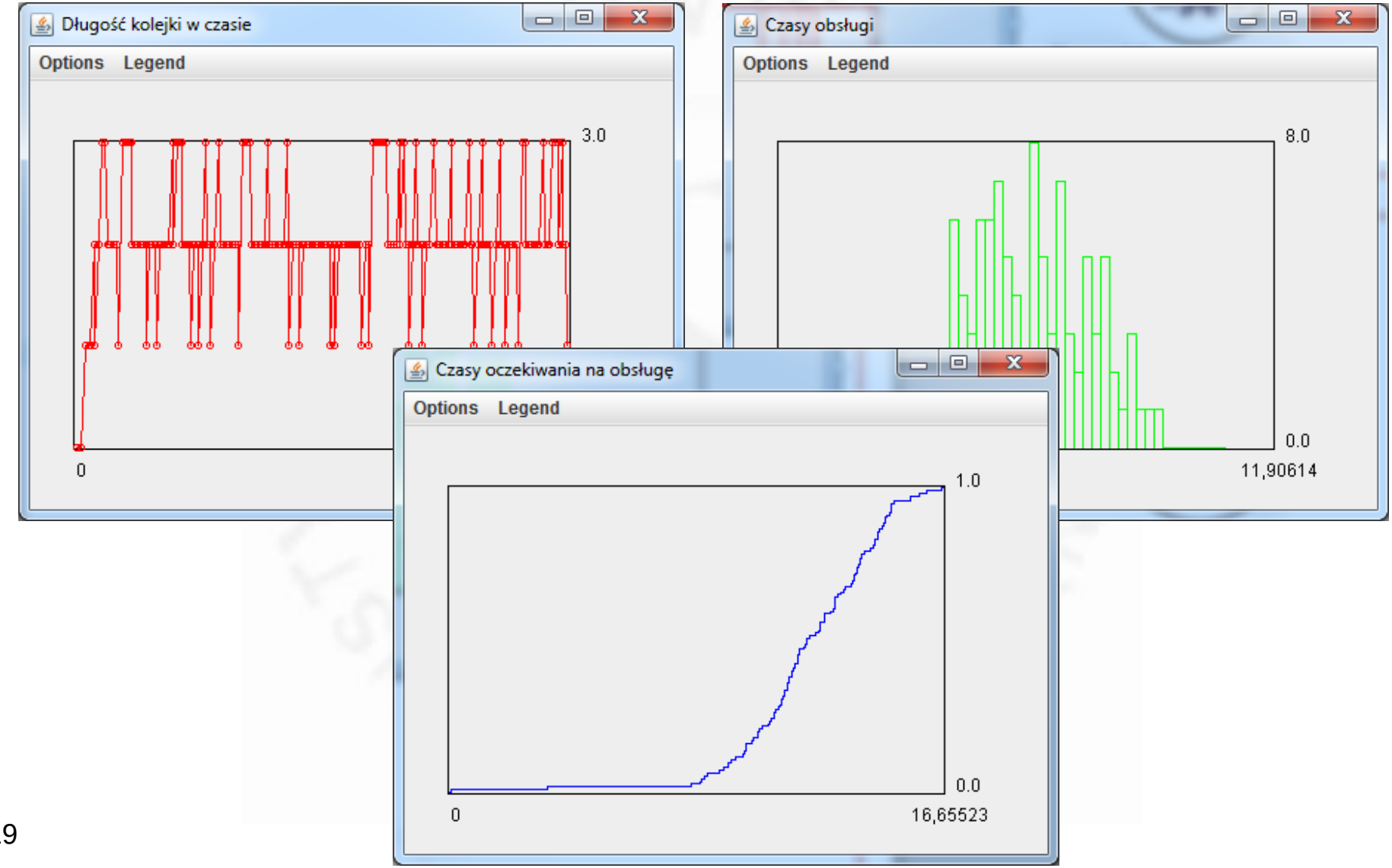
DisSimLab

- Przykład użycia klasy Diagram:

```
/* Zdefiniowanie zmiennych monitorowanych */
MonitoredVar mv1 = new MonitoredVar();
MonitoredVar mv2 = new MonitoredVar();
...
Diagram diagram1 = new Diagram(DiagramType.TIME, "Zmiany stanow ...");
diagram1.add(mv1, java.awt.Color.RED);
diagram1.add(mv2, java.awt.Color.BLUE);
diagram1.show();
Diagram diagram2 = new Diagram(DiagramType.DISTRIBUTION, "Dystrybuanta ...");
diagram2.add(mv1, java.awt.Color.RED);
diagram2.show();
Diagram diagram3 = new Diagram(DiagramType.HISTOGRAM, "Histogram ...");
diagram3.add(mv2, java.awt.Color.RED);
diagram3.show();
```

DisSimLab

- Przykład użycia klasy Diagram:



Zadanie do wykonania

- W celu demonstracji działania symulacji krokowej zasymulować następujący system:
  - Ruch pojazdów na trasie o określonej długości podzielonej na równe odcinki (liczba odcinków ustalana).
  - Pojazdy pojawiają się na początku trasy co losowy czas zgodnie z rozkładem wykładniczym z parametrem `lambda`.
  - Pojazdy na trasie poruszają się ze stałą, wylosowaną na początku prędkością zgodnie z rozkładem równomiernym z przedziału [c, d].
  - Długość pokonanej drogi przez pojazd:

$$s(k) = s(k - 1) + v * dt,$$

$$dt = t(k) - t(k - 1).$$

  - Pojawienie się pojazdu na początku drogi sygnalizowane jest komunikatem zawierającym: [czas symulacyjny zdarzenia], rodzaj zdarzenia i nazwę pojazdu, bieżącą pozycję, długość przebytej drogi, czas jazdy.
  - Dla każdej zmiany dyskretnej pozycji (odcinka) przez pojazd wysyłany jest komunikat zawierający: [czas symulacyjny zdarzenia], rodzaj zdarzenia, nazwę pojazdu, bieżącą pozycję, długość przebytej drogi, czas jazdy.
  - Po dotarciu do końca trasy pojazd parkuje przez losowy czas zgodnie z rozkładem równomiernym z przedziału [e, f], po czym zaczyna jazdę powrotną z nowo wylosowaną prędkością:
    - dotarcie do końca trasy oraz rozpoczęcie jazdy powrotnej sygnalizowane jest komunikatami zawierającymi: [czas symulacyjny zdarzenia], rodzaj zdarzenia, nazwę pojazdu, bieżąca pozycja, długość przebytej drogi, czas jazdy.
  - Po dotarciu do początku trasy pojazd znika z systemu co sygnalizowane jest komunikatem zawierającym: [czas symulacyjny zdarzenia], rodzaj zdarzenia, nazwę pojazdu, bieżącą pozycję, długość przebytej drogi, czas jazdy.

Zadanie do wykonania c.d.

- Stan systemu obejmuje:
  - kolekcje pojazdów na trasie i na parkingu
  - stan symulowanych pojazdów
    - realna (ciągła) przebyta droga: s(k) = s(k – 1) + v * dt, (dt = t(k – 1) – t(k)),
    - bieżąca, dyskretna pozycja na trasie (numer/indeks odcinka).
- Wyznaczyć (oszacować) charakterystyki:
  - średnia liczba pojazdów na trasie (`Statistics.weightedMean`),
  - średni czas przejazdu całej trasy w obie strony (`Statistics.arithmeticMean`).
- Zobrazować na diagramie:
  - przebieg zmian liczby pojazdów na trasie w czasie (`DiagramType.TIME`),
  - dystrybuantę czasu przejazdu (`DiagramType.DISTRIBUTION`).

- Koncepcja rozwiązania:
  - Zdefiniować klasę `Pojazd` jako obiektu poruszającego się po odcinkach trasy:
    - id, prędkość, przebyta droga, nr odcinka (pozycja);
  - Zdefiniować klasę `Trasa` opisującą stan symulowanego systemu.
    - Zdefiniować strukturę (listę) obiektów klasy `Pojazd` poruszających się po trasie,
    - liczba pojazdów na trasie.
  - Zdefiniować trzy klasy zdarzeń repetytywnych poprzez specjalizację klasy `BasicSimEvent`:
    - klasa zdarzenia generującego nowe pojazdy na początku trasy,
    - klasa zdarzenia opisującego ruch pojazdów po trasie,
    - klasa zdarzenia opisującego parkowanie pojazdów.

Przykład definicji klasy obiektu symulacyjnego

```
public class MySimObj extends BasicSimObj {
    /* atrybuty pomocnicze */
    RNGenerator rng;
    /* parametry symulacyjne */
    double minV, maxV, ...;
    / * atrybuty stanu systemu */
    List<Pojazd> pojazdyNaTrasie;
    List<Pojazd> pojazdyNaParkingu;
    MonitoredVar monVar1;
    MonitoredVar monVar2;
    double czasWygenerowaniaNowegoPojazdu;
    ...
    @Override
    public void reflect(IPublisher iPublisher, INotificationEvent iNotificationEvent) {
    }
    @Override
    public boolean filter(IPublisher iPublisher, INotificationEvent iNotificationEvent) {
        return false;
    }
}
```

Przykład definicji klasy zdarzenia repetytywnego

```
public class MyEvent1 extends BasicSimEvent<MySimObj, Object> {
    public MyEvent1(MySimObj entity, Object o, double period)
            throws SimControlException {
        super(entity, o, period);
        /* ... */
    }
    @Override
    protected void stateChange() throws SimControlException {
        MySimObj mySimObj = getSimObj();
        /* ... */
    }
    @Override
    protected void onTermination() throws SimControlException {
    }
    @Override
    public Object getEventParams() {
        return null;
    }
}
```

Przykład uruchomienia symulacji

```
public class Main {
    public static void main(String[] args) {
        SimManager sm = SimManager.getInstance();
        sm.setEndSimTime(1000.0);
        MySimObj mySimObj = new MySimObj(...);
        double krok = 0.1;
        MyEvent1 zd1 = new MyEvent1(mySimObj, null, krok);
        MyEvent2 zd2 = new MyEvent2(mySimObj, null, krok);
        new MyEvent3(mySimObj, null, krok);
        sm.startSimulation();
        /* Na bazie zmiennych monitorowanych wyliczenie i wypisanie charakterystyk */
        System.out.println(„Sr. licz. ...: ” + Statistics.weightedMean(mySimObj.monVar1));
        ...
        /* Na bazie zmiennych monitorowanych zobrazowanie graficzne w postaci diagramów */
        Diagram d1 = new Diagram(DiagramType.TIME, „Liczba ...");
        d1.add(mySimObj.monVar1);
        ...
        d1.show();
    }
}
```