



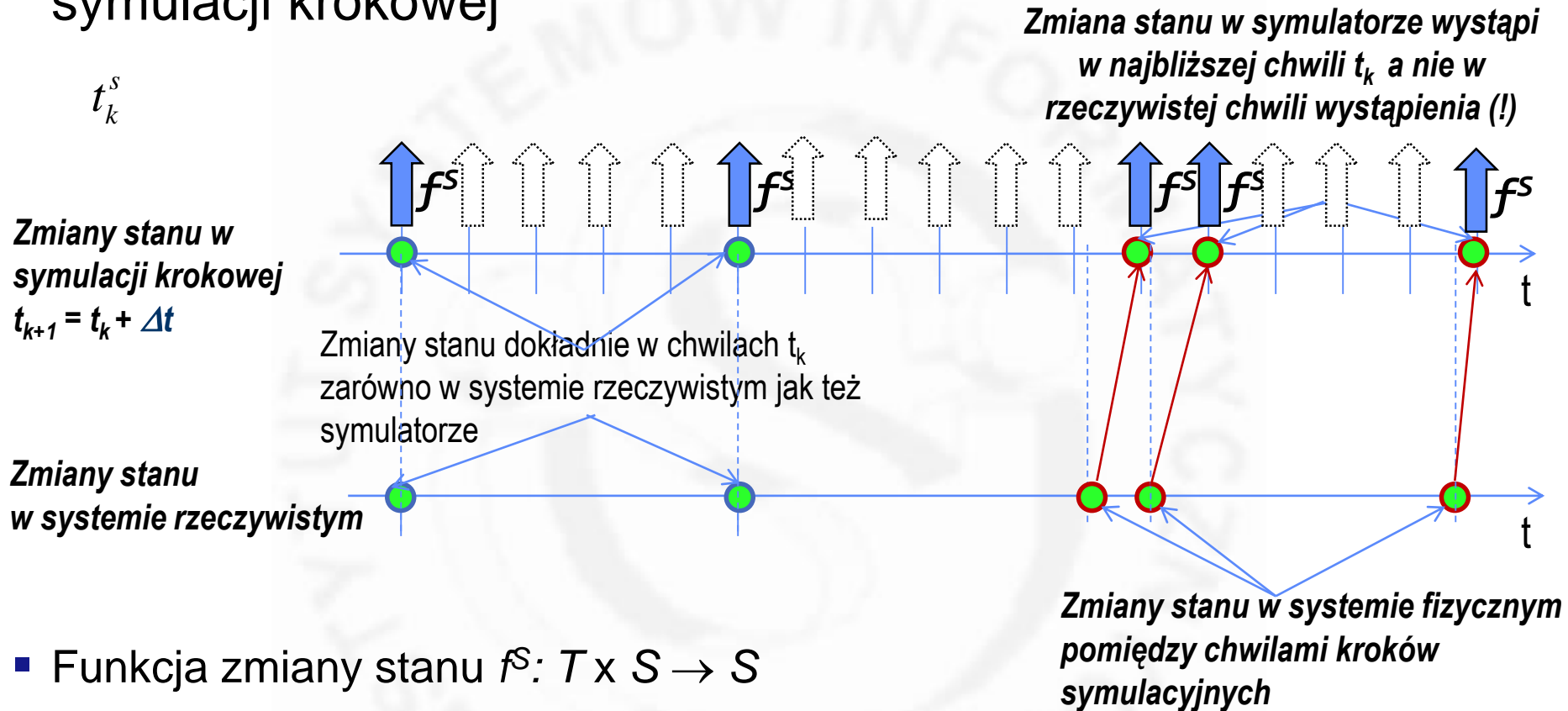
Laboratorium 3

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

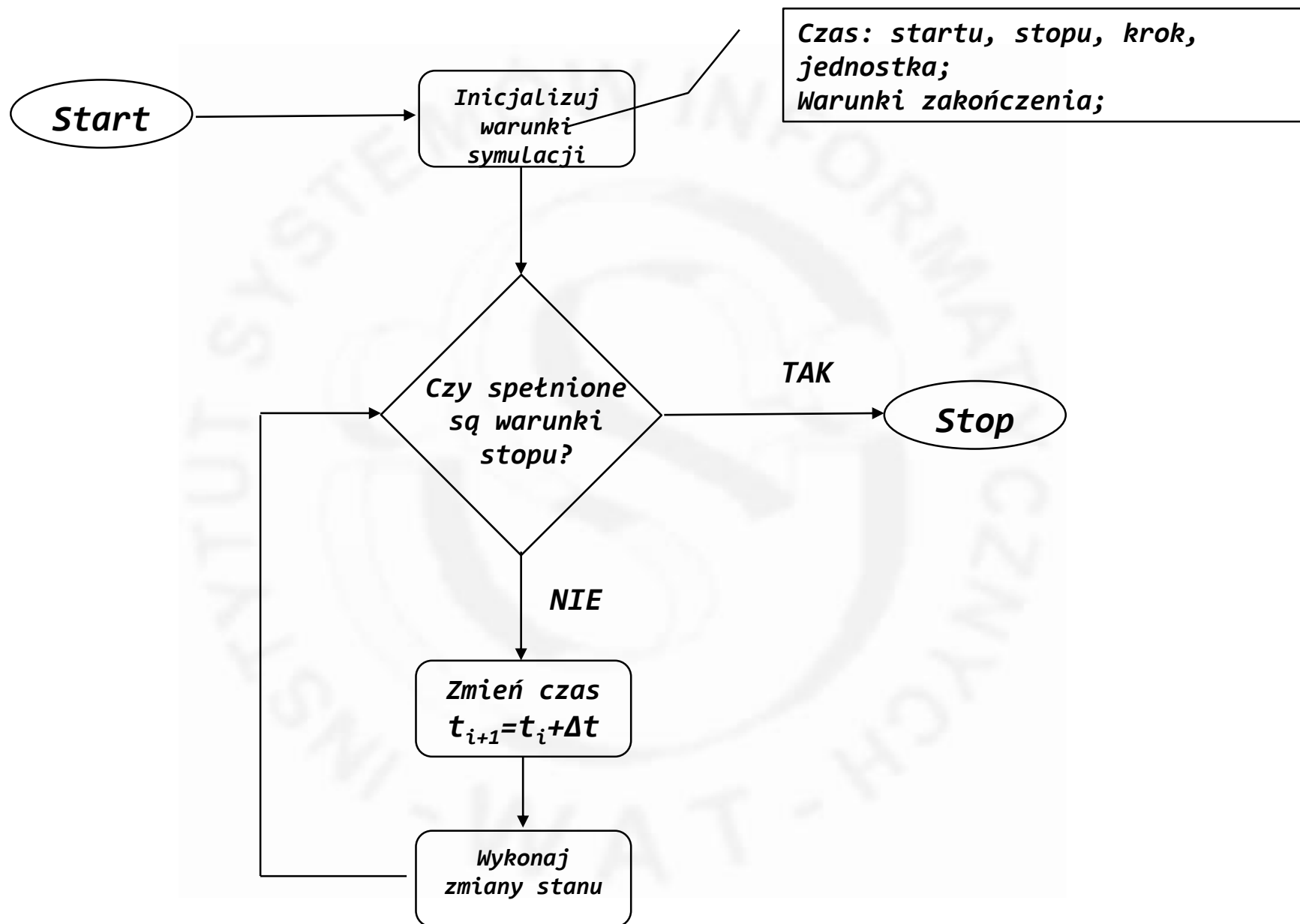
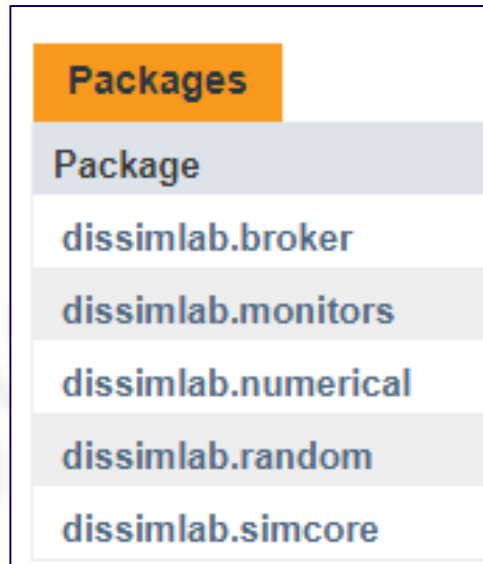


Diagram pakietów



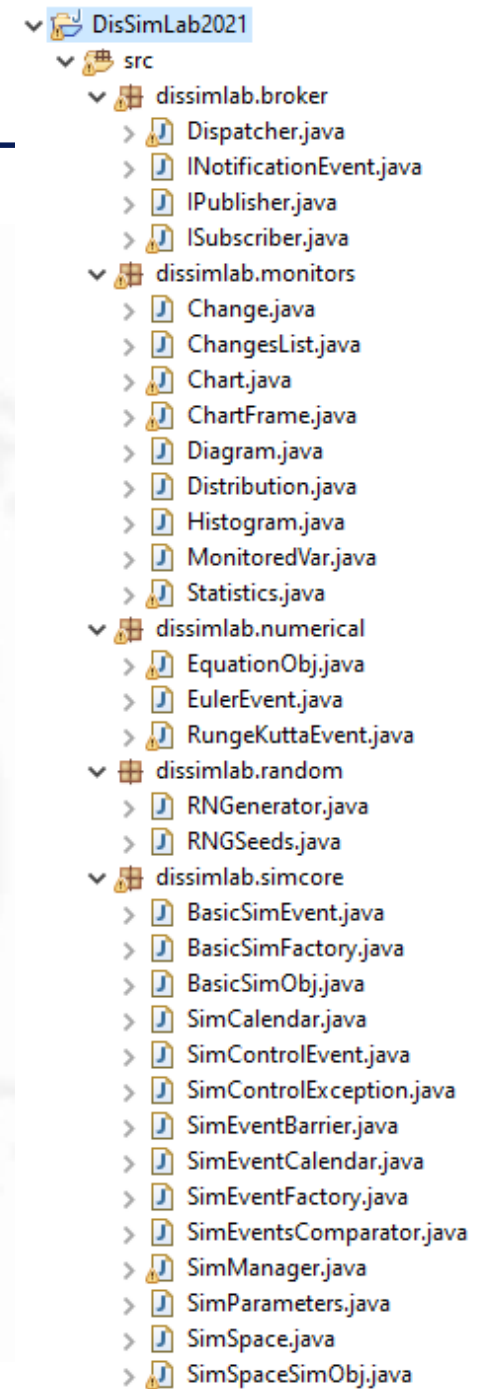
simcore – pakiet grupujący klasy odpowiedzialne za zarządzanie eksperymentem, upływ czasu symulacyjnego, zdarzenia i zmiany stanu;

monitors – klasy odpowiedzialne za monitorowanie, gromadzenie i udostępnienie do analizy statystycznej szeregów czasowych pochodzących ze wskazanych zmiennych programowych;

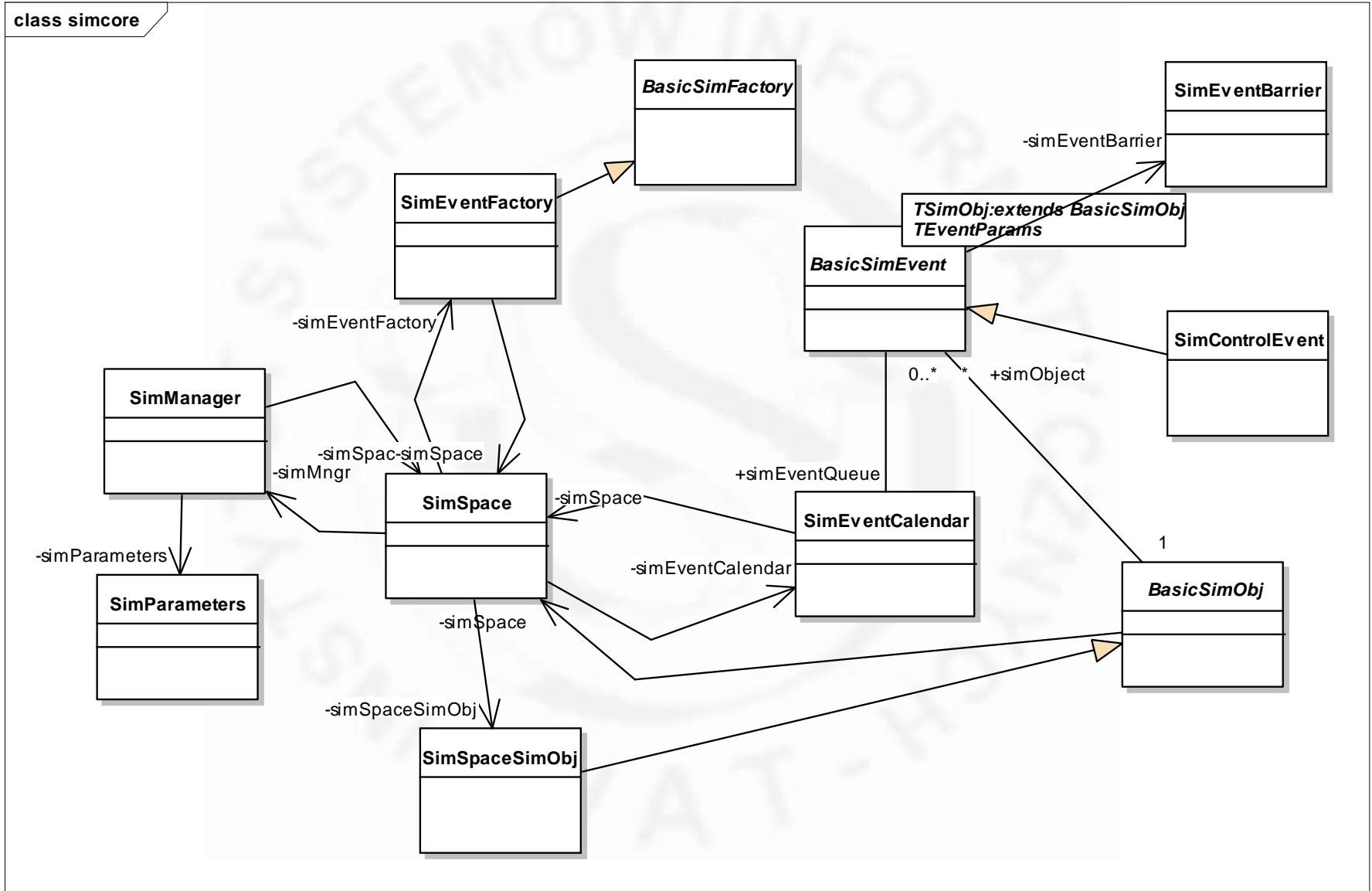
random – pakiet klas generatora liczb (pseudo)losowych;

broker – klasy umożliwiające przesyłanie komunikatów pomiędzy obiektami symulacyjnymi (agentami, środowiskiem);

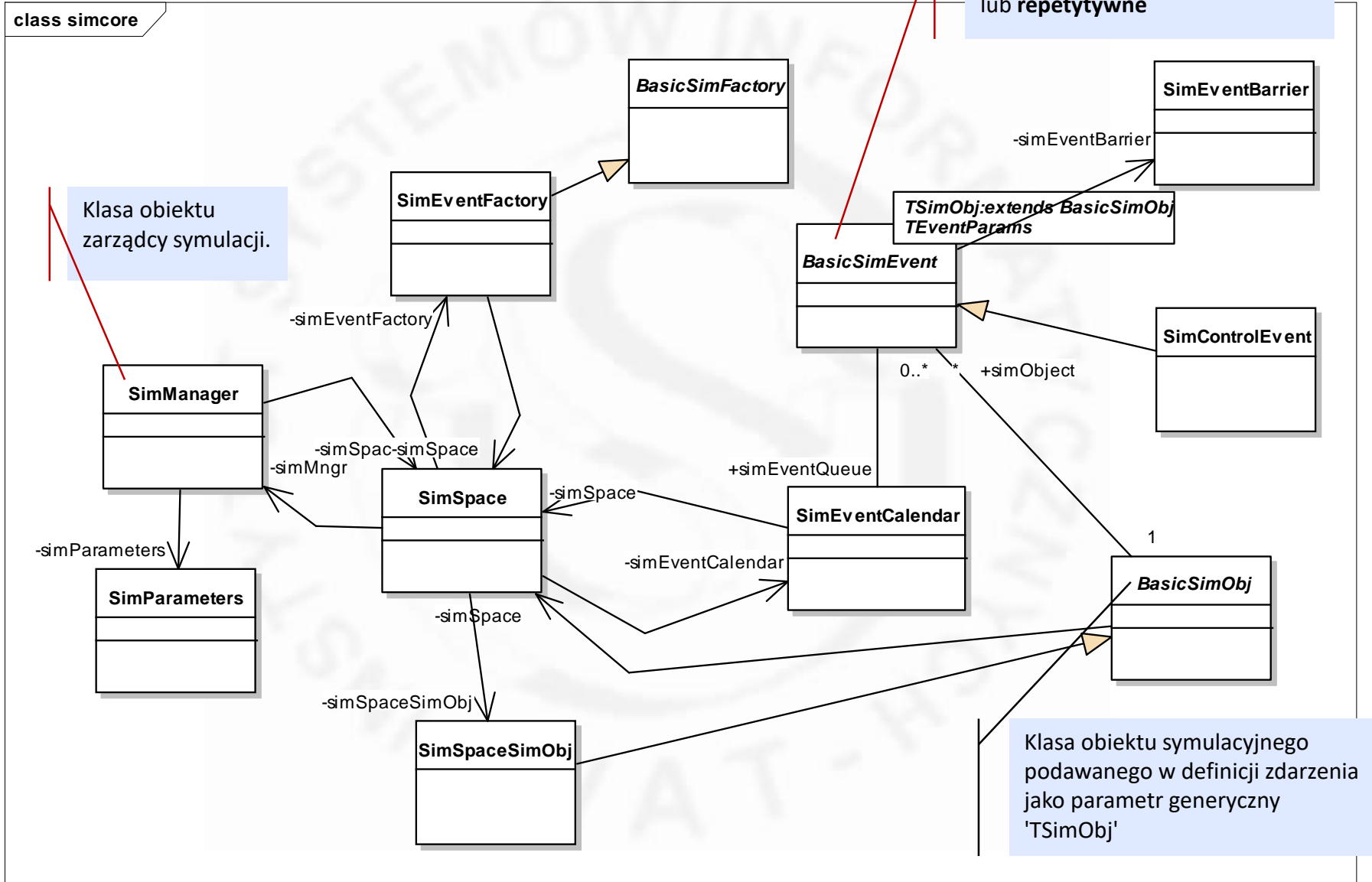
numerical – pakiet klas (jedno-wiele)krokowych metod numerycznych do iteracyjnego przybliżonego rozwiązania równań różniczkowych zwyczajnych, spełniających założenia o istnieniu oraz jednoznaczności rozwiązania.



Główne klasy pakietu „dissimlab.core”



Główne klasy pakietu „dissimlab.core”



SimManager

- astronomicalTimeCorrection: double = 0.0
- astronomicalTimeShift: double = SimParameters.M...
- astronomicalTimeStep: double = SimParameters.d...
- commonDispatcher: Dispatcher
- controlState: SimProcessStatus = SimProcessStatu...
- currentSimTime: double = SimParameters.M...
- endSimTime: double = SimParameters.M...
- eventsProcessed: long = 0
- finishSimTime: double = 0.0
- pauseStartTime: double = 0.0
- simManager: SimManager
- simMode: SimMode = SimMode.ASAP
- simParameters: SimParameters
- simSpace: SimSpace
- simTimeRatio: double = SimParameters.D...
- simTimeScale: double = SimParameters.D...
- stChngCounter: int = 0

- + getCommonDispatcher() : Dispatcher
- + getControlStatus() : SimProcessStatus
- + getEndSimTime() : double
- + getFinishSimTime() : double
- + getInstance() : SimManager
- + getNumberProcessedEvents() : long
- + getSimSpace() : SimSpace
- + getSimTimeRatio() : double
- + getSimTimeScale() : double
- + getSimTimeStep() : double
- + getStChngCounter() : int
- + incStChng() : void
- + initializeSimTime(double) : void
- + initInstance() : SimManager
- ~ nextEvent() : void
- + pauseSimulation() : void
- + resumeSimulation() : void
- setCurrentSimTime(double) : void
- + setEndSimTime(double) : void
- + setSimTimeRatio(double) : void
- + setSimTimeScale(double) : void
- + setSimTimeStep(double) : void
- + simDate(SimParameters.SimDateField) : int
- + SimManager()
- + simTime() : double
- + simTimeFormatted() : String
- + startSimulation() : void
- + stopSimulation() : void

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.

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

BasicSimObj	<i>IPublisher</i> <i>ISubscriber</i>
<pre>- simEventList: LinkedList<BasicSimEvent<BasicSimObj, Object>> - simSpace: SimSpace</pre>	
<pre>~ add(BasicSimEvent<BasicSimObj, Object>) : void + BasicSimObj() + BasicSimObj(SimSpace) ~ createSimEvent(BasicSimEvent<BasicSimObj, Object>, double) : void ~ createSimEvent(BasicSimEvent<BasicSimObj, Object>) : void + getCommonDispatcher() : Dispatcher ~ getFirst() : BasicSimEvent<BasicSimObj, Object> + getSimEventList() : LinkedList<BasicSimEvent<BasicSimObj, Object>> ~ getSimSpace() : SimSpace ~ getSize() : int ~ proceedPauseSimulation() : void ~ proceedRescheduleSimEvent(BasicSimEvent<BasicSimObj, Object>, double) : boolean ~ proceedStopSimulation() : void ~ proceedTerminateSimEvent(BasicSimEvent<BasicSimObj, Object>) : boolean ~ processSimEvent(BasicSimEvent<BasicSimObj, Object>) : void ~ removeAll() : void ~ removeThis(BasicSimEvent<BasicSimObj, Object>) : boolean + simDate(SimParameters.SimDateField) : int + simTime() : double + simTimeFormatted() : String # stopSimulation(double) : void # stopSimulation() : void + terminateAllSimEvents() : void</pre>	

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>

```

class BasicSimEvent<TSimObj extends BasicSimObj, TEventParams> implements INotificationEvent<TEventParams> {
    # eventParams: TEventParams = null
    - publishable: boolean = false
    - repetitionPeriod: double = 0.0
    - runTime: double
    - simEventBarrier: SimEventBarrier = null
    - simObject: TSimObj = null
    - simPriority: int = SimParameters.D...
    ~ simStatus: SimEventStatus

    + BasicSimEvent()
    + BasicSimEvent(double)
    + BasicSimEvent(TEventParams)
    + BasicSimEvent(double, TEventParams)
    + BasicSimEvent(TEventParams, int)
    + BasicSimEvent(double, TEventParams, int)
    + BasicSimEvent(TSimObj)
    + BasicSimEvent(TSimObj, double)
    + BasicSimEvent(TSimObj, double, TEventParams)
    + BasicSimEvent(TSimObj, double, int)
    + BasicSimEvent(TSimObj, double, TEventParams, int)
    + BasicSimEvent(TSimObj, SimEventBarrier, TEventParams)
    + BasicSimEvent(TSimObj, SimEventBarrier, TEventParams, int)
    + BasicSimEvent(TEventParams, double)
    + BasicSimEvent(TSimObj, TEventParams, double)
    + BasicSimEvent(TEventParams, double, int)
    + BasicSimEvent(TSimObj, TEventParams, double, int)
    # getCommonDispatcher() : Dispatcher
    + getRepetitionPeriod() : double
    + getRunTime() : double
    + getSimEventBarrier() : SimEventBarrier
    + getSimObj() : TSimObj
    + getSimPriority() : int
    - getSimStatus() : SimEventStatus
    + isPublishable() : boolean
    # onTermination() : void
    ~ processState() : void
    + reschedule(double) : boolean
    + setPublishable(boolean) : void
    + setRepetitionPeriod(double) : void
    ~ setRunTime(double) : void
    ~ setSimEventBarrier(SimEventBarrier) : void
    ~ setSimObj(TSimObj) : void
    ~ setSimStatus(SimEventStatus) : void
    + simDate(SimParameters.SimDateField) : int
    + simTime() : double
    + simTimeFormatted() : String
    # stateChange() : void
    # stopSimulation(double) : void
    # stopSimulation() : void
    + terminate() : boolean
    + toString() : String
}
    
```

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.

BasicSimEvent()

Konstruktor domyślny.

BasicSimEvent(double delay)

Konstruktor tworzący zdarzenie dla obiektu `simSpaceSimObj` powołanego jako wspólny obiekt w symulacji.

BasicSimEvent(double delay, TEventParams params)

Konstruktor tworzący zdarzenie dla obiektu `simSpaceSimObj` powołanego jako wspólny obiekt w symulacji.

BasicSimEvent(double delay, TEventParams params, int priority)

Konstruktor tworzący zdarzenie dla obiektu `simSpaceSimObj` powołanego jako wspólny obiekt w symulacji.

BasicSimEvent(SimEventBarrier barrier, TEventParams params)

Konstruktor tworzący odłożone w czasie zdarzenie dla obiektu `simSpaceSimObj` powołanego jako wspólny obiekt w symulacji.

BasicSimEvent(TEventParams params)

Konstruktor tworzący natychmiastowe zdarzenie dla obiektu `simSpaceSimObj` powołanego jako wspólny obiekt w symulacji.

BasicSimEvent(TEventParams params, double period)

Konstruktor tworzący repetytywne zdarzenie dla obiektu `simSpaceSimObj` powołanego jako wspólny obiekt w symulacji.

BasicSimEvent(TEventParams params, double period, int priority)

Konstruktor tworzący repetytywne zdarzenie dla obiektu `simSpaceSimObj` powołanego jako wspólny obiekt w symulacji.

`BasicSimEvent(TEventParams params, int priority)`

Konstruktor tworzący natychmiastowe zdarzenie dla obiektu `simSpaceSimObj` powołanego jako wspólny obiekt w symulacji.

`BasicSimEvent(TSimObj entity)`

Konstruktor tworzący natychmiastowe zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

`BasicSimEvent(TSimObj entity, double delay)`

Konstruktor tworzący zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

`BasicSimEvent(TSimObj entity, double delay, int priority)`

Konstruktor tworzący zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

`BasicSimEvent(TSimObj entity, double delay, TEventParams params)`

Konstruktor tworzący zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

`BasicSimEvent(TSimObj entity, double delay, TEventParams params, int priority)`

Konstruktor tworzący zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

`BasicSimEvent(TSimObj entity, SimEventBarrier barrier)`

Konstruktor tworzący odłożone w czasie zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

`BasicSimEvent(TSimObj entity, SimEventBarrier barrier, TEventParams params)`

Konstruktor tworzący odłożone w czasie zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

`BasicSimEvent(TSimObj entity, TEventParams params, double period)`

Konstruktor tworzący repetytywne zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

`BasicSimEvent(TSimObj entity, TEventParams params, double period, int priority)`

Konstruktor tworzący repetytywne zdarzenie dla obiektu `'entity'` (obiekt musi istnieć).

```
public class RNGenerator  
extends java.util.Random
```

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

Metody generujące liczbę pseudolosową jako realizację różnych rozkładów

class random

RNGenerator		Random
~ cof: double ([]) = { 76.18009173, ...		
+ PI: double = 3.1415926535897... {readOnly}		
- serialVersionUID: long = 1L {readOnly}		
+ beta(double, double) : double		
+ binomial(double, int) : double		
+ chisquare(int) : double		
+ erlang(int, double) : double		
+ exponential(double) : double		
+ fdistribution(int, int) : double		
+ gamma(double, double) : double		
+ generateSeed() : long		
+ geometric(double) : double		
- lngamma(double) : double		
+ lognormal(double, double) : double		
+ normal(double, double) : double		
+ poisson(double) : double		
+ probability(double) : boolean		
+ RNGenerator(long)		
+ RNGenerator()		
+ student(int) : double		
+ triangular(double) : double		
+ uniform(double, double) : double		
+ uniformInt(int) : int		
+ uniformInt(int, int) : int		
+ weibull(double, double) : double		

RNGSeeds

+ ClockSeed(Date) : long
+ ClockSeed() : long

■ Przykład generowania liczb pseudolosowych:

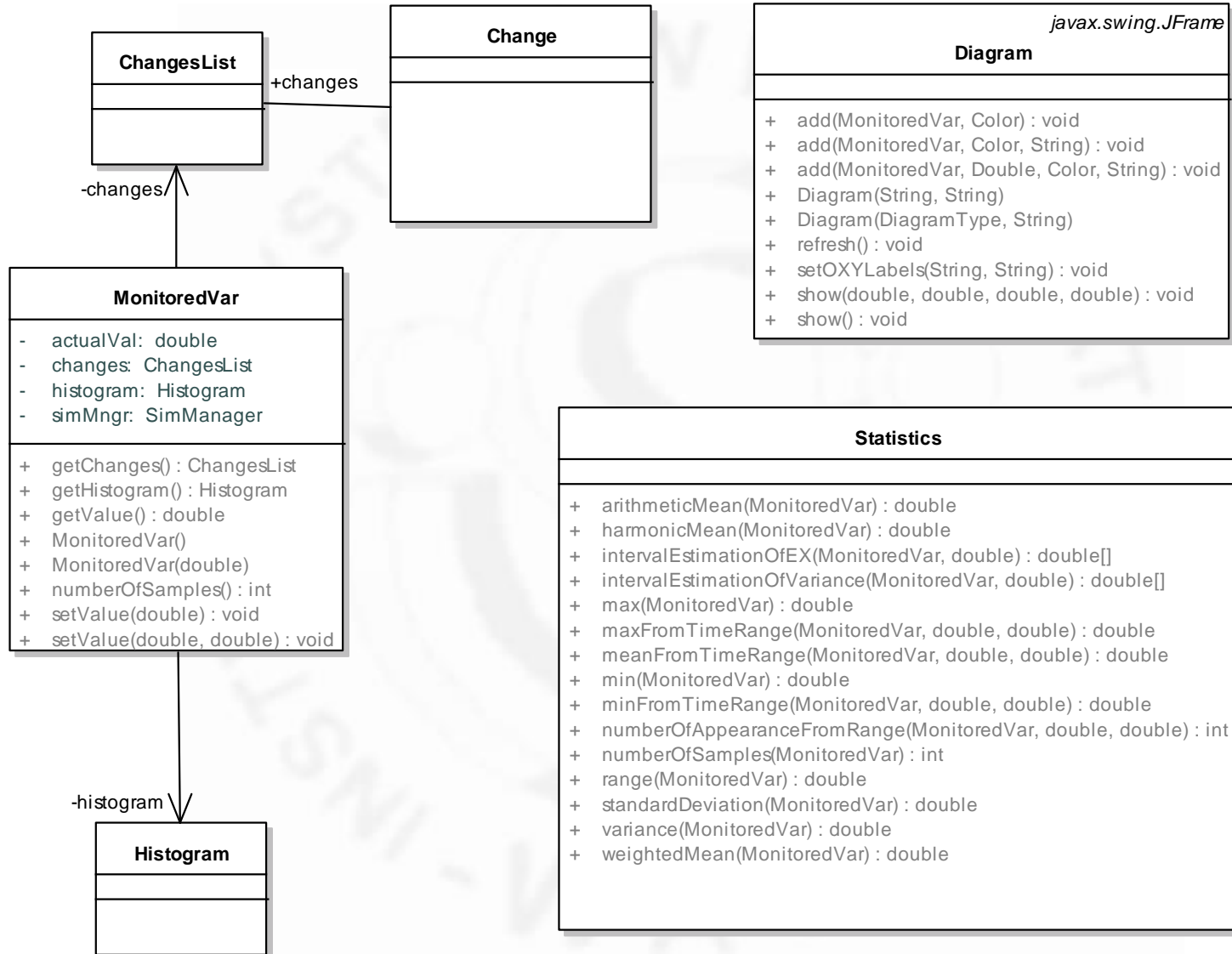
```
RNGenerator sg = new RNGenerator();          /* utworzenie nowego
generatora bez podania ziarna */

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

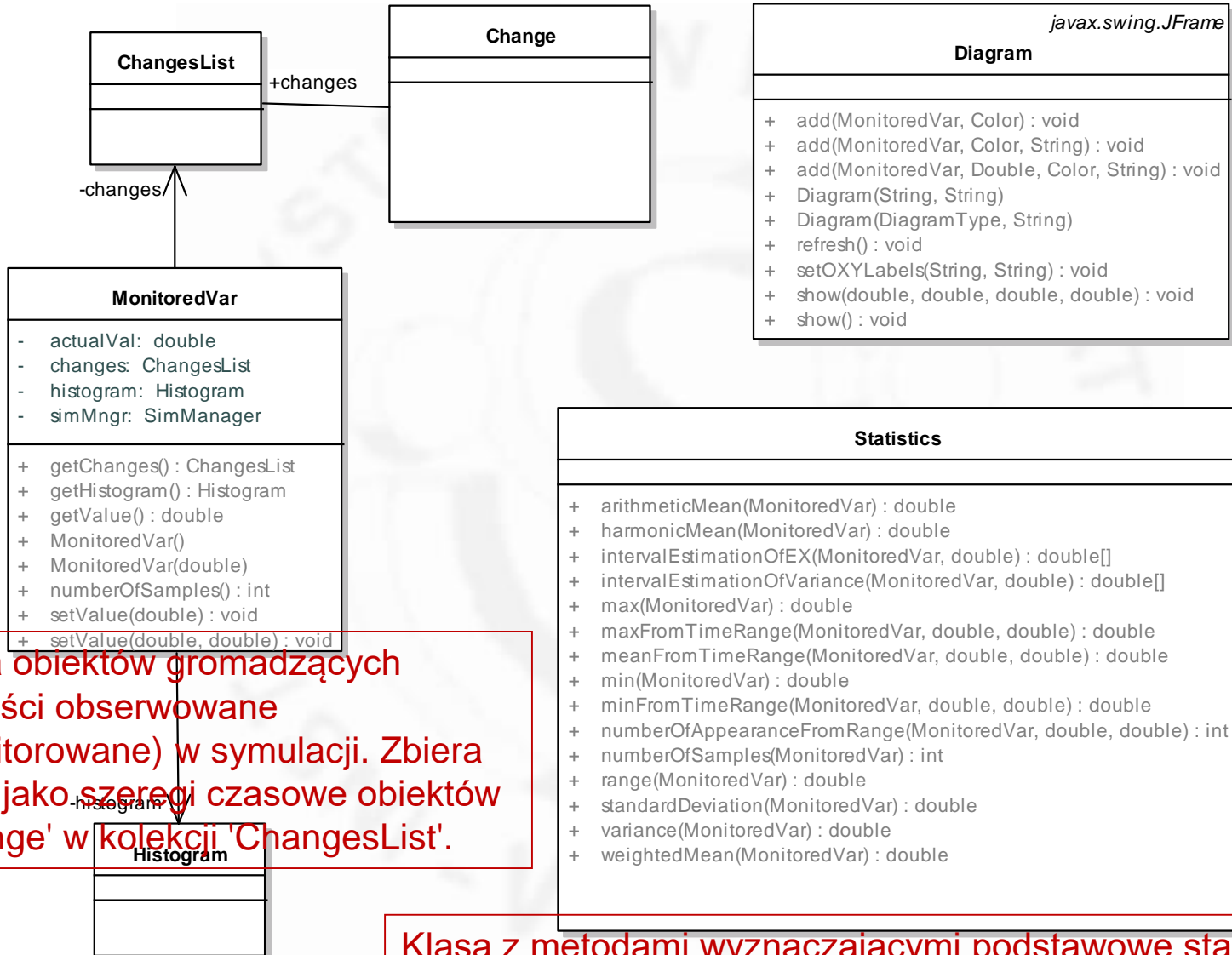
RNGenerator sg2 = new RNGenerator(seed); /* utworzenie nowego
generatora liczb pseudolosowych z podaniem ziarna jako
parametru */

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

class monitors



class monitors

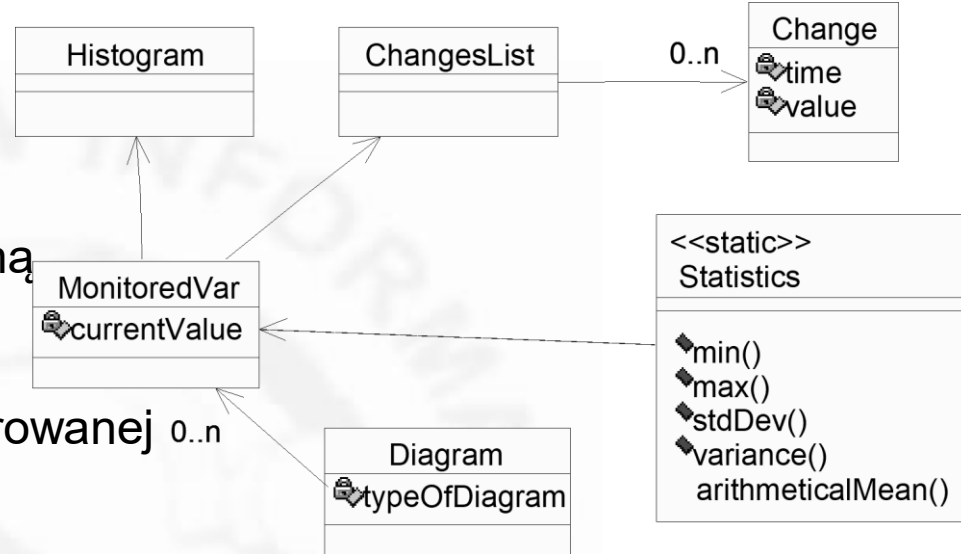


Klasa obiektów gromadzących wartości obserwowane (monitorowane) w symulacji. Zbiera dane jako szeregi czasowe obiektów 'Change' w kolekcji 'ChangesList'.

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

Diagram klas pakietu Monitors:

- MonitoredVar
 - reprezentująca zmienną monitorowaną
- ChangeList
 - listę zmian wartości zmiennej monitorowanej 0..n
- 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;



■ 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);

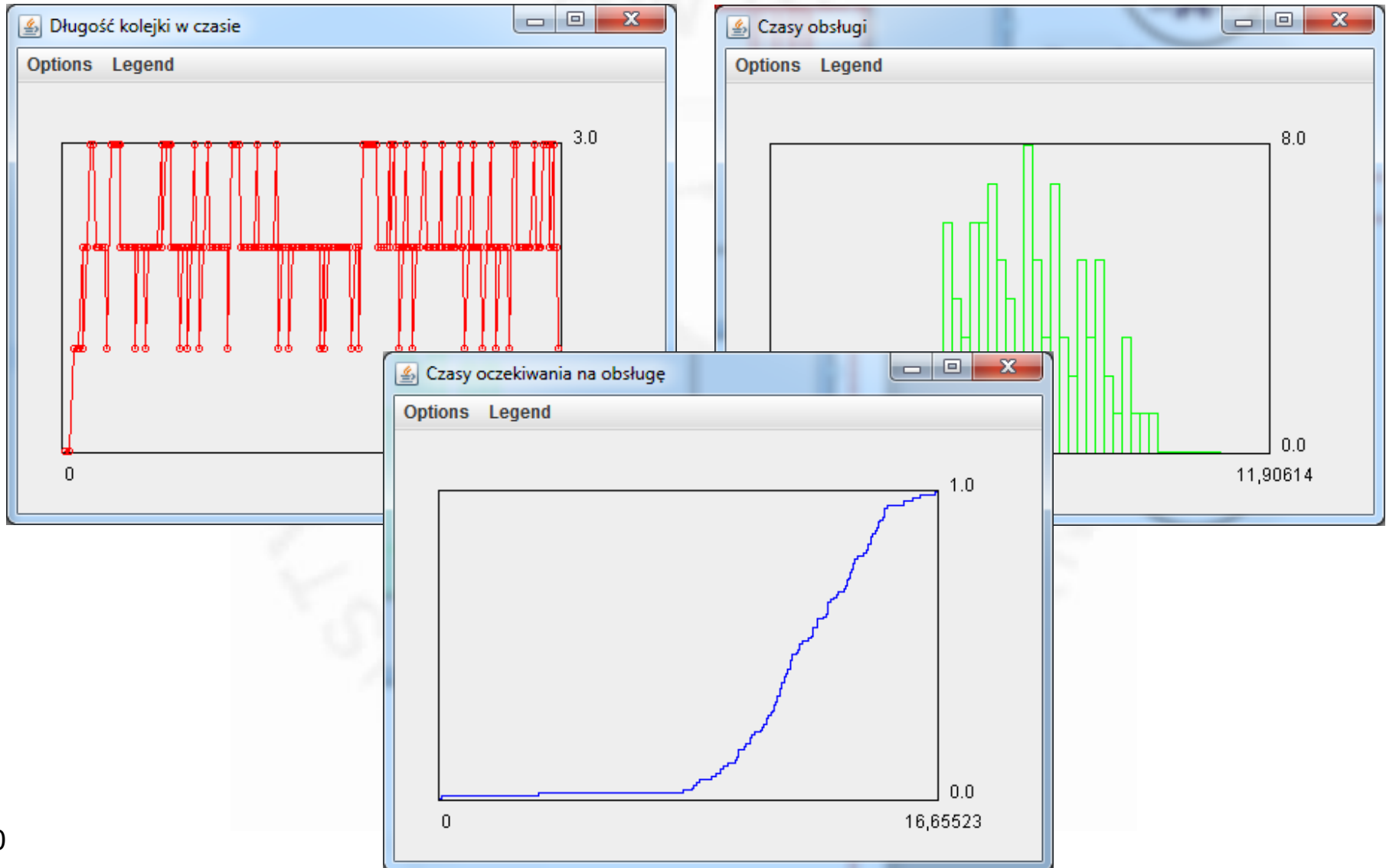
/* estymacja przedziałowa dla wartości oczekiwanej, wyliczenie
przedziału ufności dla zmiennej EX i gamma równego 0.9, jako
wynik zwracane są dwie liczby w tablicy dwuelementowej, z
których pierwsza jest lewym krańcem przedziału ufności, druga
natomiast prawym krańcem przedziału ufności*/

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

■ Przykład użycia klasy Diagram:

```
/* Zdefiniowanie zmiennych monitorowanych */  
MonitoredVar mv1 = new MonitoredVar();  
MonitoredVar mv2 = new MonitoredVar();  
...  
Diagram diagram1 = new Diagram(DiagramType.TIME, "Zmiany stanów ...");  
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();
```

■ Przykład użycia klasy Diagram:



Zadanie do wykonania

- W celu demonstracji działania symulacji krokowej zasymulować następujący system:
 - Ruch pieszych po szlaku o określonej długości podzielonego na równe odcinki (liczba odcinków ustalana).
 - Piesi pojawiają się na początku szlaku co losowy czas zgodnie z rozkładem równomiernym z przedziału $[a, b]$.
 - Piesi na szlaku poruszają się ze stałą, wylosowaną na początku prędkością zgodnie z rozkładem równomiernym z przedziału $[c, d]$ (jednostka: liczba odcinków na sekundę).
 - Droga przebyta przez pieszego na szlaku:

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

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

- Pojawienie się pieszego na początku szlaku sygnalizowane jest komunikatem zawierającym: [czas symulacyjny zdarzenia], rodzaj zdarzenia i nazwę obiektu.
- W każdym kroku symulacji wysyłany jest komunikat zawierający: [czas symulacyjny zdarzenia], rodzaj zdarzenia, nazwę obiektu, bieżącą pozycję, przebytą drogę.
- Po dotarciu do końca szlaku pieszy znika z systemu, co sygnalizowane jest komunikatem zawierającym: [czas symulacyjny zdarzenia], rodzaj zdarzenia, nazwę obiektu, przebytą drogę, czas przemarszu.

Zadanie do wykonania c.d.

- Stan systemu obejmuje:
 - kolekcję pieszych na szlaku,
 - stan symulowanych pieszych
 - realna (ciągła) przebyta droga: $s(k) = s(k - 1) + v * dt$, ($dt = t(k - 1) - t(k)$),
 - bieżąca, dyskretna pozycja na szlaku (numer odcinka).
- Wyznaczyć (oszacować) charakterystyki:
 - średnia liczba obiektów na szlaku (`Statistics.weightedMean`),
 - średni czas przemarszu (`Statistics.arithmeticMean`).
- Zobrazować na diagramie:
 - przebieg zmian liczby pieszych na szlaku w czasie (`DiagramType.TIME`),
 - dystrybuantę czasu przemarszu (`DiagramType.DISTRIBUTION`).
- Koncepcja rozwiązania:
 - Zdefiniować klasę `Pieszy` jako obiektu poruszającego się po odcinkach szlaku:
 - id, prędkość, przebyta droga, nr odcinka (pozycja);
 - Zdefiniować klasę `Szlak` opisującą stan symulowanego systemu.
 - Zdefiniować strukturę (listę) obiektów klasy `Pieszy` poruszających się po szlaku,
 - liczba pieszych na szlaku.
 - Zdefiniować dwie klasy zdarzeń repetytywnych poprzez specjalizację klasy `BasicSimEvent`:
 - klasa zdarzenia opisującego ruch pieszych po szlaku,
 - klasa zdarzenia generującego nowych pieszych.

Przykład definicji klasy obiektu symulacyjnego

```
public class Szlak extends BasicSimObj {
    /* atrybuty pomocnicze */
    RNGenerator rng;
    /* parametry symulacyjne */
    double minV, maxV, ...;
    / * atrybuty stanu systemu */
    List<Pieszy> listaPiesznychNaSzlaku;
    MonitoredVar czasPrzemarszu;
    MonitoredVar liczbaPiesznychNaSzlaku;
    double czasPojawieniaSieNowegoPieszego;
    ...
    @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<Szlak, Object> {  
    public MyEvent1(MySimObj entity, Object o, double period)  
        throws SimControlException {  
        super(entity, o, period);  
        /* ... */  
    }  
    @Override  
    protected void stateChange() throws SimControlException {  
        Szlak szlak = 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);  
        //      SimControlEvent stopEvent = new SimControlEvent(1000.0,  
        //      SimControlStatus.STOPSIMULATION);  
        Szlak szlak = new Szlak(...);  
        double krok = 0.1;  
        MyEvent1 zd1 = new MyEvent1(szlak, null, krok);  
        MyEvent2 zd2 = new MyEvent2(szlak, null, krok);  
        sm.startSimulation();  
        /* Na bazie zmiennych monitorowanych wyliczenie i wypisanie charakterystyk */  
        System.out.println("Sr. licz. piesz.: " + Statistics.weightedMean(szlak.liczPiesz));  
        ...  
        /* Na bazie zmiennych monitorowanych zobrazowanie graficzne w postaci diagramów */  
        Diagram d1 = new Diagram(DiagramType.TIME, "Liczba pieszych ...");  
        d1.add(so.liczPiesz);  
        d1.show();  
    }  
}
```