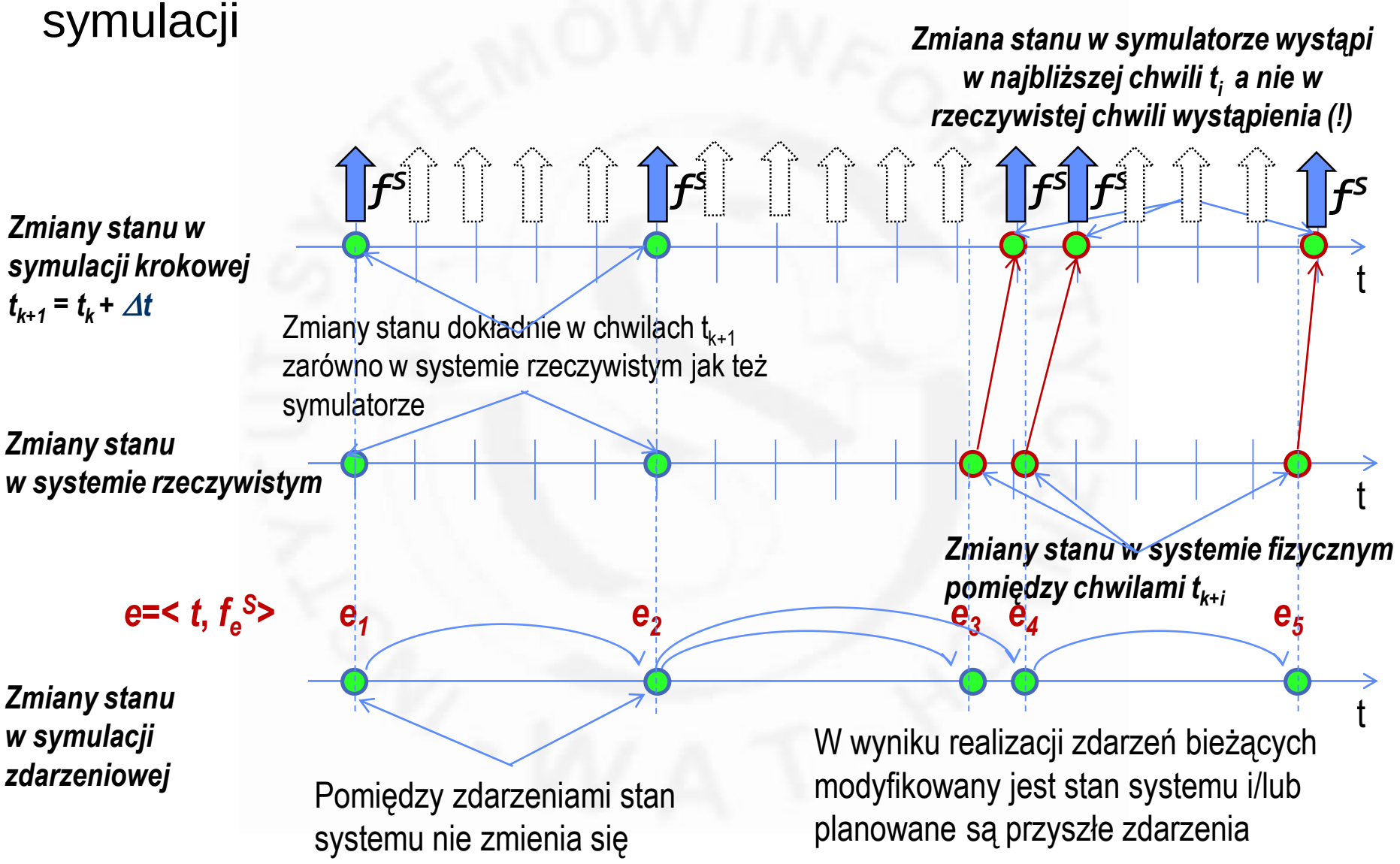
Laboratorium 4
Symulacja zdarzeniowa

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

Symulacja zdarzeniowa - koncepcja

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



Planowanie zdarzeń

- Pod pojęciem zdarzenia *e* należy rozumieć zaplanowaną algorytmicznie zmianę stanu obiektu/systemu w określonej chwili czasu symulacyjnego: $e=< t, f_e^S >, e \in E, t \in T$;
- Skutki zdarzenia obejmują:
  - zmianę stanu obiektu/systemu;
  - opcjonalne zaplanowanie nowych zdarzeń jako konsekwencji nowego stanu systemu;
- Zdarzenia opisane różnymi funkcjami zmiany stanu to zdarzenia różnych klas;
- W symulacji opartej na zdarzeniach przyjmuje się następujące uproszczenie modelowe – stan systemu nie ulega zmianie do czasu realizacji kolejnego zdarzenia;

Planowanie zdarzeń

- Zdarzenia muszą zachodzić w kolejności wynikającej z chwili ich zajścia (realizacji) – w tym celu definiuje się kolekcję zdarzeń (zwaną „kalendarzem zdarzeń”), przechowującą uporządkowane chronologicznie zdarzenia, które oczekują na realizację/zajście;
- Zdarzenia w kolekcji są uporządkowane wg wartości czasu symulacyjnego planowanego ich zajścia;
- Wstawianie nowego zdarzenia do kalendarza zdarzeń odbywa się z zachowaniem porządku;
- Jednoczesność zdarzeń w danej chwili czasu:
  - kolejność ich obsługi w symulacji sekwencyjnie realizowanej nie powinna mieć znaczenia dla wyników symulacji;

Planowanie zdarzeń – rodzaje zdarzeń

- Z punktu widzenia czasu zajścia, zdarzenia mogą być:
  - bezwarunkowe (najczęstszy rodzaj zdarzeń):
    - określony (znany) czas zajścia zdarzenia,
    - zdarzenie trafia do kalendarza zdarzeń,
  - warunkowe:
    - nieznany czas zajścia zdarzenia,
    - zajście zdarzenia uwarunkowane określoną zmianą stanu (zależne od innych zdarzeń),
    - zdarzenie przechowywane poza kalendarzem w dedykowanej strukturze,
    - wyzwalane przez inne zdarzenie trafia do kalendarza z bieżącym czasem realizacji;

Planowanie zdarzeń

- Przed rozpoczęciem symulacji musi być zaplanowane co najmniej jedno zdarzenie bezwarunkowe;
- Podtrzymanie symulacji wymaga umieszczania w kalendarzu nowych zdarzeń (planowanie zdarzeń);
- Bieżący czas symulacyjny równy jest wartości czasu pierwszego zdarzenia;
- Zakończenie symulacji może nastąpić z chwilą:
  - upływu przewidzianego czasu symulacji;
  - obsłużenia wszystkich zdarzeń z kalendarza zdarzeń;
  - wystąpienia ustalonej liczby zdarzeń;
  - wystąpienia określonego zdarzenia;
  - zarejestrowania określonej liczby obserwacji;

Planowanie zdarzeń – algorytm

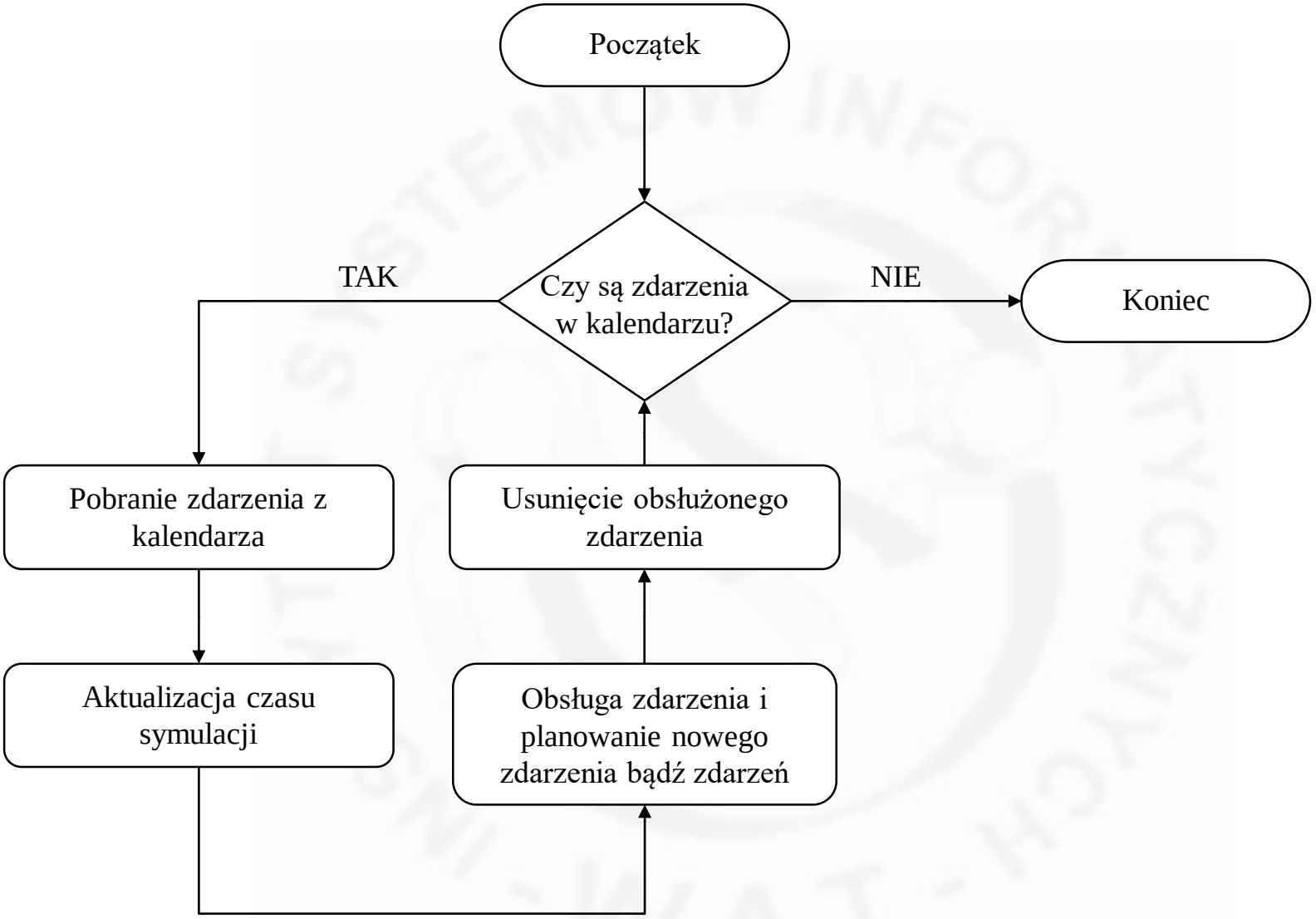
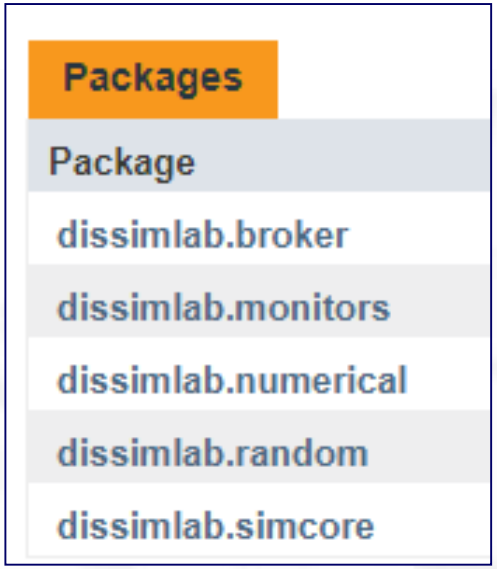
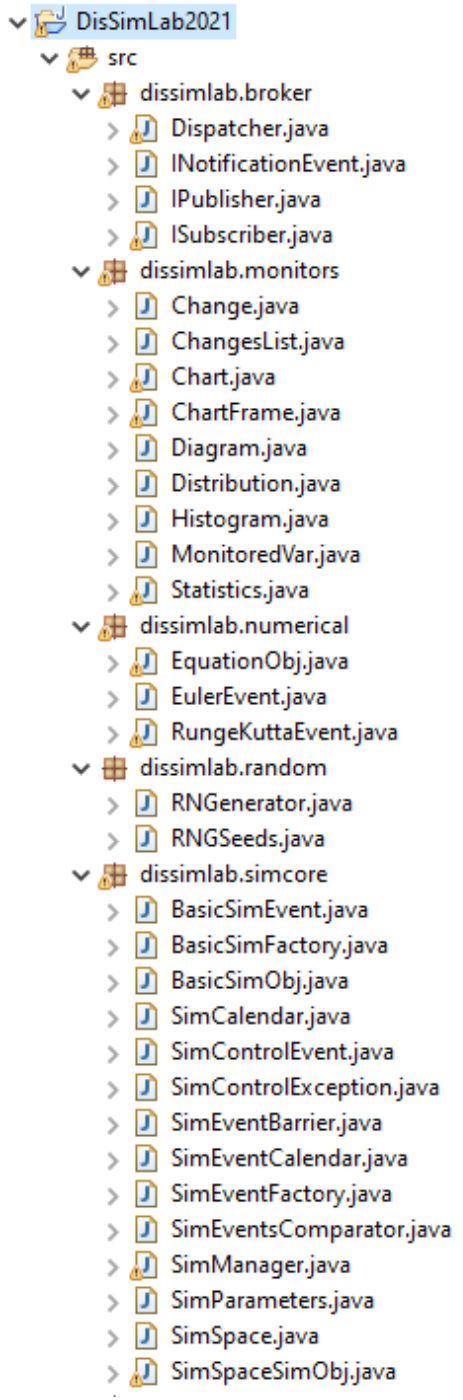


Diagram pakietów

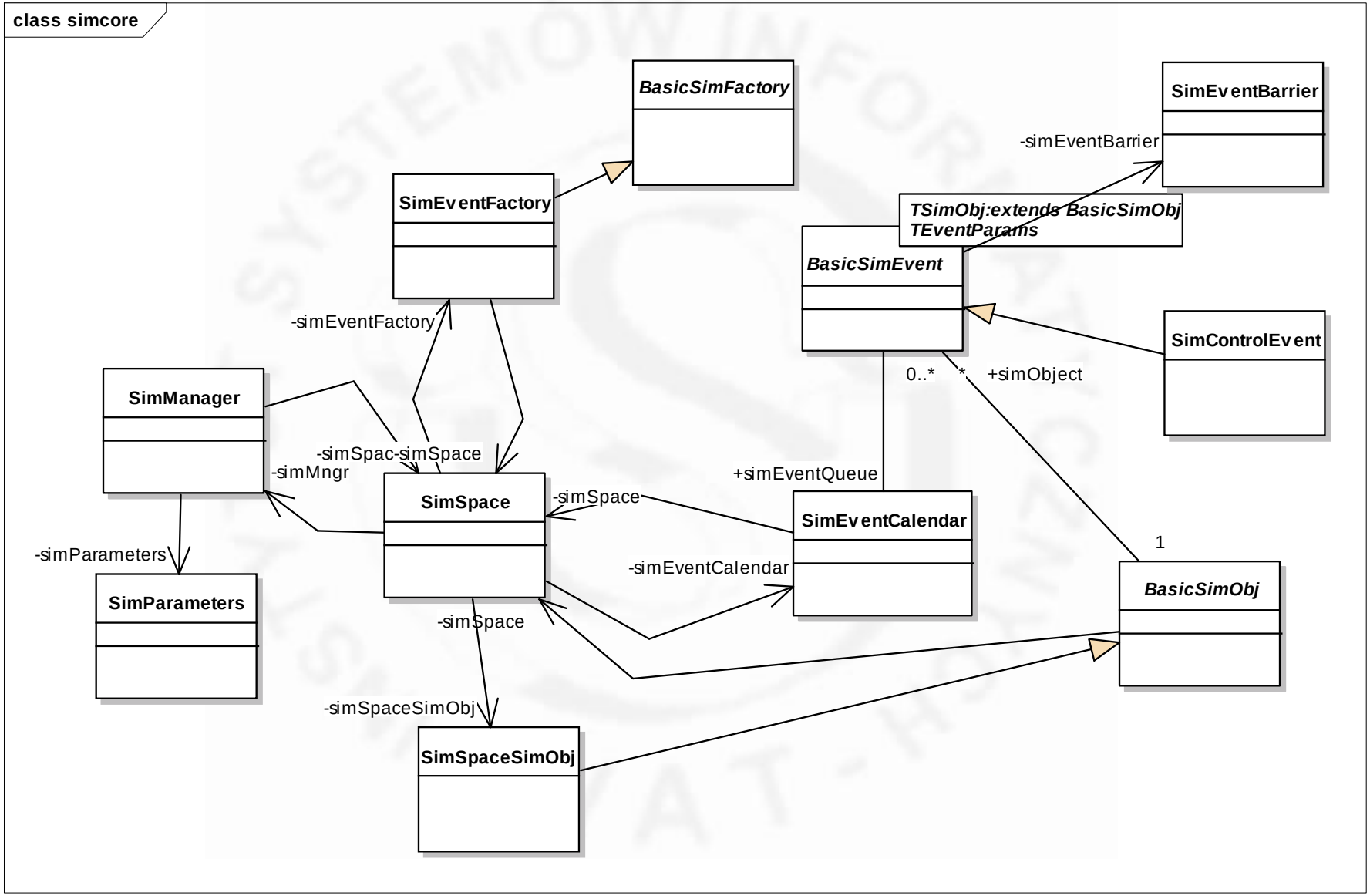




simcore – pakiet grupujący klasy odpowiedzialne za zarządzanie eksperymentem, upływ czasu symulacyjnego, zdarzenia i zmiany stanu;
monitors – klasy odpowiedzialne za monitorowanie, gromadzenie i udostępnienie do analizy statystycznej szeregów czasowych pochodzących ze wskazanych zmiennych programowych;
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.

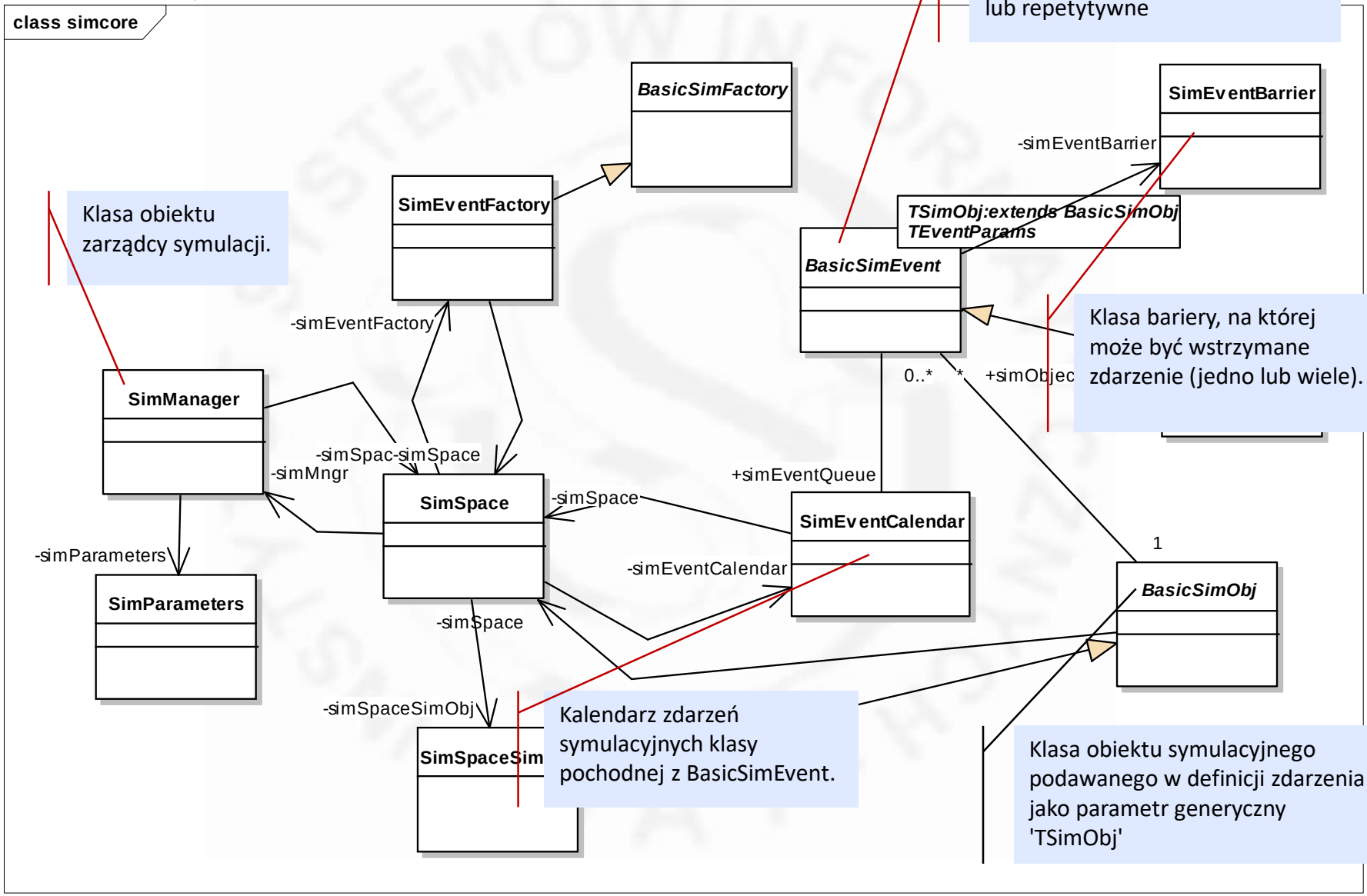
DisSimLab

Główne klasy pakietu „dissimlab.core”



DisSimLab

Główne klasy pakietu „dissimlab.core”



Klasa zdarzenia symulacyjnego, realizowanego jako jednorazowe lub repetytywne

Klasa obiektu zarządcy symulacji.

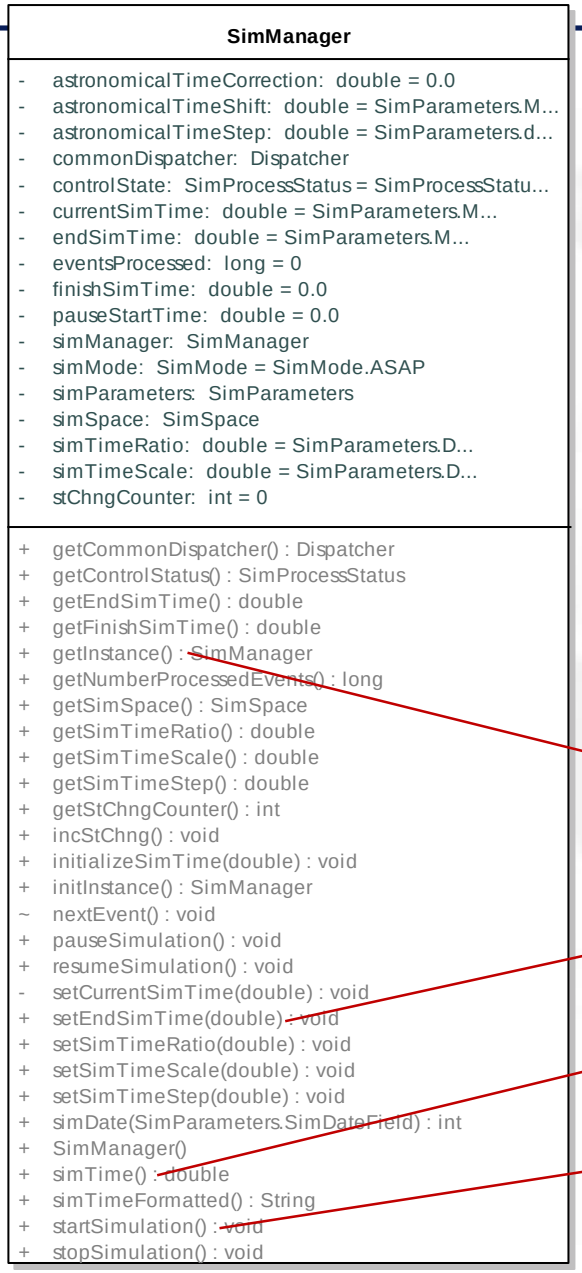
Klasa bariery, na której może być wstrzymane zdarzenie (jedno lub wiele).

Kalendarz zdarzeń symulacyjnych klasy pochodnej z BasicSimEvent.

Klasa obiektu symulacyjnego podawanego w definicji zdarzenia jako parametr generyczny 'TSimObj'

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 daty lub godziny, wyliczoną w odniesieniu do umownego początku symulacji.

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 (bezwarunkowe) lub na tzw. barierze (warunkowe). 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 abstrakcyjna, która powinna być nadpisana dla zdefiniowania zmiany stanu zamiast metody 'stateChange'.

Metoda ustawia wartość okresu czasu symulacyjnego, z jakim zaplanowane zdarzenie będzie cyklicznie powtarzane.

Metoda abstrakcyjna, która powinna być nadpisana dla zdefiniowania zmiany stanu w wyniku zdarzenia.

DisSimLab

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.

DisSimLab

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, TEventParams params)

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(TEventParams params, int priority)

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

BasicSimEvent(TSimObj entity, double delay, int priority)

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, TEventParams params, double period)

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

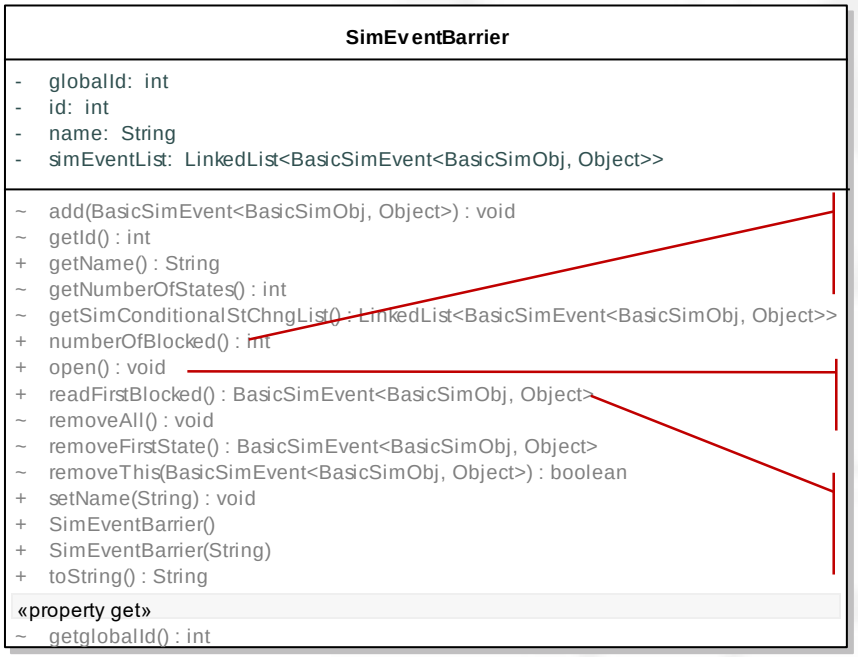
BasicSimEvent(TSimObj entity, TEventParams params, double period, int priority)

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

DisSimLab

public class SimEventBarrier

Klasa bariery, na której może być wstrzymane zdarzenie warunkowe (jedno lub wiele). Realizacja zdarzenia rozpocznie się w chwili otwarcia bariery metodą 'open' – zdarzenie zostanie zaplanowane w kalendarzu na bieżącą chwilę.

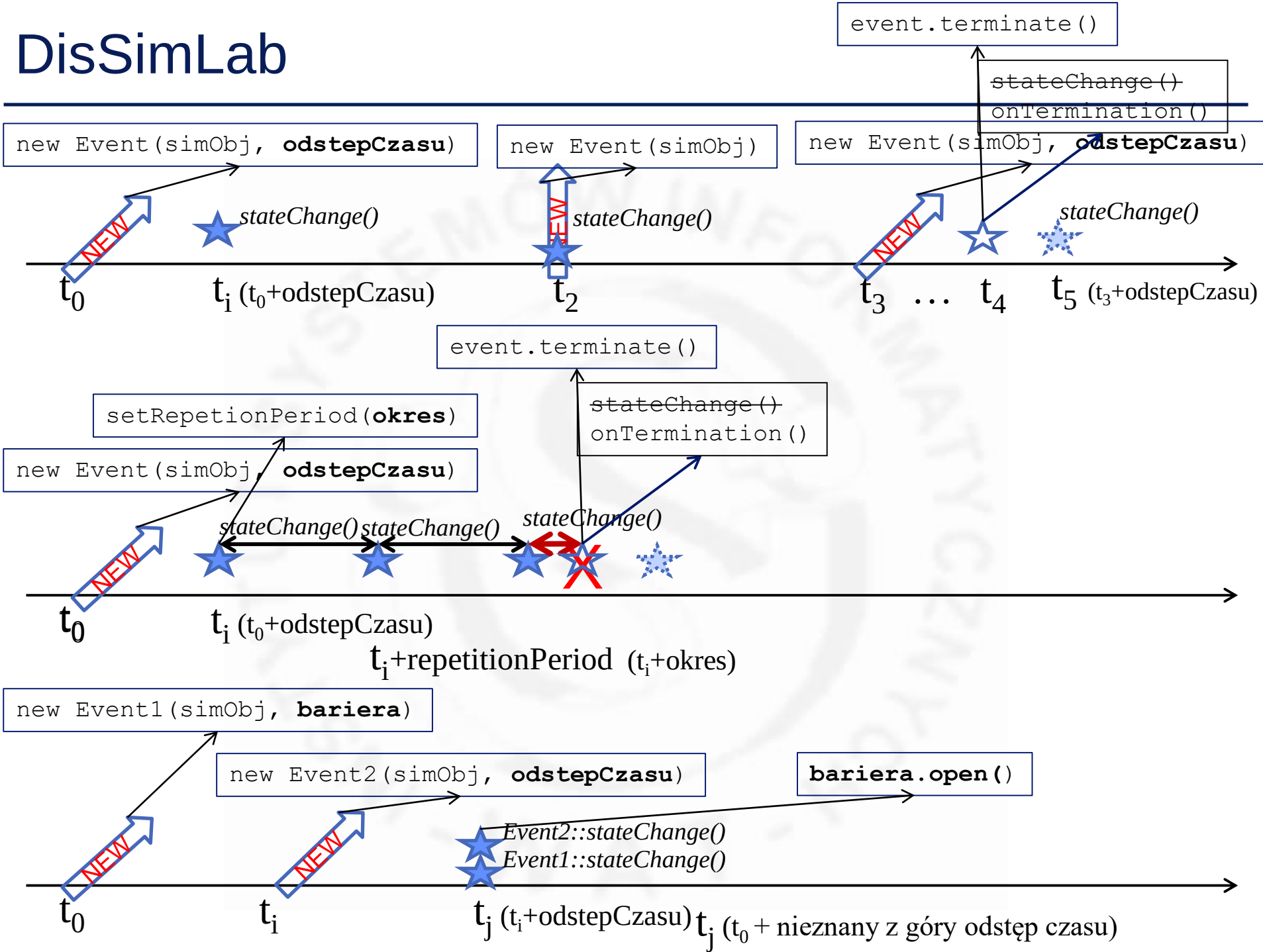


Metoda odczytuje liczbę obiektów zdarzeń wstrzymanych na tej barierze.

Metoda otwiera (podnosi) barierę.

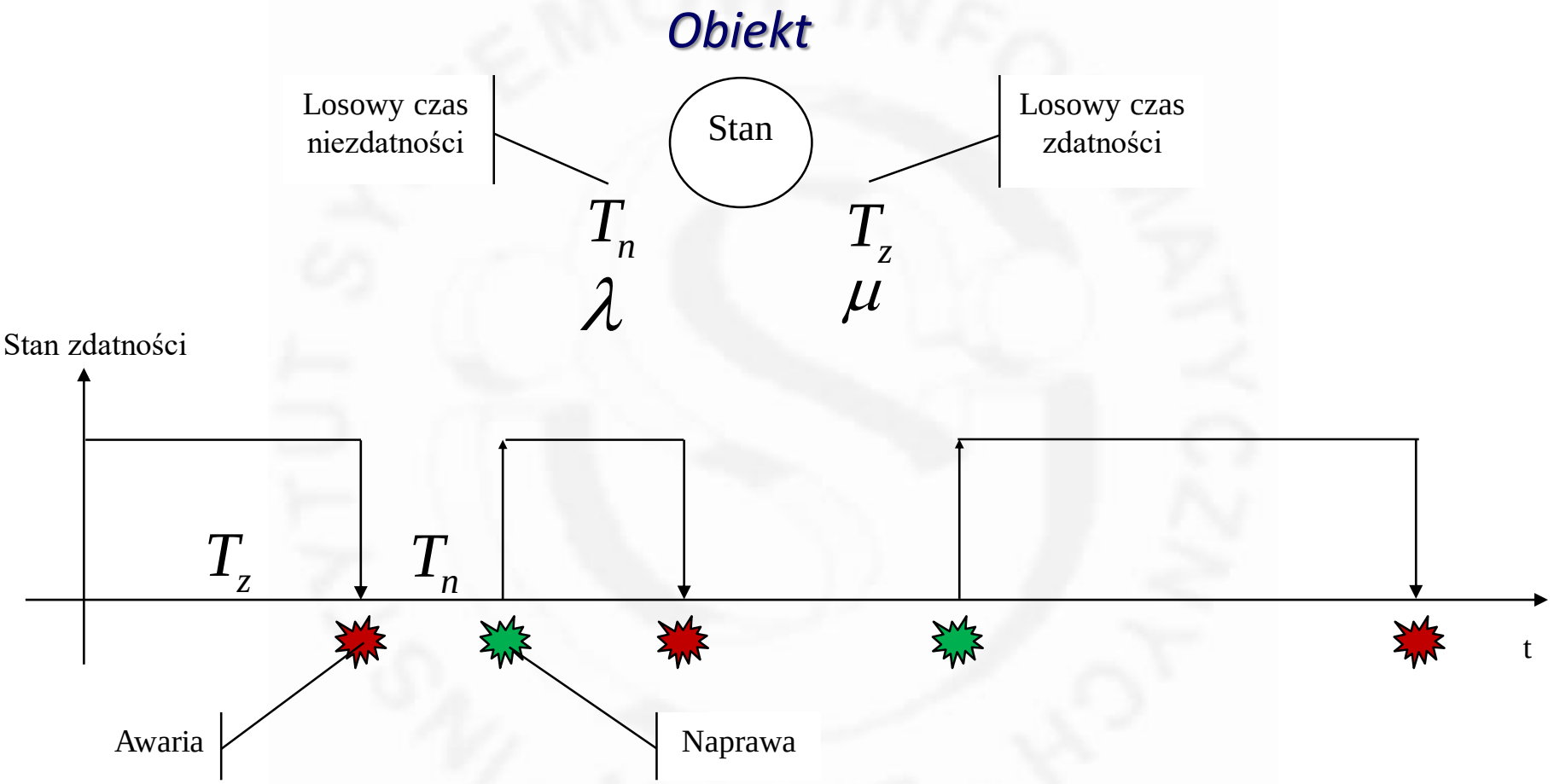
Metoda odczytuje referencję pierwszego zdarzenia wstrzymanego na barierze.

DisSimLab

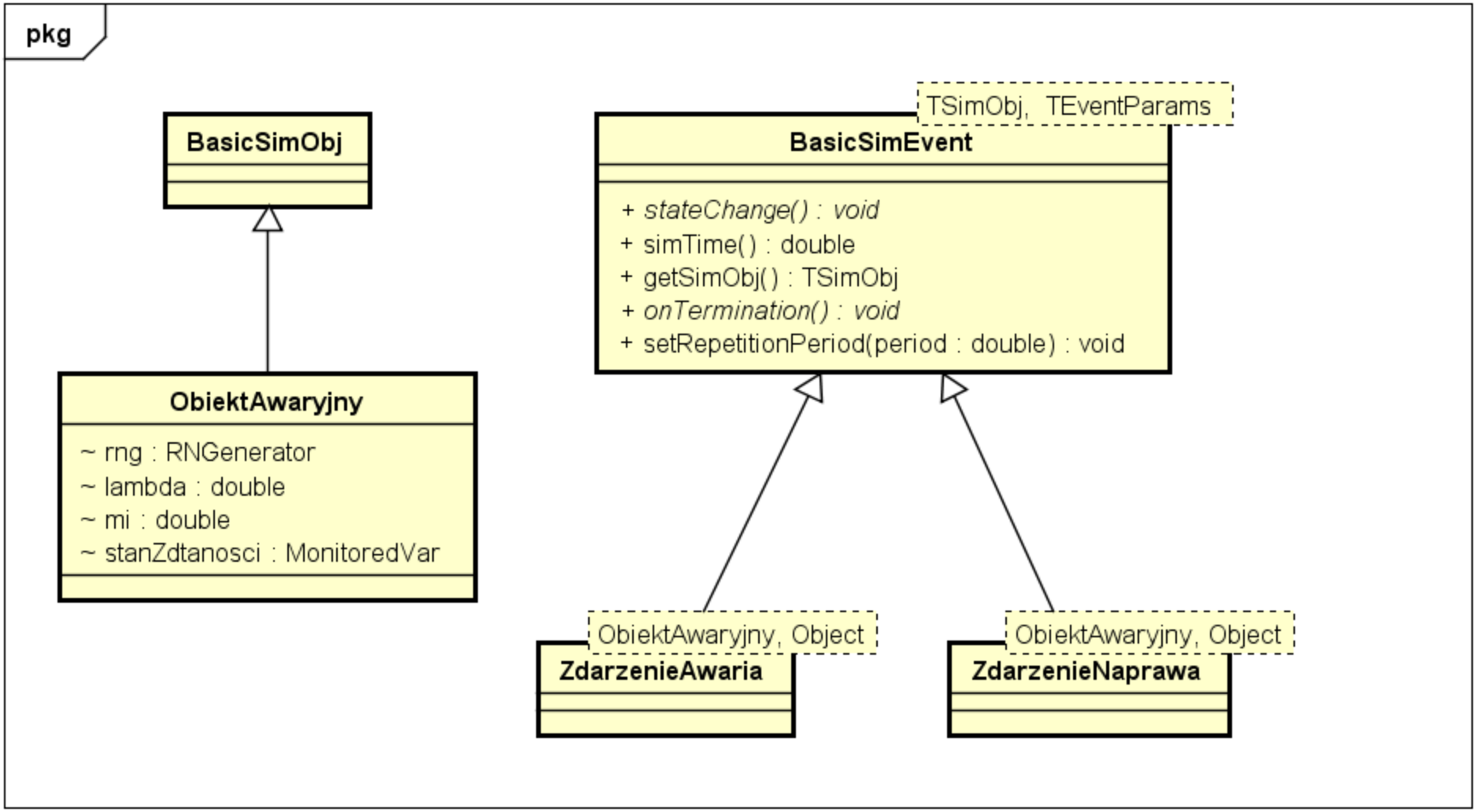

event.terminate()
stateChange()
onTermination()
new Event(simObj, odstepCzasu)
new Event(simObj)
new Event(simObj, odstepCzasu)
NEW
stateChange()
stateChange()
stateChange()
t0
ti (t0+odstepCzasu)
t2
t3 ... t4
t5 (t3+odstepCzasu)
event.terminate()
stateChange()
onTermination()
setRepetionPeriod(okres)
new Event(simObj, odstepCzasu)
stateChange()
stateChange()
stateChange()
t0
ti (t0+odstepCzasu)
ti+repetitionPeriod (ti+okres)
new Event1(simObj, bariera)
new Event2(simObj, odstepCzasu)
bariera.open()
Event2::stateChange()
Event1::stateChange()
t0
ti
tj (ti+odstepCzasu)
tj (t0 + nieznany z góry odstęp czasu)

Przykład

- Symulacja procesu awarii i napraw obiektu



Przykład – diagram klas



Przykład – klasa `ZdarzenieNaprawa`

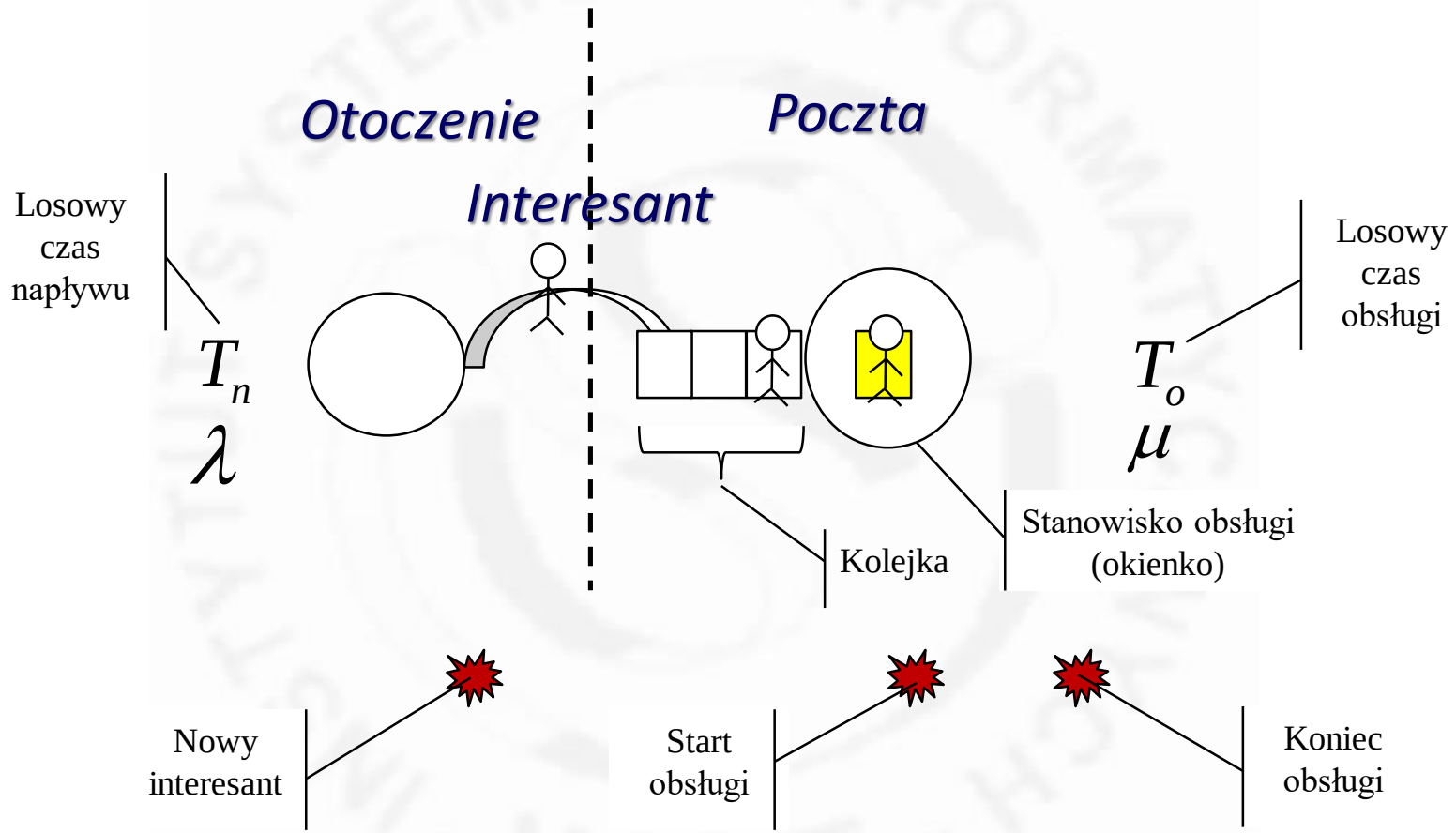
```
7
8   public class ZdarzenieNaprawa extends BasicSimEvent<ObiektAwaryjny, Object> {
9
10      public ZdarzenieNaprawa(ObiektAwaryjny entity, double delay) throws SimControlException {
11          super(entity, delay);
12      }
13
14      @Override
15      protected void stateChange() throws SimControlException {
16          ObiektAwaryjny ob = getSimObj();
17          ob.stanZdatnosci.setValue(1);
18          double dt = ob.rng.exponential(ob.lambda);
19          new ZdarzenieAwaria(ob, dt);
20      }
21
22      @Override
23      protected void onTermination() throws SimControlException {}
25
26      @Override
27      public Object getEventParams() { return null; }
30  }
31
```

Przykład – uruchomienie symulacji

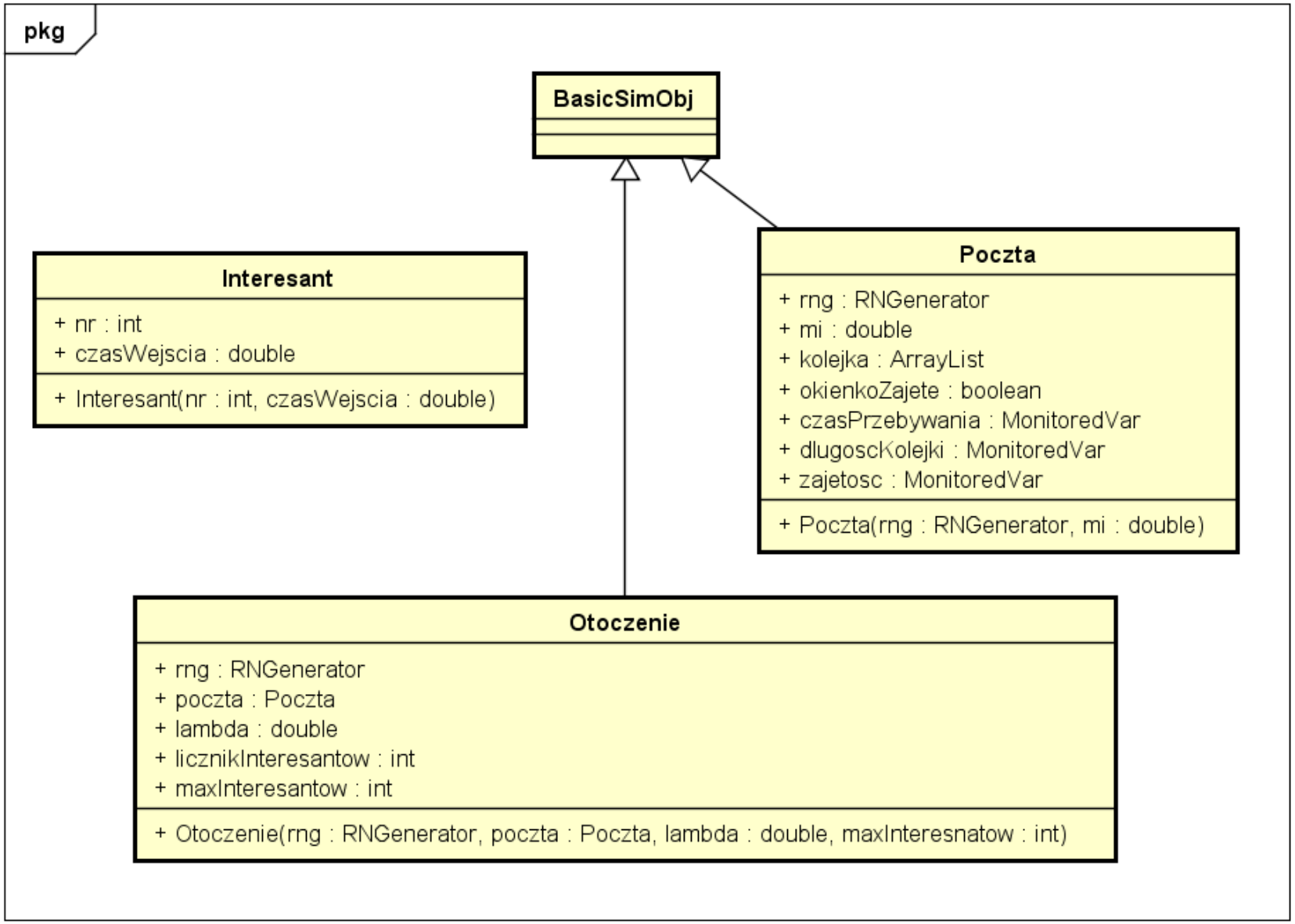
```
12
13  public class Main {
14
15      public static void main(String[] args) throws SimControlException {
16          SimManager sm = SimManager.getInstance();
17          sm.setEndSimTime(200);
18          ObiektAwaryjny ob = new ObiektAwaryjny( lambda: 0.1,  mi: 0.5);
19          new ZdarzenieNaprawa(ob,  delay: 0);
20          sm.startSimulation();
21          Diagram d1 = new Diagram(Diagram.DiagramType.HISTOGRAM,
22                  title: "Histogram stanu zdatnosci");
23          d1.add(ob.stanZdatnosci, Color.BLUE);
24          Diagram d2 = new Diagram(Diagram.DiagramType.TIME,
25                  title: "Zmiana w czasie stanu zdatnosci");
26          d2.add(ob.stanZdatnosci, Color.RED);
27          d1.show();
28          d2.show();
29      }
30  }
31
```

Zadanie

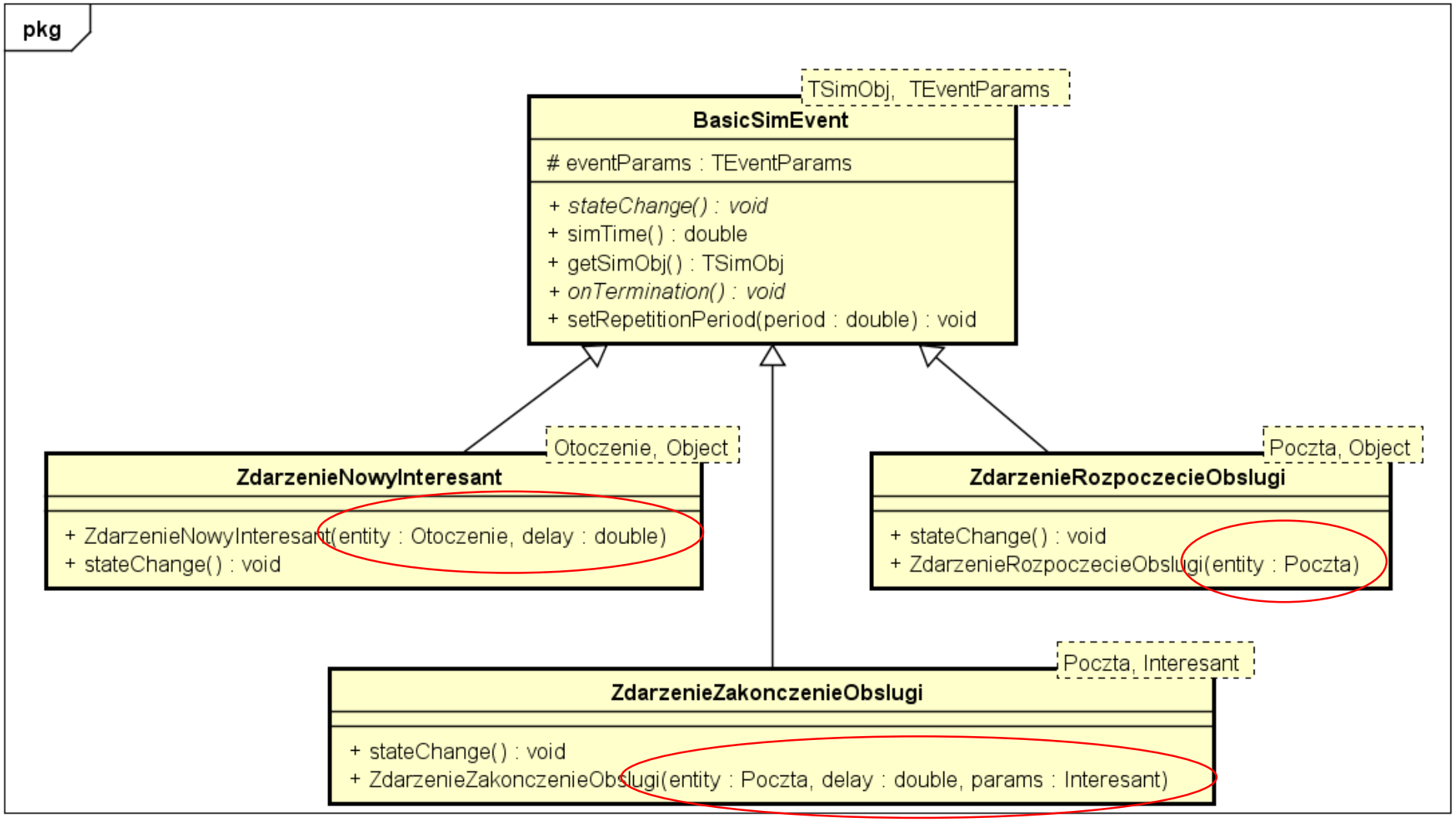
- Symulacja małego punktu pocztowego



Zadanie – diagram klas 1



Zadanie – diagram klas 2



Zadanie – identyfikacja skutków zdarzeń

- Zdarzenie pojawienia się interesanta pod warunkiem, że licznik interesantów mniejszy od max liczby interesantow do wygenerowania (`licznikInteresantow < maxInteresantow`)
  1. powołanie obiektu interesanta(id, czas wejścia):

     `new Interesant(...)`
  2. aktualizacja licznika interesantów
  3. dodanie interesanta do kolejki na poczcie
  4. aktualizacja długości kolejki na poczcie (zmienna monitorowana)
  5. jeżeli jest wolne okienko na poczcie, zaplanowanie zdarzenia rozpoczęcia obsługi natychmiast

     `new ZdarzenieRozpoczecieObslugi(...)`
  6. wylosowanie czasu, po którym pojawi się kolejny interesant (wykorzystać obiekt klasy `RNGenerator`) z parametrem ***lambda***
  7. zaplanowanie zdarzenia pojawienia się kolejnego interesanta

     `new ZdarzenieNowyInteresant(...)`

Zadanie – identyfikacja skutków zdarzeń

- Zdarzenie rozpoczęcia obsługi pod warunkiem, że kolejka nie jest pusta:
  1. pobranie interesanta z kolejki na poczcie pod zm. lokalną
  2. aktualizacja długości kolejki na poczcie (zmienna monitorowana)
  3. aktualizacja atrybutu `okienkoZajete`
  4. aktualizacja zajętości okienka na poczcie (zmienna monitorowana)
  5. wylosowanie czasu zakończenia obsługi (wykorzystać obiekt klasy `RNGenerator`) z parametrem ***mi***
  6. zaplanowanie zdarzenia zakończenia obsługi (przekazanie referencji obiektu interesanta jako parametru zdarzenia)

     ```
     new ZdarzenieZakonczeniaObslugi(...)
     ```

Zadanie – identyfikacja skutków zdarzeń

- Zdarzenie zakończenia obsługi:
  1. aktualizacja atrybutu `okienkoZajete`
  2. aktualizacja zajętości okienka na poczcie (zmienna monitorowana)
  3. aktualizacja czasu przebywania interesanta (zmienna monitorowana) – bieżący czas sym. (`simTime()`) i czas wejścia interesanta (obiekt interesanta widziany przez atrybut `eventParams`)
  4. zaplanowanie zdarzenia rozpoczęcia obsługi natychmiast

     `new ZdarzenieRozpoczecieObslugi(...)`

Zadanie

- Na podstawie powyższego diagramu klas napisać program symulujący działanie poczty;
- Wypisać na ekran odpowiednie komunikaty (logi) w trakcie symulacji w przykładowej formie:

  [czas symulacyjny] rodzaj zdarzenia, id interesanta, stan systemu

- Oszacować:
  - Średnią długość kolejki,
  - Średni czas przebywania interesanta,
  - Średnią zajętość okienka;
- Zobrazować:
  - Zmianę w czasie długości kolejki,
  - Dystrybuantę czasu przebywania;