Programowanie i projektowanie obiektowe

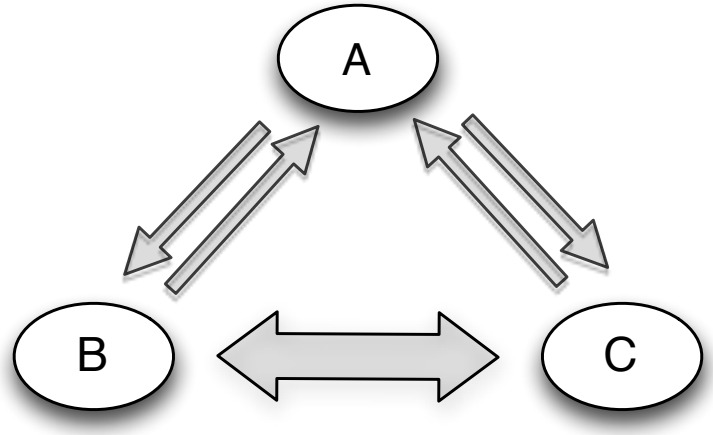
Przykład symulacji

Paweł Daniluk

Wydział Fizyki

Jesień 2012

Symulacja sieci kolejowej



Zasady

- Na torze może przebywać tylko jeden pociąg
- Stacja ma określoną liczbę peronów i może przyjąć co najwyżej jeden pociąg na peron.
- Pociąg, który nie może wjechać na stację, czeka na torze przed wjazdem.
- Pociąg, który nie może wjechać na zajęty tor, czeka na stacji blokując peron.

Klasy

Pociąg

```
public class Train {
    private int speed; // w km/h
    private String name;
    private Scheduler scheduler;
    Place location = null;
    Route route;


    Train(String name, int speed, Scheduler scheduler, Route route) {
        this.name = name;
        this.speed = speed;
        this.scheduler = scheduler;
        this.route = route;
    }
    ...
}
```

Klasy c.d.

Trasa

```
public class Route {
    private List<Station> stops;
    private int pos;

    Route(List<Station> stops) {
        this.stops = new ArrayList<Station>(stops);
        pos = 0;
    }

    public Station nextStop() {
        pos = (pos + 1) % stops.size();

        return currentStop();
    }

    public Station currentStop() {
        return stops.get(pos);
    }
}
```

Klasy c.d.

Miejsce

```
public abstract class Place {
    String name;

    Place(String name) {
        this.name = name;
    }
}
```

Klasy c.d.

Tor

```
public class Track extends Place {
    int length;
    Train train = null;
    Station from, to;

    Track(String name, Station from, Station to, int length) {
        super(name);
        this.from = from;
        this.to = to;
        this.length = length;
    }

    public boolean isEmpty() {
        return train == null;
    }

    Station otherEnd(Station s) {
        if (from == s) {
            return to;
        } else if (to == s) {
            return from;
        }
        throw new IllegalArgumentException("Ten tor (" + this.toString() + ") nie biegnie do stacji " + s.toStr
    }

    public int travelTime(int speed) {
        return (int) (Math.ceil((float) length / (float) speed * 60.0)) + ((new Random()).nextInt(20));
    }
    ...
}
```

Klasy c.d.

Stacja

```
public class Station extends Place {
    private Map<Station, Track> outTracks;
    private int nPlat;
    private Train[] platforms;

    Station(String name, int nPlat) {
        super(name);
        this.nPlat = nPlat;
        platforms = new Train[nPlat];
        outTracks = new HashMap<Station, Track>();
    }
    void addTrack(Track t) {
        Station s = t.otherEnd(this);
        outTracks.put(s, t);
    }
    private int getPlatform(Train t) {
        for (int i = 0; i < nPlat; i++) {
            if (platforms[i] == t) {
                return i;
            }
        }
        return -1;
    }
    private int freePlatform() {
        return getPlatform(null);
    }
    Track getTrack(Station nextStation) {
        return outTracks.get(nextStation);
    }
    ...
}
```

Klasy c.d.

Scheduler

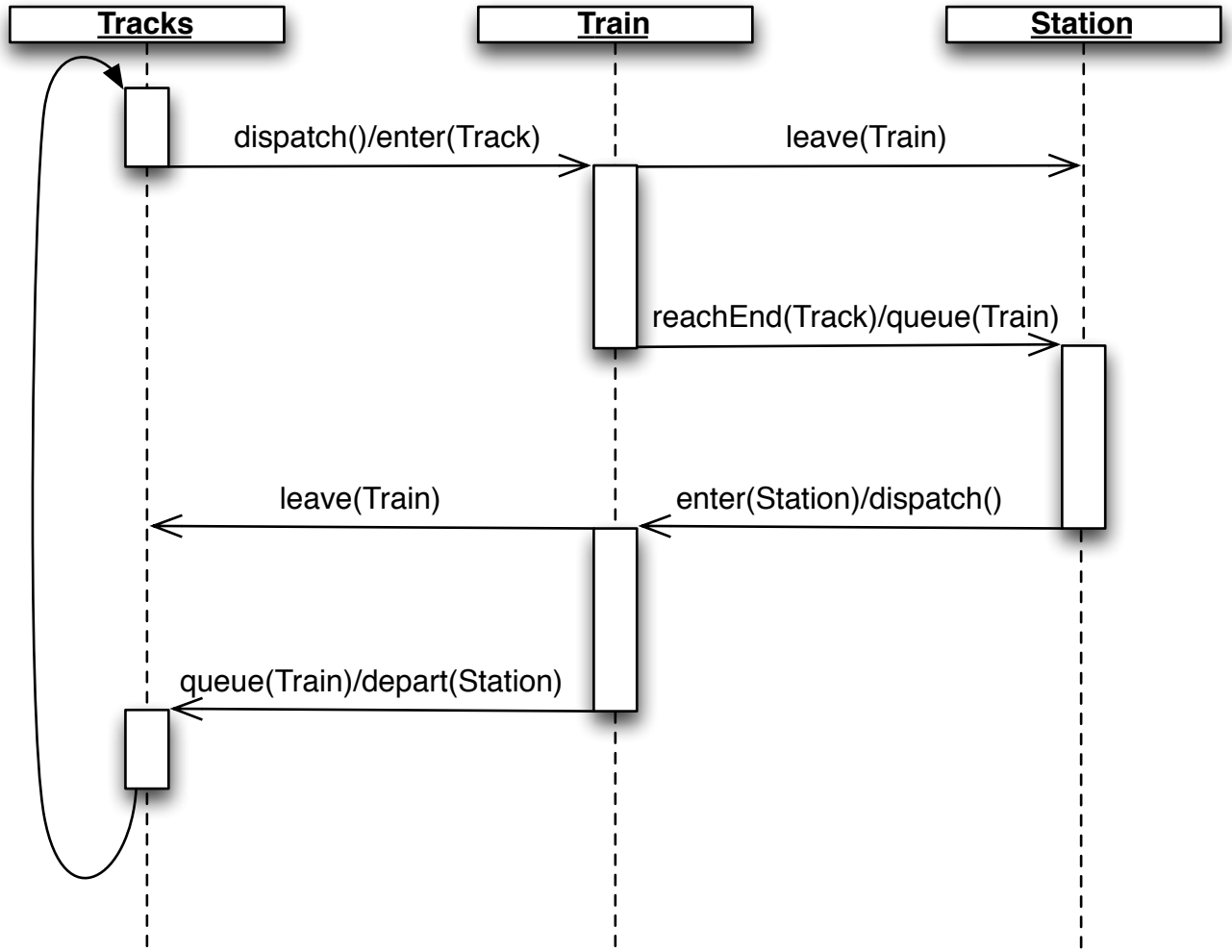
```
public class Scheduler {
    PriorityQueue<Entry> queue = new PriorityQueue<Entry>();
    private int time = 0;

    int getTime() {
        return time;
    }

    public void add(int time, Runnable task) {
        Entry e = new Entry(time, task);
        queue.add(e);
    }

    public void run() {
        while (!queue.isEmpty()) {
            Entry e = queue.poll();

            time = e.getTime();

            e.getTask().run();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {}
        }
    }
}
```

Klasy c.d.

Entry

```
public class Entry implements Comparable {
    private int time;
    private Runnable task;

    Entry(int time, Runnable task) {
        this.time = time;
        this.task = task;
    }

    @Override
    public int compareTo(Object t) {
        if (t instanceof Entry) {
            return getTime() - ((Entry) t).getTime();
        }
        throw new UnsupportedOperationException("Not supported.");
    }

    public int getTime() {
        return time;
    }

    public Runnable getTask() {
        return task;
    }
}
```

Sekwencja zdarzeń



Zachowanie miejsca

```
public abstract class Place {
    Queue<Train> waiting = new LinkedList<Train>();
    String name;

    Place(String name) {
        this.name = name;
    }

    public void queue(Train t) {
        waiting.add(t);
        this.dispatch();
    }

    @Override
    public String toString() {
        return name;
    }

    abstract void dispatch();
    abstract void leave(Train t);
}
```

Zachowanie toru

```
@Override
public void dispatch() {
    if (!waiting.isEmpty() && isEmpty()) {
        train = waiting.poll();
        train.enter(this);
    }
}

@Override
public void leave(Train t) {
    if (train != t) {
        throw new IllegalArgumentException("Tego pociagu (" + t.toString() + ") nie ma na torze "
                + this.toString() + ".");
    }

    train = null;
    this.dispatch();
}
```

Zachowanie stacji

```
@Override
 void dispatch() {
     int p;

     if (!waiting.isEmpty() && (p = freePlatform()) >= 0) {
         platforms[p] = waiting.poll();
         platforms[p].enter(this);
     }
 }

 @Override
 void leave(Train t) {
     int p = getPlatform(t);
     if (p == -1) {
         throw new IllegalArgumentException("Tego pociagu (" + t.toString() + ") nie ma ma na stacji "
                 + this.toString() + ".");
     }

     platforms[p] = null;
     this.dispatch();
 }
```

Zachowanie pociągu

```
void enter(final Track track) {
    int time = track.travelTime(speed) + scheduler.getTime();

    if(location!=null) location.leave(this);
    location = track;

    System.out.println(String.valueOf(scheduler.getTime()) + ": Pociąg " + this.toString()
           + " wjechał na tor " + track.toString() + ".");

    scheduler.add(time, new Runnable() {
        @Override
        public void run() {
            Train.this.reachEnd(track);
        }
    });
}

private void reachEnd(Track track) {
    System.out.println(String.valueOf(scheduler.getTime())+": Pociąg " + this.toString()
          + " dojechał do końca toru " + track.toString()+".");
    route.currentStop().queue(this);
}
```

Zachowanie pociągu c.d.

```
void enter(final Station station) {
    int time = scheduler.getTime() + 20 + ((new Random()).nextInt(5))-2;

    if(location!=null) location.leave(this);
    location = station;

    System.out.println(String.valueOf(scheduler.getTime()) + ": Pociąg " + this.toString()
          + " wjechał na stację " + station.toString() + ".");

    scheduler.add(time, new Runnable() {
        @Override
        public void run() {
            Train.this.depart(station);
        }
    });
}

private void depart(Station station) {
    Station nextStation = route.nextStop();

    Track nextTrack = station.getTrack(nextStation);

    System.out.println(String.valueOf(scheduler.getTime()) + ": Pociąg " + this.toString()
          + " odjeżdża ze stacji " + station.toString() + " po torze "+nextTrack.toString()+".");

    nextTrack.queue(this);
}
```

Inicjalizacja

```
public class Main {
    public static void main(String args[]) {
        Station A = new Station("A", 1);
        Station B = new Station("B", 2);
        Station C = new Station("C", 2);

        Track AB = new Track("AB", A, B, 100);
        Track BA = new Track("BA", B, A, 100);
        Track AC = new Track("AC", A, C, 200);
        Track CA = new Track("CA", C, A, 200);
        Track BCB = new Track("BCB", B, C, 300);

        A.addTrack(AB); A.addTrack(AC);
        B.addTrack(BA); B.addTrack(BCB);
        C.addTrack(CA); C.addTrack(BCB);

        Scheduler scheduler = new Scheduler();

        List<Station> l1 = new ArrayList<Station>();
        l1.add(A); l1.add(B); l1.add(C);
        List<Station> l2 = new ArrayList<Station>();
        l2.add(B); l2.add(A); l2.add(C);

        Route r1 = new Route(l1);
        Route r2 = new Route(l2);
        Train tr1 = new Train("tr1", 100, scheduler, r1);
        Train tr2 = new Train("tr2", 80, scheduler, r2);

        A.putTrain(tr1, 0);
        B.putTrain(tr2, 0);

        scheduler.run();
    }
```

Zadanie 1 – Symulacja sieci kolejowej

Zadanie

Zaimplementować i przetestować opisany przykład.

Zadanie 2 – Symulacja rynku kapitałowego (za 3 punkty)

Zadanie

Na giełdzie w Kogutkowie Górnym dokonuje się obrotu akcjami lokalnej spółdzielni mleczarskiej. Notowania odbywają się w trybie ciągłym. W ramach pojedynczej transakcji inwestorzy mogą dokonać zakupu lub zbycia dokładnie jednej akcji. Zlecenia mogą być wystawiane z limitem ceny (górnym w przypadku kupna i dolnym w przypadku sprzedaży) lub po bieżącym kursie. Niewykonane zlecenia są kolejkowane: sprzedaży rosnąco, a kupna malejąco wg limitu ceny. Cena transakcji jest ustalana jako średnia z limitów parowanych zleceń. Zaimplementować system transakcyjny wraz z kilkoma rodzajami agentów dokonujących operacji giełdowych. Przetestować skuteczność różnych strategii inwestycyjnych.

Wskazówka

Brakujące szczegóły specyfikacji należy uzupełnić w zgodzie ze zdrowym rozsądkiem.