

13.06.2023r

# Autobusy

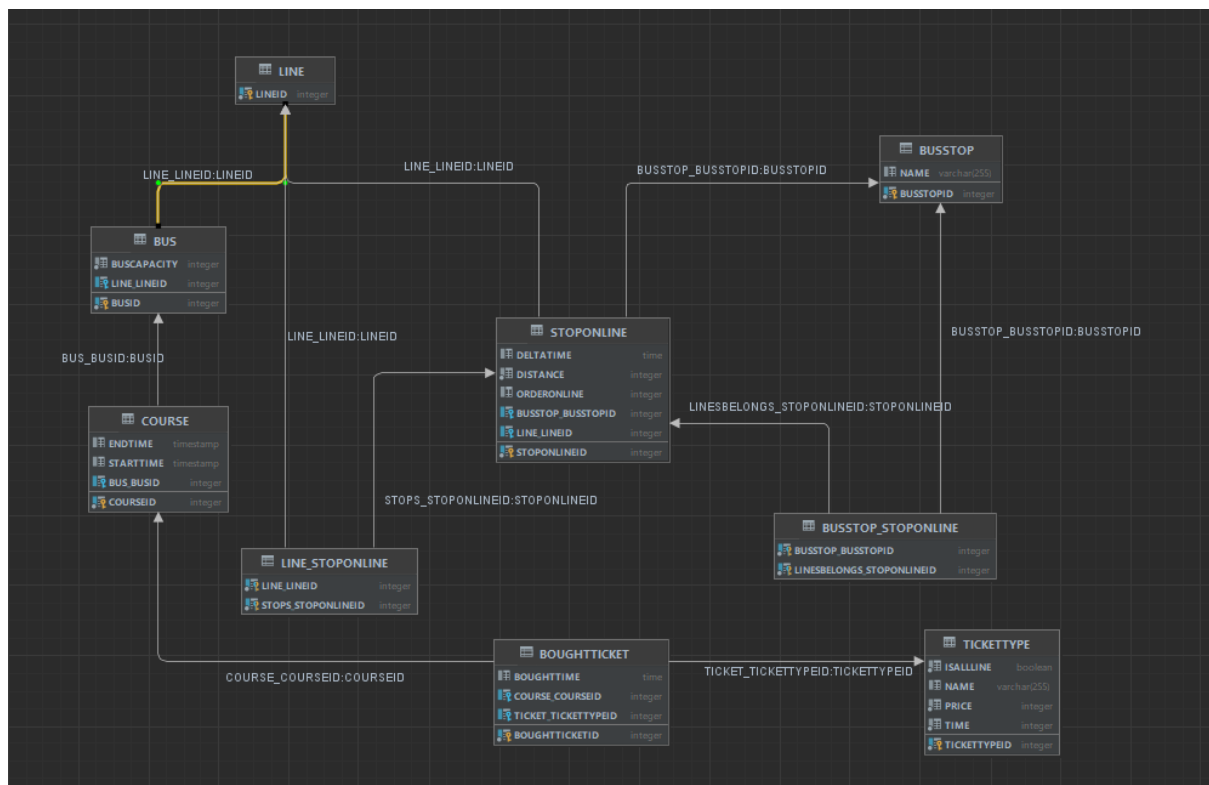
**Aplikacja do kupowania biletów i wyszukiwania połączeń**

Małgorzata Krupanek  
Magdalena Skrok  
Filip Jędrzejewski

## Użyte technologie

- Hibernate
- Java
- Apache

## Schemat Bazy



# Tabele

## 1) Bus

Zawiera informację na jakiej linii jeździ podany autobus oraz jaka jest jego pojemność. Przyjeliśmy, że jeden autobus obsługuje dokładnie jedną linię.

atrybuty i konstruktory:

```
1 usage
@Id
@GeneratedValue(strategy = GenerationType.AUTO)

private int busID;
3 usages
public int busCapacity;




3 usages
@ManyToOne
private Line line;

Małgorzata Krupanek
public Bus() {
}

Małgorzata Krupanek
public Bus(int busCapacity) { this.busCapacity=busCapacity; }

1 usage Małgorzata Krupanek
public Bus(int busCapacity, Line line){
    this.busCapacity = busCapacity;
    this.line=line;
}
```

tabela:

BUS	
 BUSCAPACITY	integer
 LINE_LINEID	integer
 BUSID	integer

## 2) Line

Zawiera dostępne linie autobusowe.

atrybuty:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)

private int lineID;

6 usages
@OneToMany
private Set<StopOnLine> stops;

1 usage new *
public Line() {
    this.stops = new HashSet<StopOnLine>();
}
```

metody:

- int getLength() - zwraca długość trasy lini

```
public int getLength() {
    int length = 0;
    for(StopOnLine stop : this.stops){
        length += stop.getDistance();
    }
    return length;
}
```

- int LocalTime getTimeLength() - zwraca czas przebycia trasy lini

```
public LocalTime getTimeLength() {
    LocalTime time = LocalTime.MIN;
    for(StopOnLine stop : this.stops){
        time = time.plusHours(stop.getDeltaTime().getHour());
        time = time.plusMinutes(stop.getDeltaTime().getMinute());
    }
    return time;
}
```

- int getNoStops() - zwraca liczbę przystanków

```
public int getNoStops() { return this.stops.size(); }
```

- String getStart()/getFinish() - zwraca przystanek początkowy/końcowy lini

```
3 usages  EfJot314
public String getStart() { return this.getBusStop(0).getBusStop().getName(); }

3 usages  EfJot314
public String getFinish() { return this.getBusStop(this.getNoStops()-1).getBusStop().getName(); }
```

- StopOnLine addStopToLine(BusStop stop, LocalTime deltaTime, int distance)
- dodaje do lini kolejny przystanek

```
public StopOnLine addStopToLine(BusStop stop, LocalTime deltaTime, int distance){
    StopOnLine newStop = new StopOnLine( line: this, stop, this.getNoStops(), deltaTime, distance);
    this.stops.add(newStop);
    stop.getLines().add(newStop);
    return newStop;
}
```

- gettery

```
public int getId() { return this.lineID; }
```

tabela:

LINE
LINEID integer

### 3) BusStop

Zawiera nazwy wszystkich przystanków wraz z przydzielonymi id.

atrybuty i konstruktory:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)

private int busStopID;
2 usages
private String name;

3 usages
@OneToMany
private Set<StopOnLine> linesBelongs;

Małgorzata Krupanek +1
public BusStop() { this.linesBelongs = new HashSet<StopOnLine>(); }

1 usage  EfJot314 +1
public BusStop(String name){
    this.name = name;
    this.linesBelongs = new HashSet<StopOnLine>();
}
```

tabela:

BUSSTOP	
NAME	varchar(255)
BUSSTOPID	integer

#### 4) StopOnLine

Określa jakie i w jakiej kolejności przystanki są przyporządkowane do poszczególnych linii.

atrybuty:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private int stopOnLineID;
2 usages
private Integer orderOnLine;
2 usages
private LocalTime deltaTime;
2 usages
private int distance;
2 usages
@ManyToOne
private BusStop busStop;
2 usages
@ManyToOne
private Line line;
EfJot314
public StopOnLine() {
}
```

konstruktory:

```
public StopOnLine(Line line, BusStop busStop, Integer order, LocalTime deltaTime, int distance) {
    this.orderOnLine = order;
    this.line = line;
    this.busStop = busStop;
    this.deltaTime = deltaTime;
    this.distance = distance;
}
```

tabela:

STOPONLINE	
DELTATIME	time
DISTANCE	integer
ORDERONLINE	integer
BUSSTOP_BUSSTOPID	integer
LINE_LINEID	integer
STOPONLINEID	integer

## 5) Course

Zawiera informacje na temat kursów autobusów - czas rozpoczęcia i zakończenia kursu oraz id autobusu, który jedzie.

atrybuty:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)

private int courseID;
2 usages
private LocalDateTime startTime;
3 usages
private LocalDateTime endTime;
2 usages
@ManyToOne
private Bus bus;
```

konstruktory:

```
public Course() {}
1 usage  👤 Małgorzata Krupanek +1 *
public Course(LocalDateTime startTime, Bus bus){
    this.startTime=startTime;
    this.endTime=null;
    this.bus=bus;
}
```

tabela:

COURSE	
ENDTIME	timestamp
STARTTIME	timestamp
BUS_BUSID	integer
COURSEID	integer

## 6) TicketType

Przechowywuje dostępne typy biletów (czasowy czy na kurs, ulgowy czy normalny) wraz z cenami oraz ewentualnym czasem ważności.

atrybuty:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)

private int ticketTypeID;
2 usages
private int time;

2 usages
private String name;
2 usages
private boolean isAllline;
3 usages
private int price;
```

konstruktory:

```
public TicketType() {}

6 usages  👤 Małgorzata Krupanek
public TicketType(int time, String name, boolean isAllline, int price){

    this.time=time;
    this.name=name;
    this.isAllline=isAllline;
    this.price=price;

}
```



tabela:

TICKETTYPE	
ISALLINE	boolean
NAME	varchar(255)
PRICE	integer
TIME	integer
TICKETTYPEID	integer

## 7) BoughtTicket

Przechowuje zakupione przez użytkownika bilety wraz z informacjami kiedy był kupiony, na jaki kurs i jakiego jest typu.

atrybuty:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)

private int boughtTicketID;
2 usages
private LocalDateTime boughtTime;

2 usages
@ManyToOne
private Course course;
2 usages
@ManyToOne
private TicketType ticket;
```

konstruktory:





```
public BoughtTicket() {}

1 usage  Małgorzata Krupanek +1
public BoughtTicket(LocalTime boughtTime, Course course, TicketType ticket){

    this.boughtTime=boughtTime;
    this.course = course;
    this.ticket=ticket;

}
```

tabela:

BOUGHTTICKET		
	BOUGHTTIME	time
	COURSE_COURSEID	integer
	TICKET_TICKETTYPEID	integer
	BOUGHTTICKETID	integer

# Interfejs użytkownika

Obsługiwany jest za pomocą metody `int parseInputAndExecute(List<String> input, Session session)` w klasie `main`

```
private static int parseInputAndExecute(List<String> input, Session session){
```

Operacje w aplikacji wykonywane są za pomocą komend w konsoli:

## 1) exit

Kończy program.

```
private static int parseInputAndExecute(List<String> input, Session session){  
    //wyjście  
    if(input.get(0).equals("exit")){  
        return -1;  
    }  
}
```

## 2) help

Wypisuje dostępne komendy.

```
//pomoc  
if(input.get(0).equals("help")){  
    System.out.println("Dostępne polecenia:");  
    System.out.println("+ exit");  
    System.out.println("+ help");  
    System.out.println("+ wypisz bilety");  
    System.out.println("+ wypisz linie");  
    System.out.println("+ wypisz kursy");  
    System.out.println("+ wypisz przystanki");  
    System.out.println("+ wypisz kurs *id_kursu*");  
    System.out.println("+ wypisz polaczenia *id_przystanku_startowego* *id_przystanku_koncowego*");  
    System.out.println("+ sprawdz *id_biletu* *id_kursu_gdzie_bilet_jest_sprawdzany*");  
    System.out.println("+ kup *id_typu_biletu* *id_kursu*");  
    return 0;  
}
```

```
>>> help  
Dostępne polecenia:  
+ exit  
+ help  
+ wypisz bilety  
+ wypisz linie  
+ wypisz kursy  
+ wypisz przystanki  
+ wypisz kurs *id_kursu*  
+ wypisz polaczenia *id_przystanku_startowego* *id_przystanku_koncowego*  
+ sprawdz *id_biletu* *id_kursu_gdzie_bilet_jest_sprawdzany*  
+ kup *id_typu_biletu* *id_kursu*
```

## 3) wypisz bilety

Wypisuje dostępne typy biletów wraz z cenami.

```
//wypisywanie
if(input.get(0).equals("wypisz")){
    //bilety
    if(input.get(1).equals("bilety")){
        List<TicketType> tts = session.createQuery("SELECT t FROM TicketType t", TicketType.class).getResultList();
        System.out.println("Dostępne typy biletów:");
        for(TicketType tt : tts){
            System.out.println(tt.getId()+" : "+tt.getName()+" => "+tt.getPrice());
        }
        return 0;
    }
}
```

```
>>> wypisz bilety
Hibernate:
select
    t1_0.ticketTypeID,
    t1_0.isAllLine,
    t1_0.name,
    t1_0.price,
    t1_0.time
from
    TicketType t1_0
Dostępne typy biletów:
1: ULGOWY 20 minutowy => 2
2: ULGOWY 60 minutowy => 3
3: ULGOWY 60 minutowy lub na całą linie => 4
4: NORMALNY 20 minutowy => 3
5: NORMALNY 60 minutowy => 4
6: NORMALNY 60 minutowy lub na całą linie => 5
```

## 4) wypisz linie

Wypisuje dostępne linie z przystankami początkowymi i końcowymi.

```
//linie
if(input.get(1).equals("linie")){
    List<Line> lines = session.createQuery("SELECT l FROM Line l", Line.class).getResultList();
    System.out.println("Linie autobusowe:");
    for(Line l : lines){
        System.out.println(l.getId()+" : "+l.getStart()+" -> "+l.getFinish()+" (" +l.getLength()+")");
    }
    return 0;
}
```

```
>>> wypisz linie
Hibernate:
select
    l1_0.lineID
from
    Line l1_0
```

Linie autobusowe:

- 1: Rynek -> UJ (53)
- 2: AGH -> Kopalnia (64)
- 3: Muzeum -> Zajezdnia (50)
- 4: Kościół -> Akademik (71)
- 5: Centrum Handlowe -> Pizzeria (79)
- 6: Błonia -> AGH (30)
- 7: Rondo -> Pizzeria (46)
- 8: Kopiec -> Akademik (61)
- 9: Błonia -> Lodowisko (63)
- 10: Osiedle -> Centrum Handlowe (63)
- 11: Dworzec PKS -> UJ (52)
- 12: Muzeum -> ZOO (71)
- 13: Fort -> UJ (39)
- 14: Fort -> Kościół (21)
- 15: Lodowisko -> Rynek (26)
- 16: Rondo -> Kościół (25)
- 17: Akademik -> AGH (41)
- 18: Muzeum -> Park (17)
- 19: Dworzec PKS -> Muzeum (42)
- 20: ZOO -> Kopalnia (28)

## 5) wypisz kursy

Wypisuje aktualne kursy wraz id\_kursu i numerem lini

```
//kursy
if(input.get(1).equals("kursy")){
    List<Course> courses = session.createQuery("SELECT c FROM Course c, Line l, Bus b WHERE c.endTime IS NOT NULL and c.bus=b.busID and b.line=l.lineID", Course.class).getResultList();
    System.out.println("Aktualne kursy:");
    for(Course c : courses){
        System.out.println(c.getId()+" -> "+c.getBus().getLine().getStart()+" -> "+c.getBus().getLine().getFinish()+" (" +c.getBus().getId()+")");
    }
    return 0;
}
```

```
>>> wypisz kursy
Hibernate:
select
    c1_0.courseID,
    c1_0.bus_busID,
    c1_0.endTime,
    c1_0.startTime
from
    Course c1_0,
    Line l1_0,
    Bus b1_0
where
    c1_0.endTime is not null
    and c1_0.bus_busID=b1_0.busID
    and b1_0.line_lineID=l1_0.lineID
```

```
Aktualne kursy:
23: Muzeum -> Zajezdnia (3)
27: Rondo -> Pizzeria (7)
47: Rondo -> Pizzeria (7)
31: Dworzec PKS -> UJ (11)
14: Fort -> Kościół (14)
34: Fort -> Kościół (14)
15: Lodowisko -> Rynek (15)
35: Lodowisko -> Rynek (15)
16: Rondo -> Kościół (16)
36: Rondo -> Kościół (16)
37: Akademik -> AGH (17)
18: Muzeum -> Park (18)
38: Muzeum -> Park (18)
20: ZOO -> Kopalnia (20)
40: ZOO -> Kopalnia (20)
```

## 6) wypisz kurs [id\_kursu]

Wypisuje szczegóły kursu o danym id.

```
//dany kurs
if(input.get(1).equals("kurs")){
    int courseID = Integer.parseInt(input.get(2));
    System.out.println(courseID);
    List<Course> courses = session.createQuery("SELECT c FROM Course c, Line l, Bus b WHERE c.endTime IS NOT NULL and c.bus=b.busID and b.line=l.lineID", Course.class).getResultList();

    for(Course c : courses){
        if(c.getId() == courseID){
            System.out.println("Przystanki kursu nr. "+courseID+" ( "+c.getBus().getLine().getStart()+" -> "+c.getBus().getLine().getFinish()+" ):");
            LocalDateTime sdt = c.getStartTime();
            int distance = 0;
            for(int i=0;i<c.getBus().getLine().getNoStops();i++){
                //zbierzam dane
                StopOnLine sol = c.getBus().getLine().getBusStop(i);
                sdt = sdt.plusHours(sol.getDeltaTime().getHour());
                sdt = sdt.plusMinutes(sol.getDeltaTime().getMinute());
                distance += sol.getDistance();
                String stopName = sol.getBusStop().getName();

                //print
                System.out.println((i+1)+". "+stopName+",\t data: "+sdt.toLocalDate()+"\t godzina: "+sdt.toLocalTime()+"\t odległość: "+distance+" km");
            }
            return 0;
        }
    }
    System.out.println("Nie znaleziono takiego kursu!");
    return 0;
}
```

```
>>> wypisz kurs 23
23
Hibernate:
select
    c1_0.courseID,
    c1_0.bus_busID,
    c1_0.endTime,
    c1_0.startTime
from
    Course c1_0,
    Line l1_0,
    Bus b1_0
where
    c1_0.endTime is not null
    and c1_0.bus_busID=b1_0.busID
    and b1_0.line_lineID=l1_0.lineID
```

```
Przystanki kursu nr. 23 ( Muzeum -> Zajeżdźnia ):
1. Muzeum, data: 2023-06-12, godzina: 20:22, odległość: 4 km
2. Błonia, data: 2023-06-12, godzina: 20:30, odległość: 9 km
3. Park, data: 2023-06-12, godzina: 20:40, odległość: 12 km
4. Zamek, data: 2023-06-12, godzina: 20:54, odległość: 15 km
5. Akademik, data: 2023-06-12, godzina: 21:04, odległość: 16 km
6. Las, data: 2023-06-12, godzina: 21:05, odległość: 17 km
7. Park, data: 2023-06-12, godzina: 21:16, odległość: 21 km
8. Pizzeria, data: 2023-06-12, godzina: 21:28, odległość: 25 km
9. Dworzec PKP, data: 2023-06-12, godzina: 21:41, odległość: 26 km
10. Park, data: 2023-06-12, godzina: 21:45, odległość: 31 km
11. Lodowisko, data: 2023-06-12, godzina: 21:49, odległość: 35 km
12. Lotnisko, data: 2023-06-12, godzina: 21:50, odległość: 37 km
13. Lotnisko, data: 2023-06-12, godzina: 21:51, odległość: 39 km
14. Rynek, data: 2023-06-12, godzina: 22:05, odległość: 44 km
15. Osiedle, data: 2023-06-12, godzina: 22:11, odległość: 46 km
16. Zajeżdźnia, data: 2023-06-12, godzina: 22:24, odległość: 50 km
```

## 7) kup [id\_typu\_biletu] [id\_kursu]

Kupuje podany typ biletu na podany kurs.

```
//zakup biletu
if(input.get(0).equals("kup") && input.size() > 2){
    int ticketID = Integer.parseInt(input.get(1));
    int courseID = Integer.parseInt(input.get(2));

    //sprawdzam czy jest taki kurs
    Course myCourse = null;
    List<Course> courses = session.createQuery("SELECT c FROM Course c WHERE c.endTime IS NOT NULL", Course.class).getResultList();
    for(Course course : courses){
        if(course.getId() == courseID){
            myCourse = course;
            break;
        }
    }
    if(myCourse == null){
        //jezeli nie znaleziono danego kursu
        System.out.println("Niepoprawne ID kursu!");
        return 1;
    }

    //sprawdzam czy jest taki typ biletu
    TicketType ticketType = null;
    List<TicketType> ticketTypes = session.createQuery("SELECT t FROM TicketType t", TicketType.class).getResultList();
    for(TicketType tt : ticketTypes){
        if(tt.getId() == ticketID){
            ticketType = tt;
            break;
        }
    }
    if(ticketType == null){
        //jezeli nie znaleziono danego typu biletu
        System.out.println("Niepoprawne ID typu biletu!");
        return 1;
    }
}
```

```
>>> kup 1 21
Hibernate:
    select
        c1_0.courseID,
        c1_0.bus_busID,
        c1_0.endTime,
        c1_0.startTime
    from
        Course c1_0
    where
        c1_0.endTime is not null
Hibernate:
    select
        t1_0.ticketTypeID,
        t1_0.isAllLine,
        t1_0.name,
        t1_0.price,
        t1_0.time
    from
        TicketType t1_0
```



```

Hibernate:

values
    next value for BoughtTicket_SEQ
Hibernate:
    insert
    into
        BoughtTicket
        (boughtTime, course_courseID, ticket_ticketTypeID, boughtTicketID)
    values
        (?, ?, ?, ?)
ID kupionego biletu: 1

```

## 8) sprawdz [id\_biletu] [id\_kursu\_gdzie\_bilet\_sprawdzany]

Sprawdza czy bilet o podanym id jest ważny na podany kurs.

```

//sprawdzenie poprawności biletu
if(input.get(0).equals("sprawdz") && input.size() > 2){
    int ticketID = Integer.parseInt(input.get(1));
    int actualCourseID = Integer.parseInt(input.get(2));

    //sprawdzam czy jest taki bilet
    boolean correct = false;
    List<BoughtTicket> currTickets = session.createQuery("SELECT t FROM BoughtTicket t", BoughtTicket.class).getResultList();
    for(BoughtTicket bt : currTickets){
        if(bt.getId() == ticketID){
            correct = true;

            //sprawdzilem ze istnieje/istniał taki bilet, teraz sprawdzam czy jest poprawny
            LocalTime goodTo = bt.getBoughtTime().plusMinutes(bt.getTicket().getTime());
            if(goodTo.isBefore(LocalTime.now())){
                correct = false;
            }
            if(bt.getTicket().isAllLine() && bt.getCourse().getId()==actualCourseID){
                correct = true;
            }

            break;
        }
    }

    //ostateczny komunikat
    if(correct){
        System.out.println("Bilet jest ważny!");
    }
    else{
        System.out.println("Bilet jest nieważny!");
    }

    return 0;
}

```

```

>>> sprawdz 1 23
Hibernate:
    select
        b1_0.boughtTicketID,
        b1_0.boughtTime,
        b1_0.course_courseID,
        b1_0.ticket_ticketTypeID
    from
        BoughtTicket b1_0
Bilet jest ważny!

```

## 9) wypisz połączenia [id\_przystanku\_startowego] [id\_przystanku\_koncowego]

Wypisuje połączenia pomiędzy przystankiem początkowym i końcowym - bezpośrednie i z jedną przesiadką

Pomocnicza klasa **Connection**:

Przechowuje połączenia pomiędzy dwoma przystankami.

atrybuty i konstruktor:

```
private List<Line> lines;  
8 usages  
private List<BusStop> busStops;  
3 usages  
private List<LocalTime> deltas;  
6 usages  
private LocalDateTime startTime;  
2 usages  EfJot314  
public Connection(BusStop startStop, LocalDateTime startTime){  
    this.startTime = startTime;  
    this.busStops = new ArrayList<>();  
    this.busStops.add(startStop);  
    this.lines = new ArrayList<>();  
    this.deltas = new ArrayList<>();  
}
```

metody:

- LocalTime getDuration() - zwraca czas danego połączenia wraz z oczekiwaniem na przesiadkę

```
public LocalTime getDuration(){  
    LocalTime result = LocalTime.MIN;  
  
    //bez przesiadek  
    if(this.lines.size() == 1){  
        BusStop start = this.busStops.get(0);  
        StopOnLine lineStop = this.lines.get(0).getBusStop(0);  
        int j=1;  
        while(lineStop.getBusStop().getId() != start.getId()){  
            lineStop=this.lines.get(0).getBusStop(j);  
            j+=1;  
        }  
        //linestop jest teraz tym na którym wsiadam  
        while(lineStop.getBusStop().getId() != this.busStops.get(1).getId()){  
            lineStop=this.lines.get(0).getBusStop(j);  
            result=result.plusMinutes(lineStop.getDeltaTime().getMinute());  
            result=result.plusHours(lineStop.getDeltaTime().getHour());  
            j+=1;  
        }  
        return result;  
    }  
}
```

```

//dodaje czasy przesiadek
for(LocalTime delta : this.deltas){
    result = result.plusHours(delta.getHour());
    result = result.plusMinutes(delta.getMinute());
}

//czasy przejazdow
for(int i=0;i<this.getNoLines();i++) {
    //zmienne pomocnicze
    int j = 0;
    BusStop start = this.busStops.get(i);
    StopOnLine lineStop = this.lines.get(i).getBusStop(j);
    //do startu
    while (lineStop.getBusStop().getId() != start.getId()) {
        j += 1;
        lineStop = this.lines.get(i).getBusStop(j);
    }
    //dodaje czasy
    while (lineStop.getBusStop().getId() != this.busStops.get(i + 1).getId()) {
        j += 1;
        result = result.plusHours(lineStop.getDeltaTime().getHour());
        result = result.plusMinutes(lineStop.getDeltaTime().getMinute());
        lineStop = this.lines.get(i).getBusStop(j);
    }
}

return result;
}

```

- LocalDateTime getEndTime() - zwraca godzinę dotarcia na przystanek docelowy

```

public LocalDateTime getEndTime(){
    LocalTime duration = this.getDuration();

    LocalDateTime result = this.startTime.plusHours(duration.getHour());
    result = result.plusMinutes(duration.getMinute());

    return result;
}

```

- String toString()

```
public String toString(){

    LocalDateTime endTime = this.getEndTime();

    String t1 = "";
    if(this.startTime.getMinute() < 10){
        t1 = this.startTime.getHour()+":0"+this.startTime.getMinute();
    }
    else{
        t1 = this.startTime.getHour()+":"+this.startTime.getMinute();
    }

    String t2 = "";
    if(endTime.getMinute() < 10){
        t2 = endTime.getHour()+":0"+endTime.getMinute();
    }
    else{
        t2 = endTime.getHour()+":"+endTime.getMinute();
    }

    if(this.getNoLines() == 1){
        return "Możesz jechać linia nr "+this.lines.get(0).getId()+" o godzinie "+t1+" Planowany przyjazd o godzinie: "+t2;
    }
    return "Możesz jechać linia nr "+this.lines.get(0).getId()+" o godzinie "+t1+", z przesiadką na przystanku "+
        this.busStops.get(1).getName()+" na linie nr "+this.lines.get(1).getId()+". Planowany przyjazd o godzinie: "+t2;
}
```

komparator dla obiektów klasy Connection:

```
class ConnectionsComparator implements Comparator<Connection> {

    Ef1ot314
    @Override
    public int compare(Connection c1, Connection c2) {

        //czas dojazdu do celu
        //c1 wcześniej niż c2
        if(c1.getEndTime().isBefore(c2.getEndTime())){
            return -1;
        }
        //c2 wcześniej niż c1
        if(c2.getEndTime().isBefore(c1.getEndTime())){
            return 1;
        }

        //ten sam czas dojazdu, ale różne czasy przejazdu
        //c1 krótszy niż c2
        if(c1.getDuration().isBefore(c2.getDuration())){
            return -1;
        }
        //c2 krótszy niż c1
        if(c2.getDuration().isBefore(c1.getDuration())){
            return 1;
        }

        //gdy te same czasy przejazdu, to mniejsza liczba przesiadek jest lepsza
        return c1.getNoLines()-c2.getNoLines();
    }
}
```

kod w metodzie **parseInputAndExecute** w **main**:

czytamy argument:

```
if(input.get(1).equals("połączenia") && input.size() > 3){  
    int startID = Integer.parseInt(input.get(2));  
    int endID = Integer.parseInt(input.get(3));
```

Sprawdzamy czy istnieją przystanki podane w argumentach:

```
boolean correct = false;  
List<BusStop> busStops = session.createQuery( "SELECT bs FROM BusStop bs ", BusStop.class).getResultList();  
  
for(BusStop bss : busStops){  
    if(bss.getId() == startID){  
        for(BusStop bsE : busStops){  
            if(bsE.getId() == endID){  
                correct=true;  
                break;  
            }  
        }  
    }  
}
```

Jeżeli oba przystanki istnieją to szukamy kursów z przystankami początkowymi i końcowymi:

```
if(correct){  
    //zapytania do bazy  
    List<Course> coursesWithStart =  
        session.createQuery( "SELECT c FROM Course c, Line l, Bus b, StopOnLine so, BusStop bs WHERE c.bus=b.busID and " +  
            "b.line=l.lineID and bs.busStopID=so.busStop and so.line=l.lineID and " +  
            "bs.busStopID = :start", Course.class).setParameter( "start", startID).getResultList();  
    List<Course> coursesWithEnd = session.createQuery( "SELECT c FROM Course c, Line l, Bus b, StopOnLine so, BusStop bs " +  
        "WHERE c.bus=b.busID and b.line=l.lineID and bs.busStopID=so.busStop and so.line=l.lineID " +  
        "and bs.busStopID = :end", Course.class).setParameter( "end", endID).getResultList();
```

Szukamy instancji dla podanych id przystanków:

```
BusStop startStop = new BusStop();  
Line ln = coursesWithStart.get(0).getBus().getLine();  
for(int i=0; i<ln.getNoStops(); i++){  
    if(ln.getBusStop(i).getBusStop().getId() == startID){  
        startStop = ln.getBusStop(i).getBusStop();  
    }  
}  
BusStop endStop = new BusStop();  
ln = coursesWithEnd.get(0).getBus().getLine();  
for(int i=0; i<ln.getNoStops(); i++){  
    if(ln.getBusStop(i).getBusStop().getId() == endID){  
        endStop = ln.getBusStop(i).getBusStop();  
    }  
}
```

Tworzymy pustą listę i dodajemy połączenia, które łączą startowy przystanek i docelowy:

```

//lista polaczen
Set<Connection> connections = new TreeSet<>(new ConnectionsComparator());

for(Course c1 : coursesWithStart){
    LocalDateTime sdtCourse1 = c1.getStartTime();
    LocalDateTime start=sdtCourse1;

    int n = c1.getBus().getLine().getNoStops();

    boolean startBeforeTransfer = false;

    //przechodzę po wszystkich przystankach tego kursu
    for(int i=0; i<n; i++){
        StopOnLine sol1 = c1.getBus().getLine().getBusStop(i);
        sdtCourse1 = sdtCourse1.plusHours(sol1.getDeltaTime().getHour());
        sdtCourse1 = sdtCourse1.plusMinutes(sol1.getDeltaTime().getMinute());
        if(sol1.getBusStop().getId() == startID){
            startBeforeTransfer = true;
            start = sdtCourse1;
            //nie interesuje nas to co bylo
            if(start.isBefore(LocalDateTime.now())){
                break;
            }
        }
    }
}

```

```

//dla kazdego przystanku sprawdzam wszystkie kursy do koncowego
for(Course c2 : coursesWithEnd){
    LocalDateTime sdtCourse2 = c2.getStartTime();
    int n1 = c2.getBus().getLine().getNoStops();
    for(int j=0; j<n1; j++) {
        StopOnLine sol2 = c2.getBus().getLine().getBusStop(j);
        sdtCourse2 = sdtCourse2.plusHours(sol2.getDeltaTime().getHour());
        sdtCourse2 = sdtCourse2.plusMinutes(sol2.getDeltaTime().getMinute());

        //jezeli doszedlem do przystanku docelowego to nie ma sensu dalej iterowac
        if(sol2.getBusStop().getId() == endID){
            break;
        }
    }
}

```

```

//kurs bezposredni
if(c1.getId() == c2.getId()){
    //sprawdzam czy przystanek startowy jest przed docelowym
    Line line = c1.getBus().getLine();
    for(int k=0;k<line.getNoStops();k++){
        if(line.getBusStop(k).getBusStop().getId() == startID){
            Connection connection = new Connection(startStop, start);
            connection.addLine(line, endStop, LocalTime.MIN);

            if(connection.getEndTime().isAfter(LocalDateTime.now()) &&
                !connections.contains(connection)) connections.add(connection);
            break;
        } else if (line.getBusStop(k).getBusStop().getId() == endID) {
            break;
        }
    }
}
}

```

```

//kurs z przesiadka
if(sol1.getBusStop().getId() == sol2.getBusStop().getId()
    && sdtCourse2.isAfter(sdtCourse1) && startBeforeTransfer){
    //czas przesiadki
    LocalDateTime delta = sdtCourse2.minusHours(sdtCourse1.getHour());
    delta = delta.minusMinutes(sdtCourse1.getMinute());

    Connection connection = new Connection(startStop, start);
    connection.addLine(c1.getBus().getLine(), sol2.getBusStop(), delta.toLocalTime());
    connection.addLine(c2.getBus().getLine(), endStop, LocalTime.MIN);
    if(connection.getEndTime().isAfter(LocalDateTime.now()) &&
        !connections.contains(connection)) connections.add(connection);
}
}
}
}
}

```

Jeżeli utworzona tablica jest pusta informujemy o braku połączeń, w przeciwnym przypadku je wypisujemy:

```

if(connections.size() == 0) {
    System.out.println("Nie ma połączenia z maksymalnie jedną przesiadką pomiędzy tymi przystankami!");
}
else{
    Iterator it = connections.iterator();
    for(int i=0; i<min(15, connections.size()); i++){
        System.out.println(it.next());
    }
}
}
}

```

W razie braku wprowadzonych przystanków w bazie też informujemy o tym użytkownika:

```

//jeżeli jest problem z przystankami
else{
    System.out.println("Nie ma takiego przystanku!");
}
return 0;

```

Otrzymany efekt w konsoli:

```
>>> wypisz połączoną 1 5
```

```
Hibernate:
```

```
select
    b1_0.busStopID,
    b1_0.name
```

```
from
    BusStop b1_0
```

```
Hibernate:
```

```
select
    c1_0.courseID,
    c1_0.bus_busID,
    c1_0.endTime,
    c1_0.startTime
```

```
from
    Course c1_0,
    Line l1_0,
    Bus b1_0,
    StopOnLine s1_0,
    BusStop b2_0
```

```
where
    c1_0.bus_busID=b1_0.busID
    and b1_0.line_lineID=l1_0.lineID
    and b2_0.busStopID=s1_0.busStop_busStopID
    and s1_0.line_lineID=l1_0.lineID
    and b2_0.busStopID=?
```

```
Hibernate:
```

```
select
    c1_0.courseID,
    c1_0.bus_busID,
    c1_0.endTime,
    c1_0.startTime
```

```
from
    Course c1_0,
    Line l1_0,
    Bus b1_0,
    StopOnLine s1_0,
    BusStop b2_0
```

```
where
    c1_0.bus_busID=b1_0.busID
    and b1_0.line_lineID=l1_0.lineID
    and b2_0.busStopID=s1_0.busStop_busStopID
    and s1_0.line_lineID=l1_0.lineID
    and b2_0.busStopID=?
```

```
Mozesz jechać linia nr 14 o godzinie 22:07. Planowany przyjazd o godzinie: 22:19
```

```
Mozesz jechać linia nr 14 o godzinie 23:51. Planowany przyjazd o godzinie: 0:03
```

```
Mozesz jechać linia nr 14 o godzinie 0:45. Planowany przyjazd o godzinie: 0:57
```



# DataCreator

Klasa posiadająca metody do generowania danych.

## 1) List<BusStop> createBusStops()

Metoda zwraca listę wprowadzonych przez nas przystanków.

```
public List<BusStop> createBusStops(){
    List<BusStop> result = new ArrayList<>();

    String[] stopNames = new String[]{"Kopiec", "Zamek", "Kościół", "AGH", "Miasteczko",
        "Kopalnia", "UJ", "ZOO", "Wisła", "Błonia", "Akademik",
        "Dworzec PKP", "Dworzec PKS", "Lotnisko", "Rynek", "Centrum Handlowe",
        "Łodowisko", "Park", "Stadion", "Fort", "Rondo", "Zajezdnia", "Osiedle",
        "Pizzeria", "Las", "Muzeum", "Obserwatorium astronomiczne"};

    for(int i=0; i<stopNames.length; i++){
        result.add(new BusStop(stopNames[i]));
    }

    return result;
}
```

## 2) List<Line> createLines(int n)

Zwraca listę n linii.

```
public List<Line> createLines(int n){
    List<Line> result = new ArrayList<>();

    for(int i=0; i<n; i++){
        result.add(new Line());
    }

    return result;
}
```

### 3) List<StopOnLine> addStopsToLine(List<BusStop> stops, List<Line> lines)

W sposób losowy przydziela kolejne przystanki do każdej lini z listy.

```
public List<StopOnLine> addStopsToLines(List<BusStop> stops, List<Line> lines){
    List<StopOnLine> result = new ArrayList<>();
    for(Line line : lines){
        int nOfStops = this.randInt( a: 5, b: 25);
        for(int j=0;j<nOfStops;j++){
            BusStop stop = stops.get(this.randInt( a: 0, b: stops.size()-1));
            LocalTime time = LocalTime.of( hour: 0, minute: 0);
            if(j > 0){
                time = LocalTime.of( hour: 0, this.randInt( a: 1, b: 15));
            }

            int dist = this.randInt( a: 1, b: 5);

            result.add(line.addStopToLine(stop, time, dist));
        }
    }

    return result;
}
```

### 4) List<TicketType> createTicketTypes()

Zwraca określone przez nas dostępne typy biletów.

```
public List<TicketType> createTicketTypes(){
    List<TicketType> result = new ArrayList<>();

    //ulgowy 20 min
    result.add(new TicketType( time: 20, name: "ULGOWY 20 minutowy", isAllLine: false, price: 2));

    //ulgowy 60 min
    result.add(new TicketType( time: 60, name: "ULGOWY 60 minutowy", isAllLine: false, price: 3));

    //ulgowy 60 min lub cała linia
    result.add(new TicketType( time: 60, name: "ULGOWY 60 minutowy lub na całą linię", isAllLine: true, price: 4));

    //normalny 20 min
    result.add(new TicketType( time: 20, name: "NORMALNY 20 minutowy", isAllLine: false, price: 3));

    //normalny 60 min
    result.add(new TicketType( time: 60, name: "NORMALNY 60 minutowy", isAllLine: false, price: 4));

    //normalny 60 min lub cała linia
    result.add(new TicketType( time: 60, name: "NORMALNY 60 minutowy lub na całą linię", isAllLine: true, price: 5));

    return result;
}
```

## 5) List<Bus> createBuses(int n, List<Line> lines)

Generuje listę n autobusów i przyporządkowuje każdemu linie.

```
public List<Bus> createBuses(int n, List<Line> lines){
    List<Bus> result = new ArrayList<>();

    int nOfLines = lines.size();

    for(int i=0; i<n; i++){
        result.add(new Bus(randInt( a: 10, b: 60), lines.get(i % nOfLines)));
    }

    return result;
}
```

## 6) List<Course> createCourses(int n, List<Bus> buses, LocalDateTime actualTime)

```
public List<Course> createCourses(int n, List<Bus> buses, LocalDateTime actualTime){

    LocalDate date = actualTime.toLocalDate();
    LocalTime now = actualTime.toLocalTime();

    List<Course> result = new ArrayList<>();

    int nOfBuses = buses.size();

    int actHour = now.getHour();

    for(int i=0; i<n; i++){
        int hour1 = randInt( a: (actHour-hourDelta+24)%24, b: (actHour-1+24)%24);
        int minute = randInt( a: 0, b: 59);

        LocalTime courseStartTime = LocalTime.of(hour1, minute);
        LocalDateTime cst = courseStartTime.atDate(date);

        Course newCourse = new Course(cst, buses.get(i%nOfBuses));

        //sprawdzanie kiedy kurs sie zakonczył/ma sie zakonczyc
        LocalTime courseTime = newCourse.getBus().getLine().getTimeLength();

        LocalTime courseEndTime = courseStartTime.plusHours(courseTime.getHour());
        courseEndTime = courseEndTime.plusMinutes(courseTime.getMinute());

        LocalDateTime cet = courseEndTime.atDate(date);

        //jesli kurs przeszedł przez polnoc to zmieniam dzien konca kursu na kolejny
        if(cst.isAfter(cet)){
            cet = cet.plusDays(1);
        }

        //sprawdzam czy kurs sie juz zakonczył
        if(cet.isBefore(actualTime)){
            newCourse.endCourse(cet);
        }

        //dodaje kurs do wynikowej listy
        result.add(newCourse);
    }

    return result;
}
```

## 7) BoughtTicket buyTicket(TicketType tt, Course course)

Tworzy i zwraca kupiony bilet.

```
public BoughtTicket buyTicket(TicketType tt, Course course){  
    return new BoughtTicket(LocalTime.now(), course, tt);  
}
```

Powyższe metody wykorzystane są do generowania danych w klasie main w metodzie **void fillDatabase(Session session, DataCreator creator)**

```
private static void fillDatabase(Session session, DataCreator creator){  
    //dodaje przystanki  
    List<BusStop> stops = creator.createBusStops();  
    for(BusStop stop : stops){  
        session.save(stop);  
    }  
  
    //dodaje linie  
    List<Line> lines = creator.createLines(n: 20);  
    for(Line line : lines){  
        session.save(line);  
    }  
  
    //dodaje przystanki do linii  
    List<StopOnLine> lineStops = creator.addStopsToLines(stops, lines);  
    for(StopOnLine stop : lineStops){  
        session.save(stop);  
    }  
  
    //dodaje typy biletow  
    List<TicketType> ticketTypes = creator.createTicketTypes();  
    for(TicketType tt : ticketTypes){  
        session.save(tt);  
    }  
  
    //dodaje autobusy  
    List<Bus> buses = creator.createBuses(n: 20, lines);  
    for(Bus bus : buses){  
        session.save(bus);  
    }  
  
    //dodaje kursy  
    List<Course> courses = creator.createCourses(n: 50, buses, LocalDateTime.now());  
    for(Course course : courses){  
        session.save(course);  
    }  
}
```