java-12-nodarbiba-steps.md

Statement.java

```
package lv.lu.lesson12;

import java.text.NumberFormat;
import java.util.Locale;
import java.util.Map;

public class Statement {

    private Invoice invoice;
    private Map<String, Play> plays;

    public Statement(Invoice invoice, Map<String, Play> plays) {
        this.invoice = invoice;
        this.plays = plays;
    }

    public String prepare() {
        long totalAmount = 0;
        long volumeCredits = 0;
        final NumberFormat format = NumberFormat.getCurrencyInstance(new Locale("lv", "LV"));
        String result = "Statement for " + invoice.getCustomer() + "\n";

        for (Performance perf : invoice.getPerformances()) {
            final Play play = plays.get(perf.getPlayId());
            int thisAmount = 0;

            switch (play.getType()) {
                case "tragedy":
                    thisAmount = 4000;
```

```
                    if (perf.getAudience() > 30) {
                        thisAmount += 1000 * (perf.getAudience() - 30);
                    }
                    break;
                case "comedy":
                    thisAmount = 3000;
                    if (perf.getAudience() > 20) {
                        thisAmount += 10000 + 500 * (perf.getAudience() - 20);
                    }
                    thisAmount += 300 * perf.getAudience();
                    break;
                default:
                    throw new RuntimeException("unknown type: " + play.getType());
            }

            //add volume credits
            volumeCredits += Math.max(perf.getAudience() - 30, 0);
            //add extra credit for every ten comedy attendees
            if ("comedy" == play.getType()) volumeCredits += Math.floor(perf.getAudience() / 5);

            //print line for this order
            result += "    " + play.getName()
                    + ": " + format.format(thisAmount / 100)
                    + " (" + perf.getAudience() + " seats)\n";
            totalAmount += thisAmount;
        }

        result += "Amount owed is " + format.format(totalAmount / 100) + "\n";
        result += "You earned " + volumeCredits + " credits\n";
        return result;
    }

}
```

reeplace fraagment of code

```
            switch (play.getType()) {
                case "tragedy":
                    thisAmount = 4000;
                    if (perf.getAudience() > 30) {
                        thisAmount += 1000 * (perf.getAudience() - 30);
                    }
                    break;
                case "comedy":
                    thisAmount = 3000;
                    if (perf.getAudience() > 20) {
                        thisAmount += 10000 + 500 * (perf.getAudience() - 20);
                    }
                    thisAmount += 300 * perf.getAudience();
                    break;
                default:
                    throw new RuntimeException("unknown type: " + play.getType());
            }
```

inti method

```
    private int amountFor(Performance perf, Play play) {..}
// and call it like this
    thisAmount = amountFor(perf, play);
```

next within method change identifier

> thisAmount

into

> result

next within method change parameter

> perf

into

> aPerformance

insert new method

```
    private Play playFor(Performance perf) {
        return plays.get(perf.getPlayId());
    }
```

replace

> final Play play = plays.get(perf.getPlayId());

with

> final Play play = playFor(perf);

aizvieto

> play

ar

> playFor(perf)

rezultāts

```
    for (Performance perf : invoice.getPerformances()) {
        int thisAmount = 0;

        thisAmount = amountFor(perf, playFor(perf));
```

```
            //add volume credits
            volumeCredits += Math.max(perf.getAudience() - 30, 0);
            //add extra credit for every ten comedy attendees
            if ("comedy" == playFor(perf).getType()) volumeCredits += Math.floor(perf.getAudience() / 5);

            //print line for this order
            result += "    " + playFor(perf).getName()
                    + ": " + format.format(thisAmount / 100)
                    + " (" + perf.getAudience() + " seats)\n";
            totalAmount += thisAmount;
        }
```

nomainam

> thisAmount = amountFor(perf, playFor(perf));

pret

> thisAmount = amountFor(perf);

un

> private int amountFor(Performance aPerformance, Play play) {

pret

> private int amountFor(Performance aPerformance) {

izlabijam parametra nosaukumu!

samainam

```
    private Play playFor(Performance perf) {
        return plays.get(perf.getPlayId());
```

```
    }
```

pret

```
    private Play playFor(Performance aPerformance) {
        return plays.get(aPerformance.getPlayId());
    }
```

aizvieetojam

> thisAmount

pret

> amountFor(perf)

izdzēšam

```
            int thisAmount = 0;
            thisAmount = amountFor(perf);
```

rezultātam jābūt šādam

```
package lv.lu.lesson12;

import java.text.NumberFormat;
import java.util.Locale;
import java.util.Map;

public class Statement {

    private Invoice invoice;
```

```
private Map<String, Play> plays;

public Statement(Invoice invoice, Map<String, Play> plays) {
    this.invoice = invoice;
    this.plays = plays;
}

public String prepare() {
    long totalAmount = 0;
    long volumeCredits = 0;
    final NumberFormat format = NumberFormat.getCurrencyInstance(new Locale("lv", "LV"));
    String result = "Statement for " + invoice.getCustomer() + "\n";

    for (Performance perf : invoice.getPerformances()) {
        //add volume credits
        volumeCredits += Math.max(perf.getAudience() - 30, 0);
        //add extra credit for every ten comedy attendees
        if ("comedy" == playFor(perf).getType()) volumeCredits += Math.floor(perf.getAudience() / 5);

        //print line for this order
        result += "    " + playFor(perf).getName()
                + ": " + format.format(amountFor(perf) / 100)
                + " (" + perf.getAudience() + " seats)\n";
        totalAmount += amountFor(perf);
    }

    result += "Amount owed is " + format.format(totalAmount / 100) + "\n";
    result += "You earned " + volumeCredits + " credits\n";
    return result;
}

private Play playFor(Performance aPerformance) {
    return plays.get(aPerformance.getPlayId());
}

private int amountFor(Performance aPerformance) {
    int result = 0;
```

```
        switch (playFor(aPerformance).getType()) {
            case "tragedy":
                result = 4000;
                if (aPerformance.getAudience() > 30) {
                    result += 1000 * (aPerformance.getAudience() - 30);
                }
                break;
            case "comedy":
                result = 3000;
                if (aPerformance.getAudience() > 20) {
                    result += 10000 + 500 * (aPerformance.getAudience() - 20);
                }
                result += 300 * aPerformance.getAudience();
                break;
            default:
                throw new RuntimeException("unknown type: " + playFor(aPerformance).getType());
        }
        return result;
    }

}
```

izveidojam jaunu metodi

```
private long volumeCreditsFor(Performance perf) {
    long volumeCredits = 0;
    volumeCredits += Math.max(perf.getAudience() - 30, 0);
    if ("comedy" == playFor(perf).getType()) volumeCredits += Math.floor(perf.getAudience() / 5);
    return volumeCredits;
}
```

aizvietojam

```
        volumeCredits += Math.max(perf.getAudience() - 30, 0);
        //add extra credit for every ten comedy attendees
        if ("comedy" == playFor(perf).getType()) volumeCredits += Math.floor(perf.getAudience() / 5);
```

pret

> volumeCredits += volumeCreditsFor(perf);

izmainam šo metodi

```
  private long volumeCreditsFor(Performance perf) {
      long volumeCredits = 0;
      volumeCredits += Math.max(perf.getAudience() - 30, 0);
      if ("comedy" == playFor(perf).getType()) volumeCredits += Math.floor(perf.getAudience() / 5);
      return volumeCredits;
  }
```

pret šadu (parametyra nosaukums un rtesult)

```
  private long volumeCreditsFor(Performance aPerformance) {
      long result = 0;
      result += Math.max(aPerformance.getAudience() - 30, 0);
      if ("comedy" == playFor(aPerformance).getType()) result += Math.floor(aPerformance.getAudience() / 5);
      return result;
  }
```

izveidojam metodi

```
  private String format(long aNumber) {
      return NumberFormat.getCurrencyInstance(new Locale("lv", "LV")).format(aNumber);
  }
```

un aizvieetojam

> - ": " + format.format(amountFor(perf) / 100)

un

> result += "Amount owed is " + format.format(totalAmount / 100) + "\n";

pret

> - ": " + format(amountFor(perf) / 100)

un

> result += "Amount owed is " + format(totalAmount / 100) + "\n";

tālāk

izveidojam metodi

```
  private String eur(long aNumber) {
      return NumberFormat.getCurrencyInstance(new Locale("lv", "LV")).format(aNumber);
  }
```

un aizvietojam to sekojoshi.

nomainam

> - ": " + format(amountFor(perf) / 100)

un

> result += "Amount owed is " + format(totalAmount / 100) + "\n";

pret

> - ": " + eur(amountFor(perf) / 100)

un

> result += "Amount owed is " + eur(totalAmount / 100) + "\n";

tālāk no for cikla izmetam

```
          volumeCredits += volumeCreditsFor(perf);
```

un pievienojam papildus for ciklui

```
      for (Performance perf : invoice.getPerformances()) {
          volumeCredits += volumeCreditsFor(perf);
      }
```

pārvietojam

> ```
>       long volumeCredits = 0;
> ```

pirms otrā cikla.

izveeidojam jaunu metodi

```
  private long totalVolumeCredits() {
      long volumeCredits = 0;
      for (Performance perf : invoice.getPerformances()) {
          volumeCredits += volumeCreditsFor(perf);
      }
```

```
        return volumeCredits;
    }
```

un otro ciklu aizvietojam pret metodes izsaaukumu

> long volumeCredits = totalVolumeCredits();

aizvieetojam

> result += "You earned " + volumeCredits + " credits\n";

preet

result += "You earned " + totalVolumeCredits() + " credits\n";

un izdzēšam

> long volumeCredits = totalVolumeCredits();

```
package lv.lu.lesson12;

import java.text.NumberFormat;
import java.util.Locale;
import java.util.Map;

public class Statement {

    private Invoice invoice;
    private Map<String, Play> plays;

    public Statement(Invoice invoice, Map<String, Play> plays) {
        this.invoice = invoice;
        this.plays = plays;
    }
```

```
public String prepare() {
    long totalAmount = 0;
    String result = "Statement for " + invoice.getCustomer() + "\n";

    for (Performance perf : invoice.getPerformances()) {
        //print line for this order
        result += "    " + playFor(perf).getName()
                + ": " + eur(amountFor(perf) / 100)
                + " (" + perf.getAudience() + " seats)\n";
        totalAmount += amountFor(perf);
    }

    long volumeCredits = totalVolumeCredits();

    result += "Amount owed is " + eur(totalAmount / 100) + "\n";
    result += "You earned " + volumeCredits + " credits\n";
    return result;
}

private long totalVolumeCredits() {
    long volumeCredits = 0;
    for (Performance perf : invoice.getPerformances()) {
        volumeCredits += volumeCreditsFor(perf);
    }
    return volumeCredits;
}

private String eur(long aNumber) {
    return NumberFormat.getCurrencyInstance(new Locale("lv", "LV")).format(aNumber);
}

private long volumeCreditsFor(Performance aPerformance) {
    long result = 0;
    result += Math.max(aPerformance.getAudience() - 30, 0);
    if ("comedy" == playFor(aPerformance).getType()) result += Math.floor(aPerformance.getAudience() / 5);
    return result;
}
```

```
    private Play playFor(Performance aPerformance) {
        return plays.get(aPerformance.getPlayId());
    }

    private int amountFor(Performance aPerformance) {
        int result = 0;
        switch (playFor(aPerformance).getType()) {
            case "tragedy":
                result = 4000;
                if (aPerformance.getAudience() > 30) {
                    result += 1000 * (aPerformance.getAudience() - 30);
                }
                break;
            case "comedy":
                result = 3000;
                if (aPerformance.getAudience() > 20) {
                    result += 10000 + 500 * (aPerformance.getAudience() - 20);
                }
                result += 300 * aPerformance.getAudience();
                break;
            default:
                throw new RuntimeException("unknown type: " + playFor(aPerformance).getType());
        }
        return result;
    }

}
```

izmetam

> long volumeCredits = totalVolumeCredits();

un

aizvietojam

> result += "You earned " + volumeCredits + " credits\n";

pret

> result += "You earned " + totalVolumeCredits() + " credits\n";

izveidojam metodi

```
      private long appleJuice() {
          long totalAmount = 0;
          for (Performance perf : invoice.getPerformances()) {
              totalAmount += amountFor(perf);
          }
          return totalAmount;
      }
```

un aiz for cikla ievietojaam

> long totalAmount = appleJuice();

un izmetam

> long totalAmount = 0;

un

> totalAmount += amountFor(perf);

metodes nosaukumu

> appleJuice

aizvietojam pret

> totalAmount

izdzēšaaam

> long totalAmount = appleJuice();

atsauksmi uz lietojumiem arī pārsaucam.

aizvietojam abās meetodēs

> totalAmount()

un

> totalVolumeCredits()

starpmainīgo uz result

> prepare()

metodi

pārsaucam par

> renderPlainText()

izveidojam jaunu metodi

```
    public String prepare() {
      return renderPlainText();
  }
```

prepare meetodi pārveidojam

public String prepare() { StatementData statementData = new StatementData(); return renderPlainText(statementData); }

renderPlainText() metodees signatūru pārveeidojaaam

> private String renderPlainText(StatementData data) {

izveidojam jaunu java klasi

```
 package lv.lu.lesson12;

public class StatementData {
}

tad pielabojam to

package lv.lu.lesson12;

public class StatementData {

    private final String customer;

    public StatementData(String customer) {
        this.customer = customer;
    }

    public String getCustomer() {
        return customer;
    }
}
```

un papildinām

```
    public String prepare() {
        StatementData statementData = new StatementData(invoice.getCustomer());
        return renderPlainText(statementData);
    }
```

un iekš renderPlainText()

aizvietojam

```
    String result = "Statement for " + invoice.getCustomer() + "\n";
```

pret

```
      String result = "Statement for " + data.getCustomer() + "\n";
```

stateementData pievienojam konstruktoram otru parameetru

```
package lv.lu.lesson12;

import java.util.List;

public class StatementData {

    private final String customer;
    private final List<Performance> performances;

    public StatementData(String customer, List<Performance> performances) {
        this.customer = customer;
        this.performances = performances;
    }

    public String getCustomer() {
```

```
        return customer;
    }

    public List<Performance> getPerformances() {
        return performances;
    }
}
```

ieeviešama statemeentData otru konsstruktoraa parametru

```
    public String prepare() {
        final StatementData statementData = new StatementData(invoice.getCustomer(), invoice.getPerformances());
        return renderPlainText(statementData);
    }
```

un samainam

> invoice.getPerformances()

pret

> data.getPerformances()

nākamās 2 metodes ir ar parametru data

```
        result += "Amount owed is " + eur(totalAmount(data) / 100) + "\n";
        result += "You earned " + totalVolumeCredits(data) + " credits\n";
```

prepare metodi uzlabojam sekojoši

```
    public String prepare() {
        final StatementData statementData = new StatementData(
```

```
                invoice.getCustomer(),
                invoice.getPerformances().stream().map(this::enrichPerformance).collect(Collectors.toList()));
        return renderPlainText(statementData);
    }
```

un pievienojam metodi

```
    private Performance enrichPerformance(Performance performance) {
        return new Performance(performance.getPlayId(), performance.getAudience());
    }
```

izveidojam jaunu klasi EnrichedPerformance

```
package lv.lu.lesson12;

public class EnrichedPerformance extends Performance {

    public EnrichedPerformance(String playId, int audience) {
        super(playId, audience);
    }
}
```

izmainam enrichPerformance() meetodi

```
    private EnrichedPerformance enrichPerformance(Performance performance) {
        final EnrichedPerformance result = new EnrichedPerformance(
                performance.getPlayId(), performance.getAudience(), playFor(performance));
        return result;
    }
```

papildinam EnrichedPerformance

package lv.lu.lesson12;

public class EnrichedPerformance extends Performance {

```
    private final Play play;

    public EnrichedPerformance(String playId, int audience, Play play) {
        super(playId, audience);
        this.play = play;
    }

}
```

papildinām

```
    private EnrichedPerformance enrichPerformance(Performance performance) {
        final EnrichedPerformance result = new EnrichedPerformance(
                performance.getPlayId(),
                performance.getAudience(),
                playFor(performance));
        return result;
    }
```

Performance nomainam uz EnrichPerformance

// 22:30

nomainam

> amountFor(perf)

pret

> perf.getAmount()

enrichPerformance() meetodei pievienojam

> result.setAmount(amountFor(result));

tagad izejas tekstss ir šāds

```
package lv.lu.lesson12;

import java.text.NumberFormat;
import java.util.Locale;
import java.util.Map;
import java.util.stream.Collectors;

public class Statement {

    private Invoice invoice;
    private Map<String, Play> plays;

    public Statement(Invoice invoice, Map<String, Play> plays) {
        this.invoice = invoice;
        this.plays = plays;
    }

    public String prepare() {
        final StatementData statementData = new StatementData(
                invoice.getCustomer(),
                invoice.getPerformances().stream().map(this::enrichPerformance).collect(Collectors.toList()));
        return renderPlainText(statementData);
    }

    private EnrichedPerformance enrichPerformance(Performance performance) {
        final EnrichedPerformance result = new EnrichedPerformance(
                performance.getPlayId(),
                performance.getAudience(),
```

```
            playFor(performance));
    result.setAmount(amountFor(result));
    return result;
}

private String renderPlainText(StatementData data) {
    String result = "Statement for " + data.getCustomer() + "\n";

    for (EnrichedPerformance perf : data.getPerformances()) {
        //print line for this order
        result += "    " + perf.getPlay().getName()
                + ": " + eur(perf.getAmount() / 100)
                + " (" + perf.getAudience() + " seats)\n";
    }

    result += "Amount owed is " + eur(totalAmount(data) / 100) + "\n";
    result += "You earned " + totalVolumeCredits(data) + " credits\n";
    return result;
}

private long totalAmount(StatementData data) {
    long result = 0;
    for (EnrichedPerformance perf : data.getPerformances()) {
        result += perf.getAmount();
    }
    return result;
}

private long totalVolumeCredits(StatementData data) {
    long result = 0;
    for (EnrichedPerformance perf : data.getPerformances()) {
        result += volumeCreditsFor(perf);
    }
    return result;
}

private String eur(long aNumber) {
```

```
        return NumberFormat.getCurrencyInstance(new Locale("lv", "LV")).format(aNumber);
    }

    private long volumeCreditsFor(EnrichedPerformance aPerformance) {
        long result = 0;
        result += Math.max(aPerformance.getAudience() - 30, 0);
        if ("comedy" == aPerformance.getPlay().getType()) result += Math.floor(aPerformance.getAudience() / 5);
        return result;
    }

    private Play playFor(Performance aPerformance) {
        return plays.get(aPerformance.getPlayId());
    }

    private int amountFor(EnrichedPerformance aPerformance) {
        int result = 0;
        switch (aPerformance.getPlay().getType()) {
            case "tragedy":
                result = 4000;
                if (aPerformance.getAudience() > 30) {
                    result += 1000 * (aPerformance.getAudience() - 30);
                }
                break;
            case "comedy":
                result = 3000;
                if (aPerformance.getAudience() > 20) {
                    result += 10000 + 500 * (aPerformance.getAudience() - 20);
                }
                result += 300 * aPerformance.getAudience();
                break;
            default:
                throw new RuntimeException("unknown type: " + aPerformance.getPlay().getType());
        }
        return result;
    }

}
```

un EnrichedPerformancee

```
package lv.lu.lesson12;

public class EnrichedPerformance extends Performance {

    private final Play play;

    private long amount;

    public EnrichedPerformance(String playId, int audience, Play play) {
        super(playId, audience);
        this.play = play;
    }

    public Play getPlay() {
        return play;
    }

    public long getAmount() {
        return amount;
    }

    public void setAmount(long amount) {
        this.amount = amount;
    }

}
```

// 22:37