TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Kaarel Sööt 155215 IABB

TARKVARA DISAINIMUSTRITE ÕPETAMISEKS VAJALIKE VAHENDITE VÄLJATÖÖTAMINE JA VALIDEERIMINE

Bakalaureusetöö

Juhendaja: Gunnar Piho
Doktorikraad
Dotsent

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kaarel Sööt

19.05.2019

Annotatsioon

Tarkvara disainimustrite õpetamiseks vajalike vahendite väljatöötamine ja valideerimine

Tarkvara disainimustrid on üldised korratavad lahendused tihti esinevatele probleemidele tarkvara disainis. Bakalaureuseõppe tudengid, kes on selgeks saanud programmeerimise alusteadmised, ei ole kirjutanud piisavalt palju tarkvara, et oleks olnud võimalust samalaadsete probleemide korduvaks esinemiseks. Teadmised tarkvara disainimustritest võimaldavad mõelda süsteemide disainimisest abstraktsioonitaseme võrra kõrgemalt ja tänu laienenud sõnavarale suhelda efektiivsemalt meeskonnaliikmetega.

Käesoleva töö eesmärgiks on luua eesti keelsed vahendid tarkvara disainimustrite õpetamiseks tudengitele ning need seejärel valideerida. Vahendite hulka kuuluvad iseseisvaks õppetööks mõeldud tuntumate disainimustrite lähtekood, neile vastavad moodultestid ning seletavad eesti keelsed juhendid. Kursust läbi viivale õppejõule on koostatud kontrolltöö küsimused, millega tudengite disainimustrite alaseid teadmisi kontrollida.

Töö objekti hindamiseks kasutati loodud õppevahendeid Tallinna Tehnikaülikooli õppeaines „Infosüsteemide arendamine.” Tulemuste põhjal koostati analüüs ning arutleti edasiste võimalike tööde üle.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 22 leheküljel, 5 peatükki, 12 joonist ja 1 tabelit.

Abstract

Creating and Validating Resources for Teaching Software Design Patterns

Software design patterns are general, reusable solutions to commonly occuring problems in software design. Undergraduate students who have only recently learned programming, have not written enough software to have repeatedly encountered similar problems. Knowledge of software design patterns enable thinking about designing software systems on a higher abstraction level and communicating more efficiently with teammates by providing a common vocabulary.

The purpose of this thesis is to create and validate resources for teaching software design patterns in Estonian. Resources include self-study materials in the form of source code, unit tests and tutorials. For teachers preparing a course, there are test questions for measuring students' knowledge of design patterns.

To validate the usefulness of the created resources, students attending a course in Information Systems Development at TalTech used the materials to learn software design patterns. Their knowledge on design patterns was tested and the results analyzed. Possible future research related to teaching design patterns was discussed.

The thesis is written in Estonian and contains 22 pages of text, 5 chapters, 12 figures and 1 table.

Lühendite ja mõistete sõnastik

GoF	***Gang of Four*** tarkvara disainimustrid Tarkvara disainimustreid populariseerinud raamatu „Design Patterns: Elements of Reusable Object-Oriented Software" [1] autorid Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides on tuntud, kui Gang of Four. Antud töös on GoF mustrite all mõeldud just eelmainitud raamatus leiduvaid disainimustreid.
Moodultestimine	Moodultestimise [2] (ingl. k. *unit testing*) puhul testitakse mistahes eraldiseisvat tarkvarakomponenti (meetodit, klassi, mustrit) ehk kontrollitakse, kas tarkvara osad töötavad üksteisest sõltumatult.
MSTest	**MSTest** [3] Microsofti poolt loodud moodultestimise raamistik
Refaktoreerimine	Refaktoreerimine (ingl. k. *refactoring*) on programmikoodi, aga ka riistvarakavandi, andmebaasi vm objekti välist käitumist säilitav struktuuri lihtsustamine ja/või täiustamine. [4]
UML	***Unified Modeling Language*** ehk unifitseeritud modelleerimiskeel UML on üldotstarbeline graafiline notatsioon tarkvara või infosüsteemide visualiseerimiseks. *[5]*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Bakalaureusetöö kirjeldab tarkvara disainimustrite õpetamiseks vajalike eesti keelsete õppevahendite loomist ja valideerimist. Tulemuste põhjal tehakse järeldused materjalide kasulikkuse osas ja püstitatakse küsimusi edasiseks uurimiseks disainimustrite õpetamise teemal.

1.1 Taust ja probleem

Tarkvara disainimustrid on üldised korratavad lahendused tihti esinevatele probleemidele tarkvara disainis [1]. Bakalaureuseõppe tudengid, kes on selgeks saanud programmeerimise alusteadmised, ei ole kirjutanud piisavalt palju tarkvara, et oleks olnud võimalust samalaadsete probleemide korduvaks esinemiseks. Disainimustreid on neile omase keerukuse tõttu raske õppida [6]. Veel keerulisem on neid efektiivselt õpetada bakalaureuse taseme tudengitele [7]. Nimetatud asjaolud, koos eestikeelsete materjalide vähesusega, on loonud vajaduse õppevahendite väljatöötamiseks.

1.2 Ülesande püstitus

Käesoleva töö esmärgiks on välja töötada vahendid valitud tarkvara disainimustrite õpetamiseks. Õppevahendite hulka kuuluvad disainimustrite eestikeelsed kirjeldused, näidiskood koos moodultestide ja seletustega. Enesekontrolliks ja tulemuste valideerimiseks luuakse küsimused ja ülesanded.

Oodatav tulemus on ette valmistada tudengeid mõtlemaks tarkvara disainimisest selliselt, et see vastaks ka keerukamate infosüsteemide vajadustele ning puhta koodi põhimõtetele [8]. Samuti täiendada erialast sõnavara, mis võimaldab üheselt mõista ja lühidalt väljendada abstraktseid struktuure.

1.3 Metoodika

Bakalaureusetöö metoodikaks on valitud disainiteaduse [9] metoodika. Tegemist on tulemuspõhise metoodikaga, mis seab juhised tulemuste valideerimiseks ja itereerimiseks teadustöödes.

Õppematerjalide väljatöötamisel on aluseks võetud Gang of Four disainimustrid [1]. Disainimustrite näidiskood on pärit Dofactory [10] veebilehelt. Lähtekoodi on refaktoreeritud ning sellele on kirjutatud moodultestid.

Tulemuse valideerimine toimub teise aasta äriinfotehnoloogia õppekava bakalaureuseõppe tudengite abil. Esmalt antakse tudengitele lahendada teemakohane kontrolltöö. Seejärel töötavad tudengid õppevahenditega ja semestri lõpul korratakse kontrolltööd. Teadmiste kontrolli tulemuste põhjal koostab autor analüüsi, mis võimaldab hinnata loodud vahendite kasulikkust disainimustrite õpetamisel.

1.4 Töövahendid

Töös esitatud lähtekood on kirjutatud C# [11] programmeerimiskeeles. Moodultestideks on kasutatud MSTest [3] testimise raamistikku. Klassiskeemid on loodud Visual Studio Class Designer [12] tööriista abil. Kontrolltöö küsimuste koostamiseks ja haldamiseks on kasutatud Tallinna Tehnikaülikooli õpikeskkonda [13], mis põhineb Moodle [14] õpihaldussüsteemil.

1.5 Ülevaade tööst

Käesolev töö koosneb kolmest sisupeatükist. Esimeses peatükis „Disainimustrid" tutvustatakse disainimustrite mõistet. Teises peatükis „Lahenduse kirjeldus" kirjeldatakse antud töö objekti, ehk refaktoreeritud koodi, moodulteste, juhendeid ja kontrolltöö küsimusi. Kolmandas peatükis „Analüüs" on esitatud tudengite poolt sooritatud kontrolltööde tulemused ja nende põhjal tehtud analüüs. Viiendas peatükis „Järeldused ja edasised tööd" tehakse järeldused ja arutletakse võimalike täienduste üle, mida tulevikus läbi viia, et disainimustreid efektiivsemalt õpetada.

2 Disainimustrid

Käesolevas peatükis antakse ülevaade disainimustritest, sealhulgas disainimustrite päritolust ja bakalaureusetöös kasutusel olevate mustrite skoobist. Esimeses alapeatükis tuuakse välja disainimustrite olulisust tarkvaratehnikas. Teises alapeatükis kirjeldatakse õppeainet, mille raames toimub bakalaureusetöö käigus valminud õppematerjalide valideerimine. Kolmandas alapeatükis tutvustatakse varasemalt tehtud töid disainimustrite õpetamise teemadel.

Disainimustrid on üldised taaskasutatavad lahendused tihti korduvatele probleemidele. Idee mustrite kasutamiseks sarnaste probleemide lahendamiseks, pärineb arhitekt Christopher Alexanderilt. Alexanderi 1977. aastal avaldatud raamat "A Pattern Language: Towns, Buildings, Construction" [15] pani alguse mustrite arhitektuurilisele kontseptsioonile. Mustrite kasutamist tarkvaraarenduses populariseeris 1994. aastal avaldatud raamat "Design Patterns: Elements of Reusable Object-Oriented Software" [1]. Raamatu autoreid tuntakse ühise nime all „*Gang of Four"* (edaspidi GoF). Programmeerimise teemalisi mustreid on palju eri tüüpe [16], kuid töö keskendub GoF disainimustritele nende laialdase kasutuse tõttu. Käesolevas töös kasutatakse mõisted mustrid, disainimustrid ja GoF disainimustrid samatähenduslikult ning nende all on silmas peetud tarkvara disainimustreid, mida kirjeldatakse varasemalt välja toodud raamatus.

2.1 Disainimustrite olulisus

Disainimustrite tundmine on oluline, et osata paremini disainida tarkvara. Mustrid võimaldavad efektiivsemalt korraldada klassidevahelist struktuuri. Disainimustritega luuakse efektiivseid klassidevahelisi käitumuslikke suhteid. Veel aitavad mustrid objektide loomise loogikat eraldada klasside äriloogikast (ingl. k. *separation of concerns*). Disainimustrite tundmine võimaldab efektiivsemat suhtlust arendusmeeskonna vahel, tänu lisandunud sõnavarale [16][17]. Nimelt selle pärast, et

see võimaldab tarkvarast rääkida kõrgemal abstraktsioonitasemel, kui üksikute klasside tasemel. Nii võib meeskonna liige anda mõtte edasi kasutades kahte lauset, mis mustrite sõnavara tundmata, nõuaks sama kontseptsiooni edastamiseks palju enam lauseid.

2.2 Disainimustrite õpetamine

Disainimustrite õpetamine toimub Tallinna Tehnikaülikoolis õpetatava aine „Infosüsteemide arendus“ raames. Õppeaines kasutatakse peamiselt Microsoft .NET raamistikku [18] ja C# programmeerimiskeelt. Eelnimetatud õppeaine on üles ehitatud puhta koodi põhimõtteid silmas pidades. Puhas kood ehk *Clean Code* [8] on Robert C. Martini poolt kirjutatud tarkvara arenduse parimaid tavasid käsitlev raamat. Puhta koodi printsiipide hulka kuuluvad lähtekoodi lihtsuse säilitamine, enda mitte kordamine, koodi standarditest kinni pidamine, koodi loetavuse tähtsustamine ja koodi testimine.

2.3 Varasemalt tehtud tööd

Tarkvara disainimustrite õpetamist on pikalt peetud oluliseks osaks arvutiteaduse õppest [19]. Tudengid, kelle õppekavasse kuuluvad disanimustrite õpe, leiavad, et mustreid on neile omase keerukuse tõttu raske õppida [6]. Veel keerulisem on disainimustreid efektiivselt bakalaureuse-taseme tudengitele õpetada [7]. Harilik loengu-põhine õpe ei ole piisav, et teadmisi disainimustritest omandada [20]. Selleks, et disainimustrite ideest aru saada, on tarvis näha mustreid kasutatuna reaalsete probleemite lahendamisel [21]. Disainimustrite edukat õpetamist on dokumenteeritud mitmetel juhtudel [7][22][23], kuid kursuse materjalid ja ülesannete lähtekood avalikult kättesaadav ei ole.

3 Lahenduse kirjeldus

Käesoleva töö objektiks on disainimustrite õppematerjalid, mis koosnevad neljast osast.

- Refaktoreeritud disainimustrite lähtekood.
- Moodultestid, mis katavad disainimustrite koodi.
- Disainimustreid seletav dokumentatsioon koos juhendiga.
- Teadmiste kontrolli küsimused.

Järgnevas neljas alapeatükis kirjeldatakse eelnimetatud lahenduse osade koostamise protsessi.

3.1 Algallika koodi täiustamine ja klassiskeemide koostamine

Õppematerjalide loomisel on võetud aluseks Dofactory [10] lehel olevate GoF disainimusrite lähtekood. Autor kaalus disainimustrite lähtekoodi omal käel kirjutamist, kuid see ei osutunud otstarbekaks aja kasutuseks, sest erinevaid GoF disainimustrite lähtekoodi komplekte on võimalik leida lugematul hulgal allikatest. Aluseks võeti ülalnimetatud allikas leiduv mustrite lähtekood sellepärast, et need olid kirjutatud C# programmeerimiskeeles, mis on ka „Infosüsteemide arenduse“ õppeaine põhikeeleks. Lisaks sellele on seal leiduvad elulised näited sobivad, sest annavad tudengitele aimu, kuidas suhestuvad abstraktsed disainimustrid päris maailmaga.

Aluseks võetud lähtekood on küll õppeaine sisuga hästi sobiv, kuid ei vasta puhta koodi printsiipidele, mis on eelnimetatud kursuse keskseks teemaks. Sellepärast leidis autor, et on tarvis algset koodi refaktoreerida. Järgnevalt on kirjeldatud seda protsessi.

1. Allikas olevate mustrite klassid olid kõik ühes failis. Klassid said liigutatud kõik eraldi failidesse.

2. Allikas leidus rohkelt kommentaare. Puhta koodi printsiipide järgi tuleks kommentaare kasutada minimaalselt, nimelt erandjuhtudel, kui ei ole võimalik meetodite ja muutujate nimedest välja lugeda, mis funktsiooni vaatluse all olev koodiplokk täidab. Autor eemaldas valdava enamuse kommentaaridest ja nimetas vajadusel meetodeid ja muutujaid ümber, et kood oleks arusaadavam.

3. Allikas olevate mustrite põhifunktsionaalsust demonstreeriti eraldi klassiga MainApp, mille Main meetodi kaudu toimus kokkupuude disainimustri põhiklasside objektidega. Selle jaoks, et näidata, kuidas mingi muster toimib, kasutati suurel hulgal *Console.WriteLine* käsklusi, mis kirjutavad teksti konsooli. Autor eemaldas *MainApp* klassi ja konsooli kirjutamise käsud. Nende asemel koostas autor moodultestid, millest on lähemalt kirjutatud alapeatükis 3.2.

Täiustatud koodi põhjal lõi autor iga mustri kohta klassiskeemid, kasutades Visual Studio Class Designer [12] tööriista. See tööriist genereerib lähtekoodi põhjal klassidevahelisi seoseid illustreeriva UML klassiskeemi [5]. Kood genereeritakse vastavalt reaalsele lähtekoodile, ehk kood on skeemiga otseses sõltuvuses ja kui muuta ühte, siis muutub ka teine. Viimaks kohandas autor Class Designer tööriista funktsionaalsuseid kasutades genereeritud skeemi, et see oleks visuaalselt paremini hoomatav.

Joonis 1 illustreerib Decorator disainimustri ühe klassi koodi enne refaktoreerimist. Joonisel 2 on sama klass pärast refaktoreerimist. Joonis 3 kujutab refaktoreeritud Decorator mustri lähtekoodi põhjal genereeritud klassiskeemi.

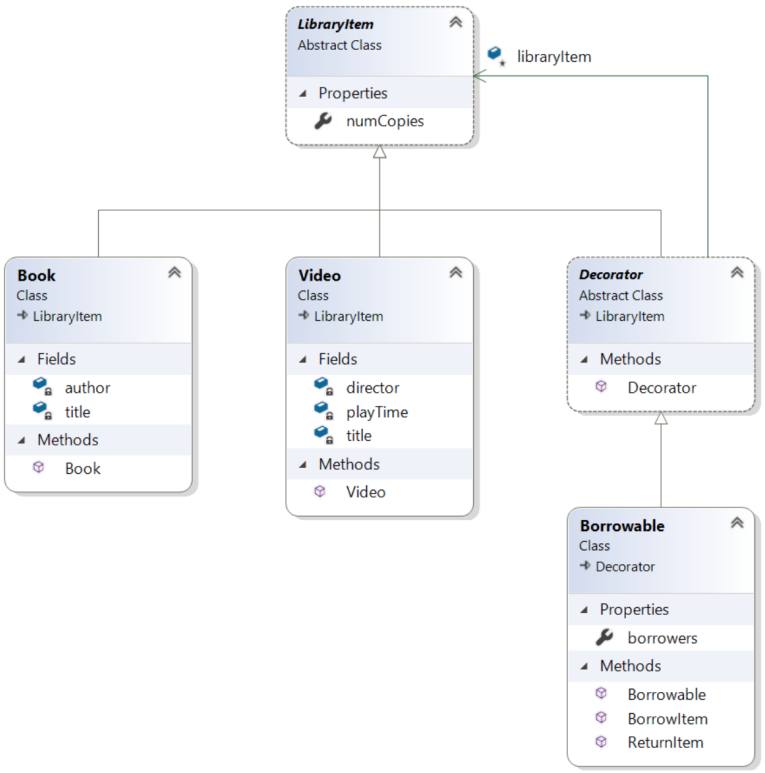
Teiste disainimustrite refaktoreeritud kood on lisatud peatükki „Lisa 1 – Refaktoreeritud disainimustrite lähtekood”.

```
/// <summary>
/// The 'ConcreteDecorator' class
/// </summary>
  class Borrowable : Decorator
  {
    protected List<string> borrowers = new List<string>();

    // Constructor
    public Borrowable(LibraryItem libraryItem)
      : base(libraryItem)
    {
    }

    public void BorrowItem(string name)
    {
      borrowers.Add(name);
      libraryItem.NumCopies--;
    }

    public void ReturnItem(string name)
    {
      borrowers.Remove(name);
      libraryItem.NumCopies++;
    }

    public override void Display()
    {
      base.Display();

      foreach (string borrower in borrowers)
      {
        Console.WriteLine(" borrower: " + borrower);
      }
    }
  }
```

Joonis 1. Borrowable klass enne refaktoreerimist

```
class Borrowable : Decorator
{
    public List<string> Borrowers { get; } = new List<string>();

    public Borrowable(LibraryItem libraryItem)
        : base(libraryItem)
    {
    }

    public void BorrowItem(string name)
    {
        Borrowers.Add(name);
        libraryItem.NumCopies--;
    }

    public void ReturnItem(string name)
    {
        Borrowers.Remove(name);
        libraryItem.NumCopies++;
    }
}
```

Joonis 2. Borrowable klass pärast refaktoreerimist



Joonis 3. Decorator disainimustri klassiskeem

3.2 Moodultestide koostamine

Järgnevalt annab autor ülevaate moodultestidest ja seejärel kirjeldab nende koostamise protsessi. Roy Osherove defineerib enda raamatus „The Art of Unit Testing" [2] moodultestimist järgmiselt: „Moodultest on hulk koodi, mis aktiveerib ühe tööühiku ja kontrollib selle tööühiku ühte kindlat tulemust. Kui lõpptulemuse eeldus osutus valeks, siis on moodultest ebaõnnestunud. Moodultesti skoop võib ulatuda ühest meetodist kuni mitmete klassideni välja." Autor on käesolevas töös lähtunud sellest definitsioonist.

Moodultestid koostati 21-le disainimustrile. Testid on kirjutatud MSTest [3] testimise raamistikuga. Järgnevalt on võetud vaatluse alla *State* disainimustri testimine. *State* võimaldab objektil muuta enda käitumist vastavat sisemise oleku muutustele, kusjuures näib nagu objekt oleks muutnud oma klassi [1].

Testide jaoks on loodud eraldi projekt *Tests*. Selle alla kuulub klass *StateTests*, mis on tähistatud atribuudiga [TestClass]. Sinna on kogutud kõik *State* mustri meetodid.

Selle jaoks, et oleks võimalik disainimustri tööd testida, tuli esmalt deklareerida *Account* klassi objekt. Seejärel instantsieeritakse objekt *Initialize* meetodis. *Initialize* meetod on tähistatud atribuudiga [TestInitialize], mis tähendab, et seda meetodit käivitatakse enne igat moodultesti. Antud lahendus tagab, et testid oleksid üksteisest sõltumatud ehk igale moodultestile luuakse uus *Account* klassi isend *account*. Joonisel 4 on välja toodud eelnevalt nimetatud atribuutid.

```
[TestClass]
public class StateTests
{
    private Account account;

    [TestInitialize]
    public void Initialize()
    {
        account = new Account("Jim Johnson");
    }
```

Joonis 4. TestClass ja TestInitialize atribuudid

Esimeseks moodultestiks on *DepositTest*. Moodultestide meetodid on tähistatud atribuudiga [TestMethod]. Test meetodid tagastavad tüübi void ning nad kasutavad

Assert klassi meetodeid, et kontrollida, kas testitav subjekt vastab oodatavale väärtusele. *DepositTesti* eesmärk on kontrollida, kas *account* objektile ehk kontole raha hoiustades see ka seal kajastub. Uut *account* objekti luues on selle *Balance* ehk kontojäägi väärtus alati võrdne nulliga. Kui pöörduda *Deposit* meetodi poole argumendiga 500.0, siis on oodatavaks tulemuseks, et kontojääk suureneb viiesaja rahaühiku võrra. Seda kontrollib testi *AreEqual* meetod, mille parameetriteks on kaks väärtust - oodatav ja tegelik väärtus. Kui need kaks osutuvad võrdseteks, siis on test läbitud, vastasel juhul testi ei läbita. *DepositTesti* kood on näidatud Joonisel 5.

```
[TestMethod]
public void DepositTest() {
    account.Deposit(500.0);

    Assert.AreEqual(account.Balance, 500.0);
}
```

Joonis 5. DepositTest meetod

DepositTesti puhul on tegemist moodultestiga, mille skoobiks on üks meetod. See tähendab, et kontrollitakse ainult *Account* klassi *Deposit* meetodit. Järgnevalt vaadeldakse moodultesti, mis on skoobilt oluliselt suurem ja hõlmab mitmeid klasse ehk kogu *State* mustrit tervikuna.

ChangeStateFromSilverToGolden testi eesmärgiks on kontrollida oleku muutust SilverState olekust, ehk algolekust, *GoldState* olekusse.

Testi alguses toimub oleku kontroll Assert.AreEqual meetodi abil, pöördudes selle poole argumentidega *account.State.GetType()* ja *typeOf(SilverState)*. Esimeseks argumendiks on pöördumine C#-i *System* nimeruumi, *Object* klassi, meetodi *GetType* poole, mis tagastab objekti tüübi. Teiseks argumendiks on *typeof* operaator, mis võimaldab selle argumendina antud klassi tüüpi avaldistes kasutada. Seega, *Assert.AreEqual(account.State.GetType(), typeof(SilverState))*, kontrollib kas account isendi tüüp vastab klassile *SilverState*.

Seejärel toimub pöördumine *Deposit* meetodi poole argumendiga 10000.0, mille tulemusena suureneb konto bilanss kümne tuhande võrra.

Viimaks toimub uus *Assert.AreEqual* meetodi kutse, kuid seekord on teiseks argumendiks GoldState. Sellega kontrollitakse, kas account isend on saanud olekuks *GoldState*. Eelmises alapeatükis välja toodud *GoldState.cs* failist oli näha, et *GoldState* oleku jaoks on tarvis, et kontol oleks vähemalt 1000 rahaühikut, seega annab test rohelise tulemuse.

Kirjeldatud testi kood on esitatud joonisel 6.

```
[TestMethod]
public void ChangeStateFromSilverToGolden()
{
    Assert.AreEqual(account.State.GetType(), typeof(SilverState));

    account.Deposit(10000.0);

    Assert.AreEqual(account.State.GetType(), typeof(GoldState));
}
```

Joonis 6. Oleku muutuse kontroll ChangeStateFromSilverToGolden meetodi abil

ChangeStateFromSilverToGolden on oma olemuselt erinev varasemalt vaadeldud *DepositTestist*, sest testitavaks tööühikuks on ühe meetodi asemel üks kogum klasse, ehk *State* muster tervikuna. Seda, millist tööühikut kontrollitakse on võimalik järeldada testi sisust ja *Assert* lausete parameetritest. Meetodi tööd testiva *DepositTesti* esimeseks parameetriks on *account.Balance* ehk *account* isendi muutuja *Balance* väärtus. State.cs lähtekoodist on näha, et *Deposit* meetod suurendab isendi *Balance* muutujat.

DepositTest testis toimub pöördumine *Deposit* meetodi poole ja kontrollitakse kontojäägi muutust, millest järeldub, et testimise skoobiks ongi ainult *Deposit* meetod. *Deposit* meetodi kood on välja toodud järgneval joonisel 7.

```
public virtual void Deposit(double amount) {
    Balance += amount;
    StateChangeCheck();
}
```

Joonis 7. State klassi Deposit meetod

ChangeStateFromSilverToGolden testis toimub samuti pöördumine *Deposit* meetodi poole, kuid nüüd kontrollitakse, kas *account* isend on muutnud enda klassi. Toimub kaks kontrolli: esimene kontrollib, et isendi klass oleks tüüpi *SilverState* ja teine, et tüüp oleks *GoldState*. Siit on selgelt näha, et kontrollitav tööühik on vähemalt ühe abstraktsioonitaseme võrra kõrgem, kui eelnevalt. Kui varasemalt piirduti ühe meetodi töö kontrolliga, siis *ChangeStateFromSilverToGolden* testi skoobiks on mitu klassi.

ChangeStateFromSilverToGolden ei kontrolli ainult klasse, vaid disainimustrit tervikuna. Eelnevalt välja toodud definitsiooni järgi võimaldab *State* muster muuta enda objekti käitumist vastavalt sisemise oleku muutustele ning näib nagu objekt oleks muutnud oma klassi – mis on täpselt see, mis testi käivitamisel juhtub ja mida testi *Assert* laused kontrollivad.

3.3 Juhendi koostamine

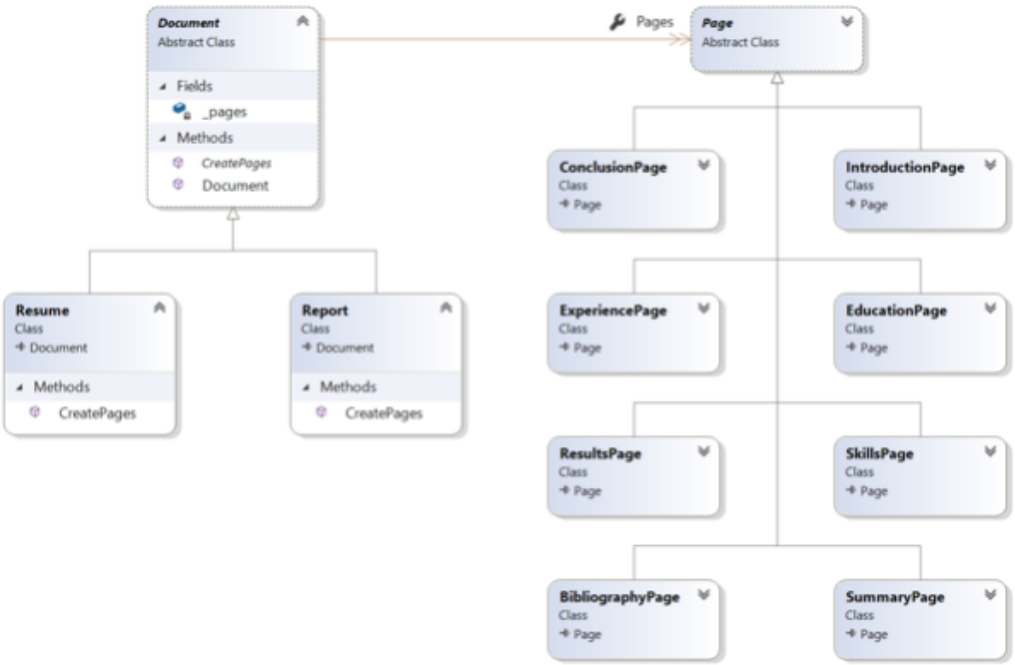
Autor koostas seletavad juhendid, mille järgi on tudengitel võimalik iseseisva õppetöö käigus omandada teadmised disainimustritest. Järgnevalt on välja toodud juhendi struktuur.

- Nimi – Disainimustrit iseloomustav nimi.
- Definitsioon – Lühike disainimustri kirjeldus, mis vastab järgnevatele küsimustele: Mida see disainimuster teeb? Mis on mustri kavatsus? Millist disainiprobleemi see muster lahendab? Definitsioon on pärit raamatust „Design Patterns: Elements of Reusable Object-Oriented Software.” [1]
- Klassiskeem – Mustri graafiline esitus. Klassiskeemid on kirjeldatud UML [5] noteerimiskeeles ja loodud kasutades Visual Studio Class Designer [12] tööriista.
- Seletav kirjeldus – Disainimustri ja sellele vastavate testide lähtekoodi üksikasjalikult seletav kirjeldus.

Joonisel 8 on kujutatud *Factory Method* mustri juhendi esimest lehekülge, mis sisaldab endas nime, definitsiooni, klassiskeemi ning osa seletavast kirjeldusest.

Factory Method

Definitsioon: Defineerib liidese objektide loomiseks, kuid laseb alamklassidel otsustada, millist klassi instantsieerida. Võimaldab klassil instantsieerimise edasi anda alamklassidele. [GoF, lk 107]



Factory Method mustri mõte on luua meetod objektide loomiseks (Document klassi CreatePages), mida alamaklassid (Resume ja Report) realiseerivad. Antud näites koosnevad alamklassid Resume ja Report erinvat tüüpi lehtedest. Resume koosneb lehtedest SkillsPage, EducationPage ja ExperiencePage - need lehed lisatakse Resume objekti loomisel konstruktori poolt kutsutud CreatePages meetodi poolt automaatselt, kui luuakse uus seda tüüpi objekt. Uue Report tüüpi objekti loomisel lisatakse sellele sama põhimõtte järgi teised viis Page alamklassi (vaata all olevat koodi).

```
abstract class Document
{
    protected Document() => CreatePages();

    public List<Page> Pages { get; } = new List<Page>();

    public abstract void CreatePages();
}
```

Joonis 8. Factory Method disainimustri juhend

3.4 Teadmiste kontrolli küsimuste koostamine

Autor koostas disainimustrite alaste teadmiste kontrolliks 60 kontrolltöö küsimust. Küsimused olid valikvastustega ja peamiselt teoreetilist laadi. Kontrolltöö küsimused sisestati Tallinna Tehnikaülikooli õpikeskkonda [13]. Samas keskkonnas sooritasid „Infosüsteemide arenduse" õppeaine tudengid küsimuste põhjal kolm kontrolltööd, mida kirjeldatakse lähemalt peatükis „Analüüs".

Küsimused jaotusid nelja kategooriasse.

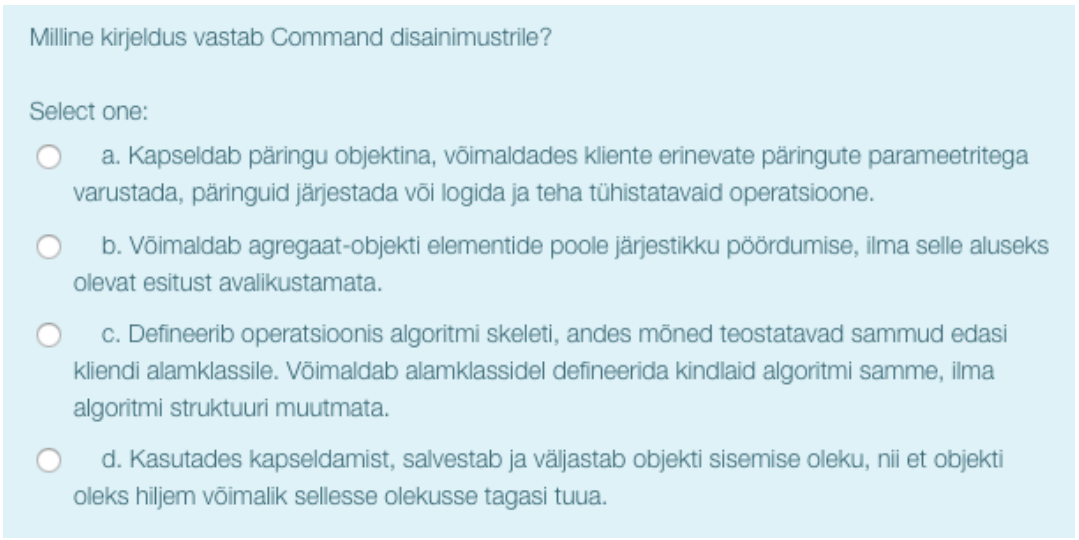
1. Käitumuslikud mustrid – Antud kategooriasse kuuluvad küsimused disainimustrite kohta, mis vastutavad tegevuste efektiivse jaotuse eest programmi eri osade vahel. Design Patterns [1] raamatu kohaselt on seda tüüpi mustriteks järgnevad: *Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method* ja *Visitor*.

2. Loomismustrid – Antud kategooriasse kuuluvad küsimused objektide loomisega seotud disainimustrite kohta. Loomismustriteks hulka loetakse järgnevaid mustreid: *Abstract Factory, Builder, Factory Method, Prototype* ja *Singleton*.

3. Struktuurimustrid – Antud kategooriasse kuuluvad küsimused disainimustrite kohta, mis vastutavad efektiivsete klassihierarhiate ja klassidevaheliste suhete eest. Struktuurimustriteks loetakse järnevaid disainimustreid: *Adapter, Bridge, Composite, Decorator, Facade, Flyweight* ja *Proxy.*

4. Üldised küsimused – Antud kategooriasse kuuluvad disainimustrite üldteadmisi puudutavad küsimused või mitut varasemalt välja toodud kategooriat hõlmavad küsimused.

Joonisel 9 on välja toodud üks käitumuslike mustrite hulka kuuluv kontrolltöö küsimus Tallinna Tehnikaülikooli õpikeskkonnas. Ülejäänud küsimused asuvad peatükis „Lisa 4 – Kontrolltöö küsimused".



Joonis 9. Käitumuslike mustrite kategooria kontrolltöö küsimus õpikeskkonnas

4 Analüüs

Käesolevas peatükis kirjeldatakse õppematerjalide valideerimise metoodikat ja tuuakse välja valideerimise osana tehtud kolme kontrolltöö tulemused. Tulemuste põhjal teostatakse analüüs.

Õppematerjalide valideerimiseks ja Infosüsteemide arendamise kursuse raames korraldatud disainimustrite õppemetoodika hindamiseks, korraldati tudengitele kolm kontrolltööd, mille vahel oli kaks kahenädalast õppeperioodi. Kontrolltöö küsimusi oli kokku 60 ja need koosnesid peamiselt teoreetilistest küsimustest GoF disainimustrite teemal.

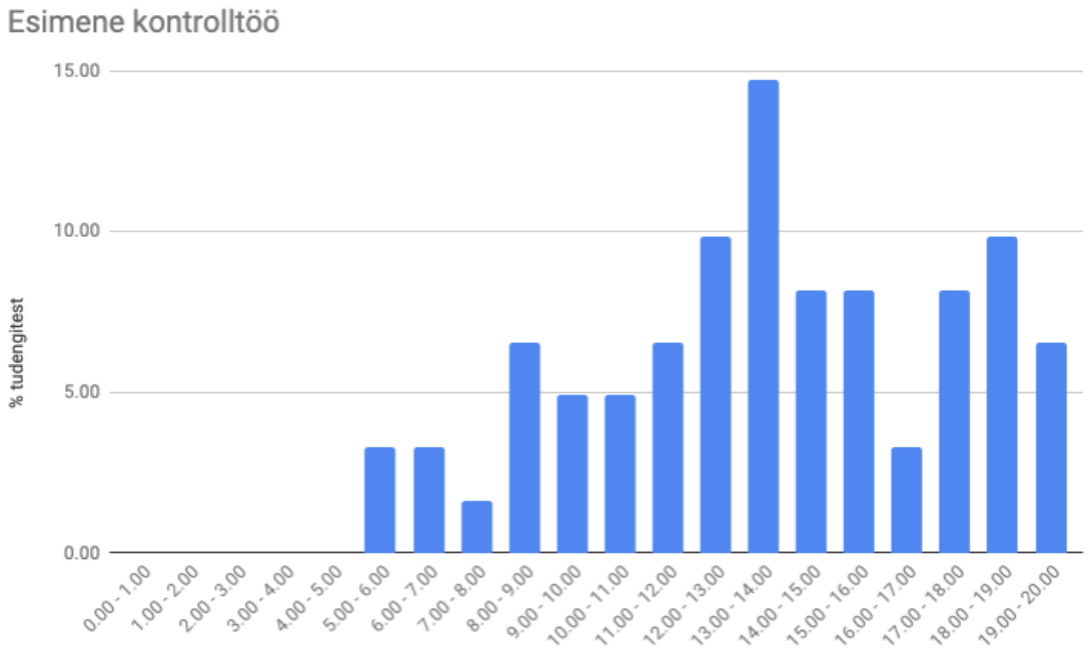
Disainimustrite omandamise hindamiseks anti tudengitele ülesandeks kõigepealt lahendada kahekümne küsimusega kontrolltöö, ilma materjalidega eelnevalt tutvumata. Seejärel jagati kätte käesoleva bakalaureusetöö raames koostatud juhendid, millega tuli tudengitel kaks nädalat iseseisvat tööd teha. Eelnimetatud õppeperioodi lõpus tehti kontrolltöö uuesti, seekord tuli vastata kolmekümnele küsimusele. Järgnes kahenädalane õppeperiood, mille hulka kuulusid loengud disainimustrite teemal ja „Design Patterns: Elements of Reusable Object-Oriented Software" raamatu läbi töötamine. Viimaks sooritasid tudengid kontrolltöö kolmandat korda.

Järgneb kontrolltöö tulemuste võrdlev analüüs ja järeldused valitud metoodika ja õppematerjalide efektiivsuse kohta.

4.1 Esimene kontrolltöö

Esimese kontrolltöö sooritamisel ei eeldatud osalejatelt eelnevat kokkupuudet tarkvara disainimustritega. Kontrolltöö koosnes 20-st juhuslikult valitud küsimusest ja seda sooritas 61 tudengit. Testi sooritamiseks oli aega 30 minutit ja maksimaalne võimalik punktisumma oli 20.

Testi tulemused grupeeriti täisarvulistesse vahemikesse nullist kahekümneni vastavalt punktisummale. Joonisel 10 on kujutatud esimese kontrolltöö punktide jaotus.



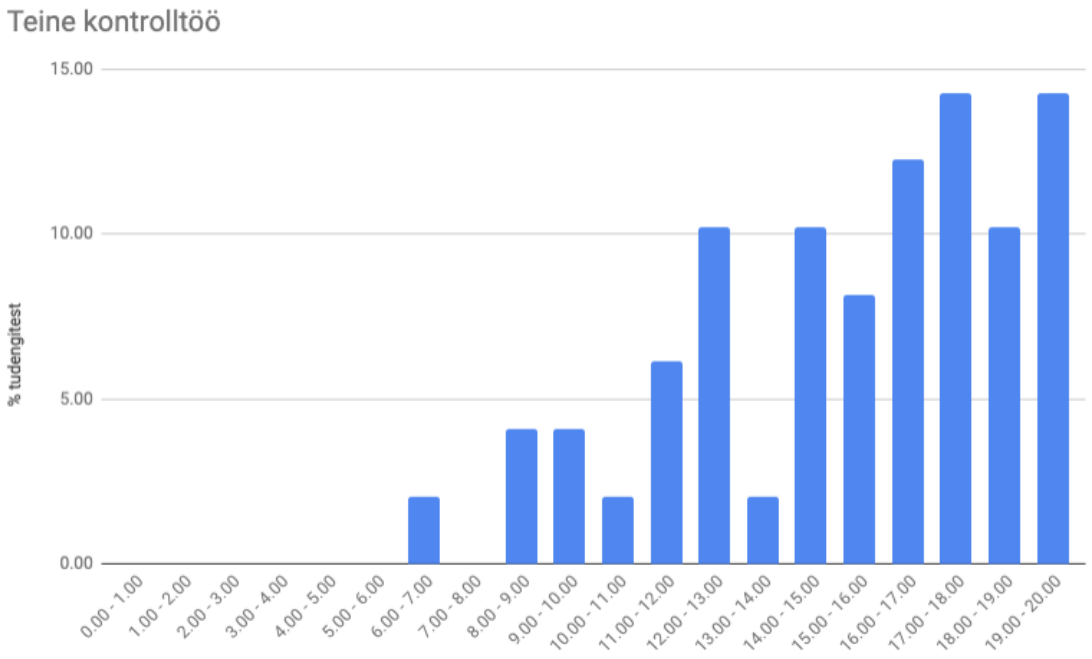
Joonis 10. Esimese kontrolltöö punktide jaotus

Kontrolltöö keskmiseks tulemuseks oli 13.33 punkti. Kõige madalam punktisumma oli 5 punkti ja kõrgeim 19.17 punkti.

4.2 Teine kontrolltöö

Teise kontrolltöö sooritamisel olid tudengid töötanud kaks nädalat käesoleva lõputöö raames koostatud õppematerjalidega. Kontrolltöö koosnes 30-st juhuslikult valitud küsimusest ja seda sooritas 49 tudengit. Testi sooritamiseks oli aega 30 minutit ja maksimaalne võimalik punktisumma oli 20.

Sarnaselt eelmisele kontrolltööle, grupeeriti tulemused täisarvulistesse vahemikesse nullist kahekümneni, vastavalt punktisummale. Joonisel 11 on kujutatud teise kontrolltöö punktide jaotus.



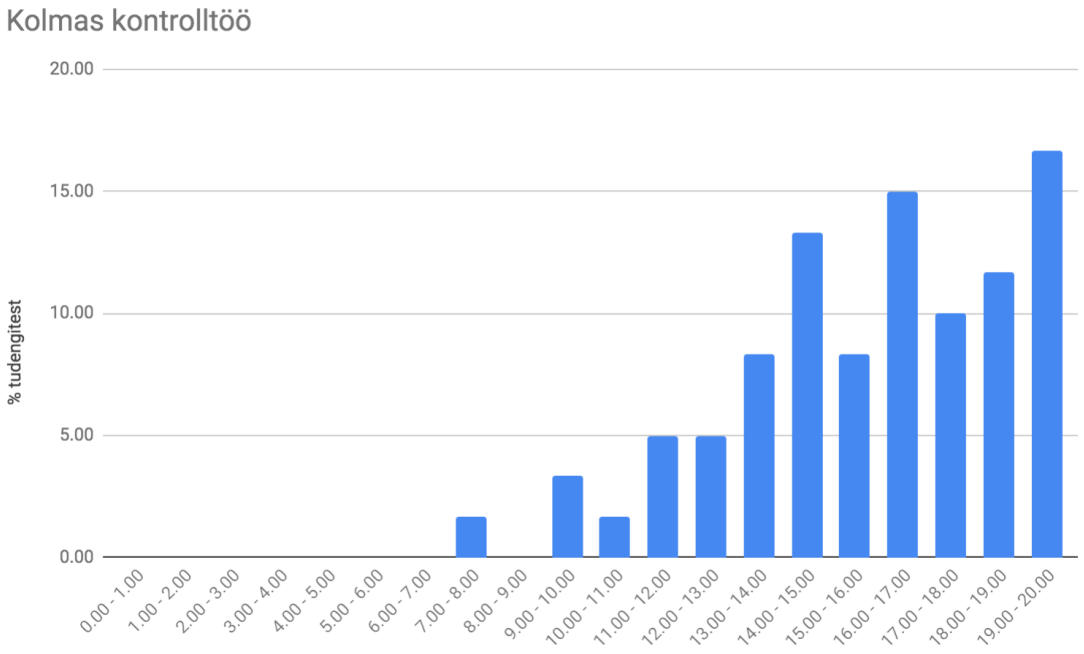
Joonis 11. Teise kontrolltöö punktide jaotus

Kontrolltöö keskmiseks tulemuseks oli 15.23 punkti. Kõige madalam punktisumma oli 6 punkti ja kõrgeim 20 punkti.

4.3 Kolmas kontrolltöö

Kolmanda kontrolltöö sooritamisel olid tudengid, lisaks eelnevale, osalenud auditoorses õppetöös ning neilt eeldati Design Patterns [1] raamatu läbi lugemist. Kontrolltöö koosnes 30-st juhuslikult valitud küsimusest ja seda sooritas 60 tudengit. Testi sooritamiseks oli aega 30 minutit ja maksimaalne võimalik punktisumma oli 20.

Joonisel **12** on kujutatud kolmanda kontrolltöö punktide jaotus.



Joonis 12. Kolmanda kontrolltöö punktide jaotus

4.4 Kontrolltöö tulemuste võrdlus

Tudengid sooritasid kursuse käigus kolm disainimustrite teemalist kontrolltööd. Kõige suurem osalus oli esimesel kontrolltööl, mida sooritasid 61 tudengit. Minimaalseks punktisummaks kolme töö peale, oli esimesel tööl 5 punkti ja suurimaks punktisummaks 20 punkti, mida saavutati kahel viimasel kontrolltööl.

Esimese kontrolltöö küsimuste arv oli 20, teise 30 ja kolmanda 30. Seejuures keskmine, minimaalne ja maksimaalne punktisumma igal sooritusel paranes.

Kontrolltööde tulemused on esitatud tabelis 1.

	Esimene kontrolltöö	Teine kontrolltöö	Kolmas kontrolltöö
Osalejate arv	61	49	60
Küsimuste arv	20	30	30
Keskmine punktisumma	13.33	15.23	15.83
Minimaalne punktisumma	5.00	6.00	7.33
Maksimaalne punktisumma	19.17	20.00	20.00

Tabel 1. Kontrolltöö tulemuste võrdlus

5 Järeldused ja edasised tööd

Käesolevas peatükis kirjeldab autor enda kogemust disainimustrite õppimisel ja selle seotust käesoleva töö objektiga. Seejärel arutletakse õppematerjalide ja valideerimismetoodika efektiivsuse üle ning tuuakse välja võimalikud edasised tööd.

5.1 Autori kogemus disainimustrite õppimisel

Autor alustas disainimustrite kohta uurimist eesmärgiga koostada eesti keelsed õppematerjalid, mille abil on võimalik bakalaureuseõppe tudengitel iseseisvalt disainimustreid õppida. Selle jaoks, et materjale koostata, tuli kõigepealt autoril endal teema põhjalikult läbi töötada. Õppeprotsessi on kirjeldatud järgnevates lõikudes.

Esimeseks õppevahendiks valis autor raamatu „Head First Design Patterns" [24], milles kasutatakse õppeprotsessi kiirendamiseks ja õpitu efektiivsemaks kinnistamiseks õppeprintsiipe nagu metakognitsioon, informatsiooni graafiline esitamine ja tähelepanu püüdvad tehnikad. Selle jaoks, et disainimustrite alaseid teadmisi süvendada, läbis autor erinevaid videopõhiseid veebiloenguid [25][26] ja kirjutas C# programmeerimiskeeles konsoolirakenduse DeliveryApp, milles olid erinevad mustrid kasutusel. DeliveryApp kujutas endast logistikaettevõtte tarne planeerimist haldavat rakendust, kus põhirõhk oli disainimustrite kasutamisel ja äriloogika osakaal oli minimaalne. Rakendus on leitav käesoleva töö peatükis „Lisa 5 – DeliveryApp rakendus". Selle rakenduse implementeerimise kohta koostas autor ka seletava juhendi, mida on võimalik kasutada õppematerjalina tutvumaks nelja disainimustriga. Õppematerjalina seda veel valideeritud ei ole ja seetõttu käesolev bakalaureusetöö seda rohkem ei käsitle.

Saadud kogemusele tuginedes, leiab autor, et disainimustrite õppimine, piiratud programmeerimise kokkupuute juures, on tõepoolest keeruline. Arusaamise omandamiseks töötati eri tüüpi vahenditega: töö raamatuga, videoloengud ja rakenduse programmeerimine. Alles pärast seda arvas autor, et omab piisavalt kogemust käesoleva töö objektiks olevate õppematerjalide väljatöötamise ette võtta.

5.2 Hinnang õppevahenditele ja edasised tööd

Kontrolltöö tulemuste põhjal võib järeldada, et käesoleva bakalaureusetöö raames koostatud õppevahenditest oli kasu disainimustrite omandamisel. Autoril on raske hinnata, kui edukalt loodud õppevahendid teadmisi edasi annavad, kuna puudus võimalus võrrelda teiste materjalide õppetulemustega. Selle jaoks, et nende efektiivsust paremini mõõta, võiks tulevikus jagada disainimustreid õppivad tudengid kaheks – esimesele poolele anda kätte töös käsitletud materjalid ja teisele poolele mõni teine õppematerjal. Selline lahendus võib anda võrdlusmomendi, kuid ei pruugi sobida kursusele, kus põhilise õppevahendina peab olema eesti keelne materjal, kuna töö kirjutamise hetkel teisi disainimustrite eesti keelseid õppematerjale autor leidnud ei olnud. Lisaks kontrolltöö küsimustele, mis peamiselt kontrollivad teoreetilisi teadmisi, võiks lisada programmeerimist nõudvaid ülesandeid. Disainimustrite programmeerimisülesannete automaatne valideerimine on mustrite olemuse tõttu keeruline [27]. Seega üks võimalik edasine uurimisteema oleks, kuidas kontrollida tudengite disainimustrite alaseid teadmisi õpikeskkondades nagu Moodle [14].

Õppematerjale on võimalik täiendada, luues seletavad juhendid refaktoreeritud lähtekoodi ja moodultestide tööpõhimõtte kirjeldamiseks mustritel, millel neid veel ei ole. Samuti, on oluline tuvastada, milliseid mustreid on lihtsam õppida ning seejärel kohandada mustrite esitamise järjekorda õppevahendites.

6 Kokkuvõte

Käesoleva töö eesmärgiks oli välja töötada eesti keelsed vahendid tarkvara disainimustrite õpetamiseks. Loodud õppevahendite hulka kuuluvad tuntumate disainimustrite lähtekood, neile vastavad moodultestid ja mustrite tööpõhimõtet seletavad juhendid. Töö käigus refaktoreeriti „Gang of Four" disainimustrite lähtekood selliselt, et see vastaks puhta koodi põhimõtetele ning koodi põhjal genereeriti klassiskeemid. Disainimustritele kirjutati moodultestid MSTest raamistikus ja kirjeldati *State* mustri testimise protsessi. Võrreldi testimise erinevusi meetodi ja mustri tasemel. Autor koostas juhendid iseseisvaks õppetööks, mis sisaldasid mustrite definitsoone, klassiskeeme ja seletavaid kirjeldusi. Disainimustrite teadmiste kontrolliks loodi valikvastustega küsimused.

Õppematerjale valideeriti bakalaureuseõppe tudengite abil, „Infosüsteemide arendamise" õppeaine raames. Valideerimiseks kasutati antud töö käigus loodud kontrolltöö küsimusi. Küsimuste põhjal koostati kolm kontrolltööd, mille vahel oli kaks kahenädalast õppeperioodi. Esimene kontrolltöö sooritati enne õppematerjalidega tutvumist. Teise kontrolltöö ajaks olid tudengid iseseisvalt töötanud õppematerjalidega kaks nädalat. Järgnes kaks nädalat disainimustrite teemalisi loenguid ja õppetöö „Design Patterns: Elements of Reusable Object-Oriented Software" raamatuga, mille järel tehti kolmas kontrolltöö.

Kontrolltööde tulemused paranesid iga sooritusega, millest autor järeldab, et õppevahenditest oli kasu disainimustrite teadmiste omandamisel. Lõputöös arutleti võimalike tulevaste tööde üle disainimustrite õpetamise ja õppematerjalide koostamise teemal.

Kasutatud kirjandus

1: E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", 1994, Addison-Wesley
2: Roy Osherove, "The Art of Unit Testing: with examples in C#", 2013, Manning Publications
3: Unit testing C# with MSTest and .NET Core. [WWW] https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest (11.05.2019)
4: Refaktoreerimine. [WWW] https://akit.cyber.ee/term/1617-refactoring (11.05.2019)
5: Martin Fowler, "UML Distilled", 1997, Addison-Wesley
6: Michael R. Wick, "Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life", 2005, Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, SIGCSE
7: Y. Tao, G. Liu, J. Mottok, R. Hackenberg, G. Hagel, "Just-in-Time-Teaching experience in a Software Design Pattern course", 2015, 2015 IEEE Global Engineering Education Conference (EDUCON)
8: Robert C. Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", 2008, Prentice Hall
9: Design Science in Information Systems Research - WISE. [WWW] https://wise.vub.ac.be/sites/default/files/thesis_info/design_science.pdf (11.05.2019)
10: .NET Design Patterns. [WWW] https://www.dofactory.com/net/design-patterns (11.05.2019)
11: C# Guide. [WWW] https://docs.microsoft.com/en-us/dotnet/csharp/ (11.05.2019)
12: Design and view classes and types with Class Designer. [WWW] https://docs.microsoft.com/en-us/visualstudio/ide/class-designer/designing-and-viewing-classes-and-types?view=vs-2019 (11.05.2019)
13: Tallinna Tehnikaülikooli õpikeskkond. [WWW] https://ained.ttu.ee/ (11.05.2019)
14: Moodle.org. [WWW] https://moodle.org/ (11.05.2019)
15: Christopher Alexander, "A Pattern Language: Towns, Buildings, Construction", 1977, Oxford University Press
16: Foutse Khomh, Yann-Gaël Guéhéneuc, "Design Patterns Impact on Software Quality: Where Are the Theories?", 2018, 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)
17: Jimmy Nilsson, "Applying Domain-Driven Design and Patterns", 2006, Addison-Wesley
18: Announcing the .NET Framework 4.8. [WWW] https://devblogs.microsoft.com/dotnet/announcing-the-net-framework-4-8/ (11.05.2019)
19: Hong Huang, Dongyong Yang, "Teaching Design Patterns: A Modified PBL Approach", 2008, The 9th International Conference for Young Computer Scientists
20: M. Antonio, G. Jiménez-Díaz, J. Arroyo, "Teaching Design Patterns Using a Family of Games", 2009, ACM SIGCSE Bulletin, 41(3)
21: John Hamer, "An Approach to Teaching Design Patterns using Musical Composition", 2004, ACM SIGCSE Bulletin, 36(3)
22: Paul Gestwicki, FuShing Sun, "Teaching Design Patterns Through Computer Game Development", 2008, Journal on Educational Resources in Computing
23: S. Stuurman, G. Florijn, "Experiences with Teaching Design Patterns", 2004, ACM SIGCSE Bulletin 36(3)
24: E. Freeman, B. Bates, K. Sierra, E. Robson, "Head First Design Patterns: A Brain-Friendly Guide", 2004,
25: C# Design Patterns: Part 1. [WWW] https://www.linkedin.com/learning/c-sharp-design-patterns-part-1 (11.05.2019)
26: Design Patterns in Object Oriented Programming. [WWW] https://www.youtube.com/playlist?list=PLrhzvIcii6GNjpARdnO4ueTUAVR9eMBpc (11.05.2019)
27: S. Retalis, P. Georgiakakis, Y. Dimitriadis, "Eliciting Design Patterns for E-Learning Systems", 2006, Computer Science Education, 16:2

Lisa 1 – Refaktoreeritud disainimustrite lähtekood

Järgnevalt on välja toodud töös käsitletud disainimustrite (Decorator ja State) refaktoreeritud lähtekood. Teiste disainimustrite lähtekood on kättesaadav GitHubi hoidlas aadressil:

https://github.com/kaarelsoot/DesignPatterns/tree/master/Study%20materials/GoF

Decorator

Book.cs

```
namespace GoF.Structural.Decorator
{
    class Book : LibraryItem
    {
        public string Author { get; }
        public string Title { get; }

        public Book(string author, string title, int numCopies)
        {
            Author = author;
            Title = title;
            NumCopies = numCopies;
        }
    }
}
```

Borrowable.cs

```
using System.Collections.Generic;

namespace GoF.Structural.Decorator
{
    class Borrowable : Decorator
    {
        public List<string> Borrowers { get; } = new List<string>();

        public Borrowable(LibraryItem libraryItem)
            : base(libraryItem)
        {
        }

        public void BorrowItem(string name)
        {
            Borrowers.Add(name);
            libraryItem.NumCopies--;
        }

        public void ReturnItem(string name)
        {
            Borrowers.Remove(name);
            libraryItem.NumCopies++;
        }
    }
}
```

Decorator.cs

```
namespace GoF.Structural.Decorator
{
    abstract class Decorator : LibraryItem
    {
        protected LibraryItem libraryItem;

        public Decorator(LibraryItem libraryItem)
        {
            this.libraryItem = libraryItem;
        }
    }
}
```

LibraryItem.cs

```
namespace GoF.Structural.Decorator
{
    abstract class LibraryItem
    {
        public int NumCopies { get; set; }
    }
}
```

Video.cs

```
namespace GoF.Structural.Decorator
{
    class Video : LibraryItem
    {
        public string Director { get; }
        public string Title { get; }
        public int PlayTime { get; }

        public Video(string director, string title,
            int numCopies, int playTime)
        {
            Director = director;
            Title = title;
            NumCopies = numCopies;
            PlayTime = playTime;
        }
    }
}
```

State

Account.cs

```
using System;

namespace GoF.Behavioral.State
{
    public class Account {

        public Account(string owner)
        {
            this.owner = owner;
            State = new SilverState(0.0, this);
        }

        private string owner;
        public double Balance
        {
            get { return State.Balance; }
        }

        public State State { get; set; }

        public void Deposit(double amount) {
            State.Deposit(amount);
```

```
            Console.WriteLine("Deposited {0:C} --- ", amount);
            Console.WriteLine(" Balance = {0:C}", Balance);
            Console.WriteLine(" Status = {0}", State.GetType().Name);
            Console.WriteLine("");
        }

        public void Withdraw(double amount) {
            State.Withdraw(amount);
            Console.WriteLine("Withdrew {0:C} --- ", amount);
            Console.WriteLine(" Balance = {0:C}", Balance);
            Console.WriteLine(" Status = {0}\n", State.GetType().Name);
        }

        public void PayInterest() {
            State.PayInterest();
            Console.WriteLine("Interest Paid --- ");
            Console.WriteLine(" Balance = {0:C}", Balance);
            Console.WriteLine(" Status = {0}\n", State.GetType().Name);
        }
    }
}
```

GoldState.cs

```
namespace GoF.Behavioral.State
{
    public class GoldState : State {

        public GoldState(State state)
            : this(state.Balance,state.Account) {
        }

        public GoldState(double balance,Account account) {
            Balance = balance;
            Account = account;
            Initialize();
        }

        private void Initialize() {
            interest = 0.05;
            lowerLimit = 1000.0;
            upperLimit = 10000000.0;
        }

        public override void PayInterest() {
            Balance += interest * Balance;
            StateChangeCheck();
        }

        protected override void StateChangeCheck() {
            if(Balance < 0.0) {
                Account.State = new RedState(this);
            } else if(Balance < lowerLimit) {
                Account.State = new SilverState(this);
            }
        }
    }
}
```

RedState.cs

```
using System;

namespace GoF.Behavioral.State
{
    public class RedState : State {

        private double serviceFee;

        public RedState(State state) {
            Balance = state.Balance;
            Account = state.Account;
            Initialize();
        }

        private void Initialize() {
            interest = 0.0;
            lowerLimit = -100.0;
            upperLimit = 0.0;
            serviceFee = 15.00;
```

```
        }

        public override void Withdraw(double amount) {
            amount = amount - serviceFee;
            if (amount < 0)
            {
                Console.WriteLine("No funds available for withdrawal!");
            }
        }

        public override void PayInterest() {
            // No interest is paid
        }

        protected override void StateChangeCheck() {
            if(Balance > upperLimit) {
                Account.State = new SilverState(this);
            }
        }
    }
}
```

SilverState.cs

```
namespace GoF.Behavioral.State
{
    public class SilverState : State {

        public SilverState(State state) :
            this(state.Balance,state.Account) {
        }

        public SilverState(double balance, Account account) {
            Balance = balance;
            Account = account;
            Initialize();
        }

        private void Initialize() {
            interest = 0.0;
            lowerLimit = 0.0;
            upperLimit = 1000.0;
        }

        protected override void StateChangeCheck() {
            if(Balance < lowerLimit) {
                Account.State = new RedState(this);
            } else if(Balance > upperLimit) {
                Account.State = new GoldState(this);
            }
        }
    }
}
```

State.cs

```
namespace GoF.Behavioral.State {
    public abstract class State {

        protected double interest;
        protected double lowerLimit;
        protected double upperLimit;
        public Account Account { get; set; }
        public double Balance { get; set; }

        public virtual void Deposit(double amount) {
            Balance += amount;
            StateChangeCheck();
        }

        public virtual void Withdraw(double amount) {
            Balance -= amount;
            StateChangeCheck();
        }

        public virtual void PayInterest() {
            Balance += interest * Balance;
            StateChangeCheck();
        }
```

```
        protected abstract void StateChangeCheck();
    }
}
```

Lisa 2 – Disainimustrite moodultestide lähtekood

Järgnevalt on välja toodud töös käsitletud disainimustrite moodultestide lähtekood. Teiste testide lähtekood on kättesaadav GitHubi hoidlas aadressil:

https://github.com/kaarelsoot/DesignPatterns/tree/master/Study%20materials/Tests

DecoratorTests.cs

```
using GoF.Structural.Decorator;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Tests.Structural
{
    [TestClass]
    public class DecoratorTests
    {
        private Borrowable borrowableVideo;
        private Video video;
        private string customer;

        [TestInitialize]
        public void Initialize()
        {
            video = new Video("Spielberg", "Jaws", 23, 92);
            borrowableVideo = new Borrowable(video);
            customer = "Customer #1";
        }

        [TestMethod]
        public void BorrowableItemHasCorrectMethodTest()
        {
            Assert.IsFalse(HasMethod(video, "BorrowItem"));
            Assert.IsTrue(HasMethod(borrowableVideo, "BorrowItem"));
        }

        [TestMethod]
        public void BorrowItemTest()
        {
            Book book = new Book("Worley", "Inside ASP.NET", 10);
            Borrowable borrowableBook = new Borrowable(book);

            int initialAvailableCopies = book.NumCopies;
            borrowableBook.BorrowItem(customer);

            Assert.AreEqual(initialAvailableCopies - 1, book.NumCopies);
            CollectionAssert.Contains(borrowableBook.Borrowers, customer);
        }

        [TestMethod]
        public void ReturnItemTest()
        {
            borrowableVideo.Borrowers.Add(customer);

            int initialAvailableCopies = video.NumCopies;
            borrowableVideo.ReturnItem(customer);

            Assert.AreEqual(initialAvailableCopies + 1, video.NumCopies);
            CollectionAssert.DoesNotContain(borrowableVideo.Borrowers, customer);
        }

        private bool HasMethod(LibraryItem objectToCheck, string methodName)
```

```
        {
            var type = objectToCheck.GetType();
            return type.GetMethod(methodName) != null;
        }
    }
}
```

StateTests.cs

```
using GoF.Behavioral.State;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Tests.Behavioral
{
    [TestClass]
    public class StateTests
    {
        private Account account;

        [TestInitialize]
        public void Initialize()
        {
            account = new Account("Jim Johnson");
        }

        [TestMethod]
        public void DepositTest() {
            account.Deposit(500.0);

            Assert.AreEqual(account.Balance, 500.0);
        }

        [TestMethod]
        public void WithdrawalTest()
        {
            account.Withdraw(100.0);

            Assert.AreEqual(account.Balance, -100.0);
        }

        [TestMethod]
        public void PayInterestIfSilverStateTest()
        {
            account.Deposit(500.0);
            account.PayInterest();

            Assert.AreEqual(account.State.GetType(), typeof(SilverState));
            Assert.AreEqual(account.Balance, 500.0);
        }

        [TestMethod]
        public void PayInterestIfGoldenStateTest()
        {
            account.Deposit(10000.0);
            account.PayInterest();

            Assert.AreEqual(account.State.GetType(), typeof(GoldState));
            Assert.AreEqual(account.Balance, 10500.0);
        }

        [TestMethod]
        public void ChangeStateFromSilverToGolden()
        {
            Assert.AreEqual(account.State.GetType(), typeof(SilverState));

            account.Deposit(10000.0);

            Assert.AreEqual(account.State.GetType(), typeof(GoldState));
        }

        [TestMethod]
        public void ChangeStateFromSilverToRed()
        {
            Assert.AreEqual(account.State.GetType(), typeof(SilverState));

            account.Withdraw(500);

            Assert.AreEqual(account.State.GetType(), typeof(RedState));
        }
    }
```

Lisa 3 – Juhendid

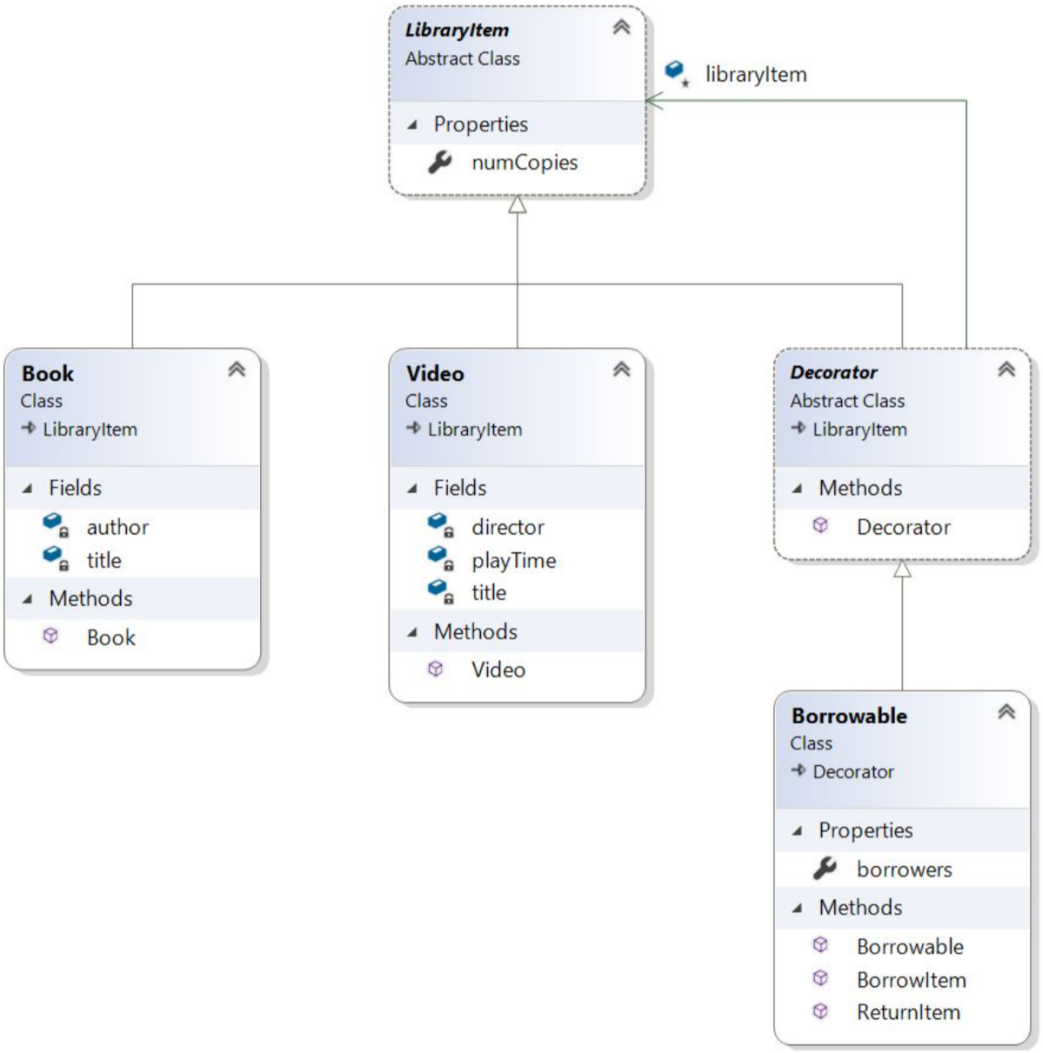
Järgnevalt on välja toodud töös käsitletud disainimustrite (Decorator ja State) kohta koostatud juhendid. Ülejäänud juhendid asuvad GitHubi hoidlas, aadressil:

https://github.com/kaarelsoot/DesignPatterns/blob/master/Tutorials.pdf

Decorator juhend

Decorator

Definitsioon: Määrab objektile dünaamiliselt lisa kohustusi. Pakub alamklasside loomise asemel funktsionaalsuse laiendamiseks paindlikku alternatiivi. [GoF, lk 175]



Antud näites on rakendatud Decorator mustrit, et lisada raamatukogus olevatele raamatutele (Book) ja videotele (Video) laenutamise funktsionaalsus, kasutades selleks Borrowable klassi. Ilma Decoratorit kasutamata peaks laenutamise funktsionaalsuse lisama igale eseme tüübile eraldi, mis võib erinevate klasside arvu tõttu olla ebapraktiline. Samuti ei pruugi alati olla võimalik iga klassi koodi muuta - näiteks teekidest pärit klasside puhul.

```
class Borrowable : Decorator
{
    public List<string> Borrowers { get; } = new List<string>();

    public Borrowable(LibraryItem libraryItem)
        : base(libraryItem)
    {
    }

    public void BorrowItem(string name)
    {
        Borrowers.Add(name);
        libraryItem.NumCopies--;
    }

    public void ReturnItem(string name)
    {
        Borrowers.Remove(name);
        libraryItem.NumCopies++;
    }
}
```

Raamatutel ja filmidel eksisteerib väärtus numCopies, mis näitab kui mitu eset raamatukogus on. Kui raamat või film on dekoreeritud Borrowable klassiga ja kasutada sellega kaasnevat meetodit BorrowItem(), siis väheneb algse objekti numCopies väärtus. Seda illustreerib test failis **DecoratorTests.cs**

```
[TestMethod]
public void BorrowItemTest()
{
    Book book = new Book("Worley", "Inside ASP.NET", 10);
    Borrowable borrowableBook = new Borrowable(book);

    int initialAvailableCopies = book.NumCopies;
    borrowableBook.BorrowItem(customer);

    Assert.AreEqual(initialAvailableCopies - 1, book.NumCopies);
    CollectionAssert.Contains(borrowableBook.Borrowers, customer);
}
```

Esemete tagastamist illustreerib järgnev test meetod.

```
[TestMethod]
public void ReturnItemTest()
{
    borrowableVideo.Borrowers.Add(customer);

    int initialAvailableCopies = video.NumCopies;
    borrowableVideo.ReturnItem(customer);

    Assert.AreEqual(initialAvailableCopies + 1, video.NumCopies);
    CollectionAssert.DoesNotContain(borrowableVideo.Borrowers, customer);
}
```

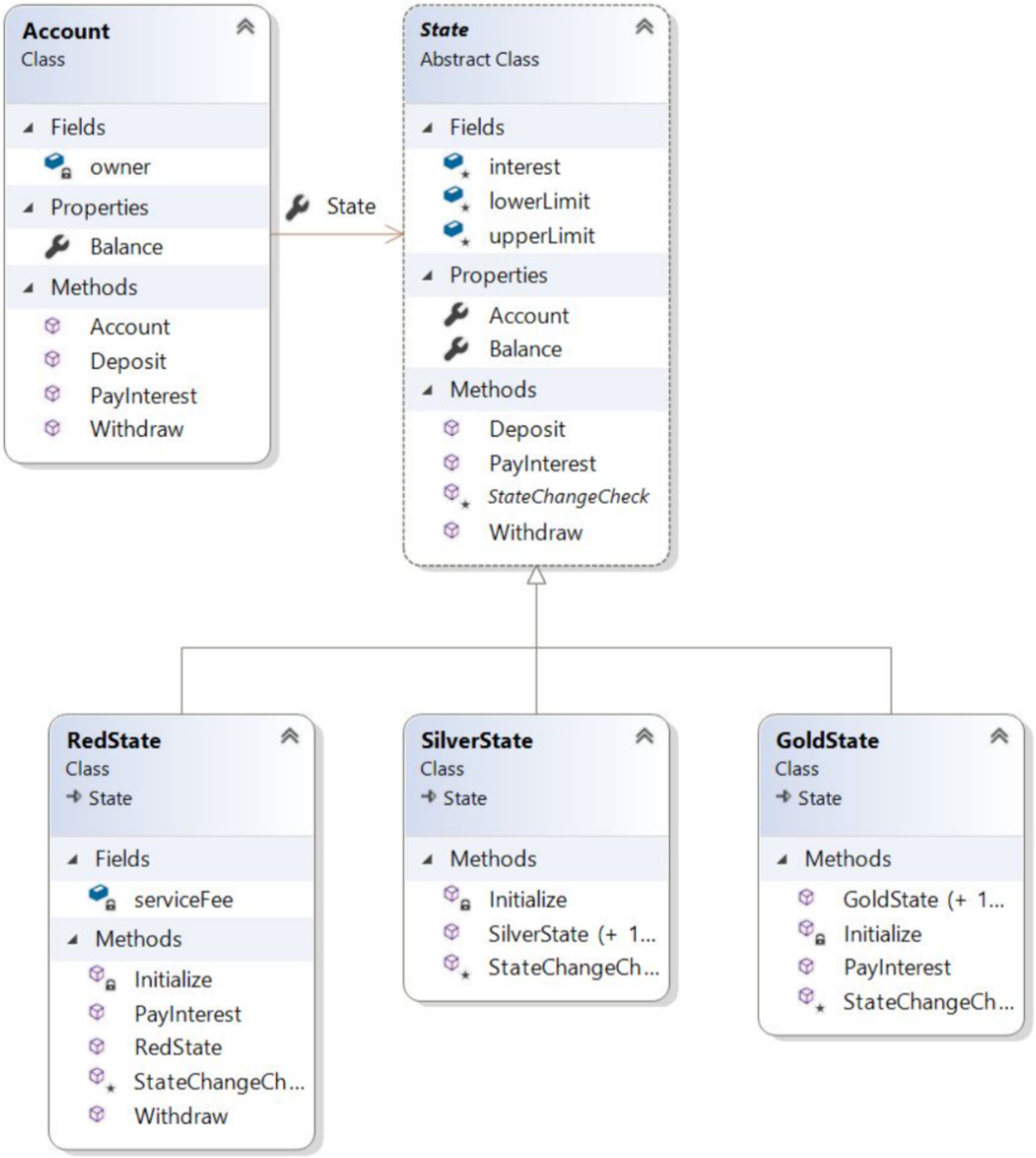
Selleks, et testida Decorator mustri tööd, on loodud abimeetod, mis kontrollib meetodi olemasolu klassis. LibraryItem klass (ilma Decoratorita) ei oma meetodit BorrowItem, kuid dekoreerides selle Borrowable'ga on meetod olemas. See on välja toodud järgnevas testis.

```
[TestMethod]
public void BorrowableItemHasCorrectMethodTest()
{
    Assert.IsFalse(HasMethod(video, "BorrowItem"));
    Assert.IsTrue(HasMethod(borrowableVideo, "BorrowItem"));
}

private bool HasMethod(LibraryItem objectToCheck, string methodName)
{
    var type = objectToCheck.GetType();
    return type.GetMethod(methodName) != null;
}
```

State juhend

State

Definitsioon: Võimaldab objektil muuta enda käitumist vastavat sisemise oleku muutustele. Objekt näib nagu oleks muutnud oma klassi. [GoF, lk 305]



Antud näites on kasutatud State disainimustrit, et võimaldada konto (Account) klassil muuta enda käitumist, sõltuvalt selle kontoseisust. Konto võib olla kolmes erinevas olekus - negatiivse kontojäägiga olek (RedState), tavaolek (SilverState), kuldliikme staatusega olek (GoldState). Iga kord, kui kontojääk muutub, toimub olekumuutuse kontroll (StateChangeCheck meetod) ning vastavalt uuele kontoseisule jääb olek samaks või asendub uue olekuga. Vaatame oleku (State) klassi.

```
public abstract class State {

    protected double interest;
    protected double lowerLimit;
    protected double upperLimit;
    public Account Account { get; set; }
    public double Balance { get; set; }

    public virtual void Deposit(double amount) {
        Balance += amount;
        StateChangeCheck();
    }

    public virtual void Withdraw(double amount) {
        Balance -= amount;
        StateChangeCheck();
    }

    public virtual void PayInterest() {
        Balance += interest * Balance;
        StateChangeCheck();
    }

    protected abstract void StateChangeCheck();
}
```

Tegemist on abstraktse klassiga, mida varem nimetatud konkreetsed klassid realiseerivad. Klassi muutujateks on interssimäär (interest) ja konto oleku alumine ja ülemine piir. Deposit ja Withdraw meetoditega toimub kontole raha lisamine ja sealt vähendamine. PayInterest meetodiga toimub intressi väljamakse. StateChangeCheck toimub pärast mistahes äsja nimetatud meetodi kutset ning sellega kontrollitakse, kas konto olek jääb samaks või muutub. SilverState klassi realiseeritud StateChangeCheck näeb välja selline:

```
protected override void StateChangeCheck() {
    if(Balance < lowerLimit) {
        Account.State = new RedState(this);
    } else if(Balance > upperLimit) {
        Account.State = new GoldState(this);
    }
}
```

Seega, kui pärast kontojäägi muutust on bilanss alla SilverState alampiiri (0.0), läheb konto olekusse RedState. Kui see aga jääb üle SilverState ülempiiri (1000.0), siis uueks olekuks saab GoldState. Alampiiri ja ülempiiri vahemikku jäädes olek jääb samaks.

```
[TestMethod]
public void ChangeStateFromSilverToGolden()
{
    Assert.AreEqual(account.State.GetType(), typeof(SilverState));

    account.Deposit(10000.0);

    Assert.AreEqual(account.State.GetType(), typeof(GoldState));
}

[TestMethod]
public void ChangeStateFromSilverToRed()
{
    Assert.AreEqual(account.State.GetType(), typeof(SilverState));

    account.Withdraw(500);

    Assert.AreEqual(account.State.GetType(), typeof(RedState));
}
```

Ülal välja toodud testid kontrollivad oleku muutust SilverState'st GoldState'i ja RedState'i. Testide käivitades saame rohelise tulemuse.

Lisa 4 – Kontrolltöö küsimused

Järgnevalt on välja toodud 15 kontrolltöö küsimust 60-st ehk kõik struktuurimustreid puudutavad küsimused. Ülejäänud küsimused on kättesaadavad GitHubi hoidlas, aadressil:

https://github.com/kaarelsoot/DesignPatterns/blob/master/Test_Questions.txt

Milline kirjeldus vastab **Adapter** disainimustrile?

Select one:

- a. Esitab objektid puu struktuurina ja näitab nende kuuluvussuhteid. Võimaldab kliendil üheselt kohelda individuaalseid objekte ja objektide kompositsioone.
- b. Määrab objektile dünaamiliselt lisa kohustusi. Pakub alamklasside loomise asemel funktsionaalsuse laiendamiseks paindlikku alternatiivi.
- c. Pakub alamsüsteemi liideste hulgale ühtset liidest. Defineerib kõrgema taseme liidese, mis teeb alamsüsteemi kasutamise lihtsamaks.
- d. Muundab klassi liidese kliendi poolt oodatavaks liideseks. Võimaldab klassidel koos töötada, mis muidu omavahel ei ühildu.

Millisele disainimustrile vastab järgnev kirjeldus?

Muundab klassi liidese kliendi poolt oodatavaks liideseks. Võimaldab klassidel koos töötada, mis muidu omavahel ei ühildu.

Select one:

- a. Decorator
- b. Proxy
- c. Adapter
- d. Facade

Milline kirjeldus vastab **Bridge** disainimustrile?

Select one:

- a. Määrab objektile dünaamiliselt lisa kohustusi. Pakub alamklasside loomise asemel funktsionaalsuse laiendamiseks paindlikku alternatiivi.
- b. Esitab objektid puu struktuurina ja näitab nende kuuluvussuhteid. Võimaldab kliendil üheselt kohelda individuaalseid objekte ja objektide kompositsioone.
- c. Võimaldab suure hulga detailsete objektide efektiivset jagamist.
- d. Lahutab abstraktsiooni selle implementatasioonist, nii et need saavad teineteisest sõltumatult muutuda.

Millist disainimustrit iseloomustab järgnev kirjeldus?

Lahutab abstraktsiooni selle implementatasioonist, nii et need saavad teineteisest sõltumatult muutuda.

Select one:

- a. Facade
- b. Flyweight
- c. Bridge
- d. Mediator

Milline kirjeldus vastab **Composite** mustrile?

Select one:

- a. Määrab objektile dünaamiliselt lisa kohustusi. Pakub alamklasside loomise asemel funktsionaalsuse laiendamiseks paindlikku alternatiivi.
- b. Esitab objektid puu struktuurina ja näitab nende kuuluvussuhteid. Võimaldab kliendil üheselt kohelda individuaalseid objekte ja objektide kompositsioone.
- c. Pakub alamsüsteemi liideste hulgale ühtset liidest. Defineerib kõrgema taseme liidese, mis teeb alamsüsteemi kasutamise lihtsamaks.
- d. Pakub asendust või kohahoidjat teisele objektile, eesmärgiga kontrollida selle objekti juurdepääsu.

Millisele disainimustrile vastab järgnev kirjeldus?

Esitab objektid puu struktuurina ja näitab nende kuuluvussuhteid. Võimaldab kliendil üheselt kohelda individuaalseid objekte ja objektide kompositsioone.

Select one:

- a. Command
- b. Composite
- c. Memento
- d. Abstract Factory

Milline kirjeldus vastab **Decorator** mustrile?

Select one:

- a. Võimaldab suure hulga detailsete objektide efektiivset jagamist.
- b. Võimaldab agregaat-objekti elementide poole järjestikku pöördumise, ilma selle aluseks olevat esitust avalikustamata.
- c. Määrab objektile dünaamiliselt lisa kohustusi. Pakub alamklasside loomise asemel funktsionaalsuse laiendamiseks paindlikku alternatiivi.
- d. Defineerib üks-mitmele sõltuvuse objektide vahel, nii et kui ühe objekti olek muutub, teavitatakse ja uuendatakse kõiki selle objekti jälgijaid.

Millist disainimustrit iseloomustab järgnev kirjeldus?

Määrab objektile dünaamiliselt lisa kohustusi. Pakub alamklasside loomise asemel funktsionaalsuse laiendamiseks paindlikku alternatiivi.

Select one:

- a. Iterator
- b. Proxy
- c. Singleton
- d. Decorator

Milline kirjeldus iseloomustab **Flyweight** disainimustrit?

Select one:

- a. Muundab klassi liidese kliendi poolt oodatavaks liideseks. Võimaldab klassidel koos töötada, mis muidu omavahel ei ühildu.
- b. Kasutades kapseldamist, salvestab ja väljastab objekti sisemise oleku, nii et objekti oleks hiljem võimalik sellesse olekusse tagasi tuua.
- c. Võimaldab suure hulga detailsete objektide efektiivset jagamist.
- d. Hoiab ära päringu saatja sidumist selle saajaga, andes rohkem kui ühele objektile võimaluse päringut käidelda. Aheldab saaja rollis olevad objektid ja annab päringu mööda ahelat edasi, kuniks leidub objekt, kes päringu lahendab.

Milline disainimuster vastab järgnevale kirjeldusele?

Võimaldab suure hulga detailsete objektide efektiivset jagamist.

Select one:

- a. Flyweight
- b. Prototype
- c. Singleton
- d. State

Milline disainimuster pakub alamsüsteemi liidestele (interface) ühtset liidest? See defineerib kõrgema taseme liidese, mis teeb alamsüsteemi kasutamise lihtsamaks.

Select one:

- a. Command
- b. Adapter
- c. Strategy
- d. Facade

Milline väide iseloomustab **Facade** disainimustrit?

Select one:

- a. Seda mustrit kasutatakse juhul, kui on tarvis objektide gruppi käsitleda, kui üksikut objekti.
- b. See muster pakub alamsüsteemi liideste hulgale ühtset liidest ja defineerib kõrgema taseme liidese, mis teeb alamsüsteemi kasutamise lihtsamaks.
- c. See muster muundab klassi liidest selliselt, et see sobituks kliendi poolt defineeritud liidesega.
- d. See võimaldab kasutajal lisada uut funktsionaalsust olemasolevale objektile.

Milline väide iseloomustab loomismustreid? (Structural patterns)

Select one:

- a. Kõik väited sobivad.
- b. Need disainimustrid annavad viisi objektide loomiseks ilma new operaatorit kasutamata, peites sellega objektide loomise loogika.
- c. Need disainimustrid vastutavad tegevuste efektiivse jaotuse eest programmi eri osade vahel.
- d. Need disainimustrid vastutavad efektiivsete klassihierarhiate ja klassidevaheliste suhete loomise eest.

Milline kirjeldus vastab **Proxy** disainimustrile?

Select one:

- a. Pakub asendust või kohahoidjat teisele objektile, eesmärgiga kontrollida selle objekti juurdepääsu.
- b. Võimaldab objektil muuta enda käitumist vastavat sisemise oleku muutustele. Objekt näib nagu oleks muutnud oma klassi.
- c. Määrab objektile dünaamiliselt lisa kohustusi. Pakub alamklasside loomise asemel funktsionaalsuse laiendamiseks paindlikku alternatiivi.
- d. Muundab klassi liidese kliendi poolt oodatavaks liideseks. Võimaldab klassidel koos töötada, mis muidu omavahel ei ühildu.

Milline disainimuster vastab järgnevale kirjeldusele?

Pakub asendust või kohahoidjat teisele objektile, eesmärgiga kontrollida selle objekti juurdepääsu.

Select one:

- a. Proxy
- b. Factory Method
- c. Abstract Factory
- d. Template Method

Lisa 5 – DeliveryApp rakendus

Järgnevalt on välja toodud autori õppeeesmärgil loodud rakenduse DeliveryApp lähtekood, testid ja üks peatükk sellele vastavast juhendist. Ülejäänud juhend on kättesaadav GitHubi hoidlas, aadressil:

https://github.com/kaarelsoot/DesignPatterns/blob/master/DeliveryApp/DeliveryApp.pdf

Lähtekood

```
using System;
using System.Collections.Generic;

namespace DeliveryApplication
{
    class DeliveryApp
    {
        public static void Main(string[] args)
        {
            AbstractDeliveryVehicleFactory factory = new CountingDeliveryVehicleFactory();

            var deliveryApp = new DeliveryApp();
            deliveryApp.PlanDelivery(factory);
        }

        private void PlanDelivery(AbstractDeliveryVehicleFactory factory)
        {
            IDeliveryVehicle deliveryBike = factory.CreateDeliveryBike();
            IDeliveryVehicle deliveryCar = factory.CreateDeliveryCar();
            IDeliveryVehicle deliveryVan = factory.CreateDeliveryVan();
            IDeliveryVehicle deliveryTruck = factory.CreateDeliveryTruck();
            IDeliveryVehicle locker = factory.CreateParcelLocker();

            Fleet fleetOfVehicles = new Fleet();

            fleetOfVehicles.Add(deliveryBike);
            fleetOfVehicles.Add(deliveryCar);
            fleetOfVehicles.Add(deliveryVan);
            fleetOfVehicles.Add(deliveryTruck);
            fleetOfVehicles.Add(locker);

            Console.WriteLine("Deliveries:");

            MakeDelivery(fleetOfVehicles);

            var fleetOfTrucks = new Fleet();

            IDeliveryVehicle truckOne = factory.CreateDeliveryTruck();
            IDeliveryVehicle truckTwo = factory.CreateDeliveryTruck();
            IDeliveryVehicle truckThree = factory.CreateDeliveryTruck();

            fleetOfTrucks.Add(truckOne);
            fleetOfTrucks.Add(truckTwo);
            fleetOfTrucks.Add(truckThree);

            Console.WriteLine("\nDeliveries made by the new fleet of trucks:");
            MakeDelivery(fleetOfTrucks);
```

```
            Console.WriteLine("\nTotal parcels delivered (including pick-ups): " +
DeliveryCounter.NumberOfDeliveries);
        }

        private void MakeDelivery(IDeliveryVehicle vehicle)
        {
            vehicle.Deliver();
        }
    }

    #region Deliver Vehicles

    public interface IDeliveryVehicle
    {
        void Deliver();
    }

    public class DeliveryBike : IDeliveryVehicle
    {
        public void Deliver()
        {
            Console.WriteLine("Bike makes a delivery");
        }
    }

    public class DeliveryCar : IDeliveryVehicle
    {
        public void Deliver()
        {
            Console.WriteLine("Car makes a delivery");
        }
    }

    public class DeliveryVan : IDeliveryVehicle
    {
        public void Deliver()
        {
            Console.WriteLine("Van makes a delivery");
        }
    }

    public class DeliveryTruck : IDeliveryVehicle
    {
        public void Deliver()
        {
            Console.WriteLine("Truck makes a delivery");
        }
    }

    #endregion

    #region Parcel Locker

    public class ParcelLocker
    {
        public void ProvidePickUp()
        {
            Console.WriteLine("Package is picked up from a parcel locker");
        }
    }

    public class ParcelLockerAdapter : IDeliveryVehicle
    {
        private ParcelLocker _parcelLocker;

        public ParcelLockerAdapter(ParcelLocker parcelLocker)
        {
            _parcelLocker = parcelLocker;
        }

        public void Deliver()
        {
            _parcelLocker.ProvidePickUp();
        }
    }

    #endregion

    #region Delivery Counter

    public class DeliveryCounter : IDeliveryVehicle
    {
        private IDeliveryVehicle _vehicle;
```

```
        public static int NumberOfDeliveries { get; private set; }

        public DeliveryCounter(IDeliveryVehicle vehicle)
        {
            _vehicle = vehicle;
        }

        public void Deliver()
        {
            _vehicle.Deliver();
            NumberOfDeliveries++;
        }
    }

    #endregion

    #region Factory

    public abstract class AbstractDeliveryVehicleFactory
    {
        public abstract IDeliveryVehicle CreateDeliveryBike();
        public abstract IDeliveryVehicle CreateDeliveryCar();
        public abstract IDeliveryVehicle CreateDeliveryVan();
        public abstract IDeliveryVehicle CreateDeliveryTruck();
        public abstract IDeliveryVehicle CreateParcelLocker();
    }

    public class CountingDeliveryVehicleFactory : AbstractDeliveryVehicleFactory
    {
        public override IDeliveryVehicle CreateDeliveryBike()
        {
            return new DeliveryCounter(new DeliveryBike());
        }

        public override IDeliveryVehicle CreateDeliveryCar()
        {
            return new DeliveryCounter(new DeliveryCar());
        }

        public override IDeliveryVehicle CreateDeliveryVan()
        {
            return new DeliveryCounter(new DeliveryVan());
        }

        public override IDeliveryVehicle CreateDeliveryTruck()
        {
            return new DeliveryCounter(new DeliveryTruck());
        }

        public override IDeliveryVehicle CreateParcelLocker()
        {
            return new DeliveryCounter(new ParcelLockerAdapter(new ParcelLocker()));
        }
    }

    #endregion

    #region Delivery Fleet

    public class Fleet : IDeliveryVehicle
    {
        List<IDeliveryVehicle> _vehicles = new List<IDeliveryVehicle>();

        public void Add(IDeliveryVehicle vehicle)
        {
            _vehicles.Add(vehicle);
        }

        public void Deliver()
        {
            foreach (IDeliveryVehicle vehicle in _vehicles)
            {
                vehicle.Deliver();
            }
        }
    }

    #endregion
}
```

Testid

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using DeliveryApplication;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace DeliveryApplicationTests
{
    [TestClass]
    public class DeliveryAppTests
    {
        #region Test setup

        AbstractDeliveryVehicleFactory factory = new CountingDeliveryVehicleFactory();

        private void MakeDelivery(IDeliveryVehicle vehicle)
        {
            vehicle.Deliver();
        }

        [TestCleanup]
        public void TestCleanup()
        {
            DeliveryCounter.NumberOfDeliveries = 0; // Resets delivery count after each test
        }

        #endregion

        #region Test Template

        [Ignore]
        public void TestTemplate()
        {
            StringWriter writer = beginReading(); // begin console capture

            // Setup test here

            List<String> consoleEntries = endReading(writer); // end console capture

            // Assert here
        }

        private StringWriter beginReading()
        {
            var writer = new StringWriter();
            Console.SetOut(writer);
            return writer;
        }

        private List<String> endReading(StringWriter writer)
        {
            writer.Flush();
            var myString = writer.GetStringBuilder().ToString();
            return myString.Split(Environment.NewLine, StringSplitOptions.RemoveEmptyEntries).ToList();
        }

        #endregion

        #region Parcel locker tests

        [TestMethod]
        public void ParcelLockerTest()
        {
            IDeliveryVehicle parcelLocker = new ParcelLockerAdapter(new ParcelLocker());
            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(parcelLocker);
            List<String> consoleEntries = endReading(writer); // end console capture

            List<String> expected = new List<String>(new []{ "Package is picked up from a parcel locker" });
            CollectionAssert.AreEqual(consoleEntries, expected, "Deliveries don't match expected deliveries");
        }

        [TestMethod]
        public void ParcelLockerTest_RandomVehicles()
        {
            Random random = new Random();
```

```
            var expectedList = new String[30];
            StringWriter writer = beginReading(); // begin console capture
            for (int i = 0; i < 30; i++)
            {
                int caseSwitch = random.Next(1,5);
                switch (caseSwitch)
                {
                    case 1:
                        MakeDelivery(new DeliveryBike());
                        expectedList[i] = "Bike makes a delivery";
                        break;
                    case 2:
                        MakeDelivery(new DeliveryCar());
                        expectedList[i] = "Car makes a delivery";
                        break;
                    case 3:
                        MakeDelivery(new DeliveryVan());
                        expectedList[i] = "Van makes a delivery";
                        break;
                    case 4:
                        MakeDelivery(new DeliveryTruck());
                        expectedList[i] = "Truck makes a delivery";
                        break;
                    case 5:
                        MakeDelivery(new ParcelLockerAdapter(new ParcelLocker()));
                        expectedList[i] = "Package is picked up from a parcel locker";
                        break;
                }
            }
            List<String> consoleEntries = endReading(writer); // end console capture

            CollectionAssert.AreEqual(consoleEntries, expectedList, "Deliveries don't match expected
deliveries");
        }

        #endregion

        #region DeliveryCounter tests

        [TestMethod]
        public void DeliveryCounterTest()
        {
            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(new DeliveryCounter(new DeliveryBike()));
            MakeDelivery(new DeliveryCounter(new DeliveryCar()));
            MakeDelivery(new DeliveryCounter(new DeliveryVan()));
            MakeDelivery(new DeliveryCounter(new DeliveryTruck()));
            List<String> consoleEntries = endReading(writer); // end console capture

            Assert.AreEqual(consoleEntries.Count, DeliveryCounter.NumberOfDeliveries, "Delivery count does not
match expected deliveries");
        }

        [TestMethod]
        public void DeliveryCounterTest_RandomNumberOfCars()
        {
            Random random = new Random();
            int randomNumber = random.Next(10, 100);

            StringWriter writer = beginReading(); // begin console capture
            for (int i = 0; i < randomNumber; i++)
            {
                MakeDelivery(new DeliveryCounter(new DeliveryCar()));
            }
            List<String> consoleEntries = endReading(writer); // end console capture

            Assert.AreEqual(consoleEntries.Count, DeliveryCounter.NumberOfDeliveries, "Delivery count does not
match expected deliveries");
        }

        [TestMethod]
        public void DeliveryCounterTest_NoDeliveries()
        {
            Assert.AreEqual(DeliveryCounter.NumberOfDeliveries, 0, "Delivery count does not match expected
deliveries");
        }

        #endregion

        #region Factory tests

        [TestMethod]
        public void FactoryTest()
```

```
        {
            IDeliveryVehicle deliveryBike = factory.CreateDeliveryBike();
            IDeliveryVehicle deliveryCar = factory.CreateDeliveryCar();
            IDeliveryVehicle deliveryVan = factory.CreateDeliveryVan();
            IDeliveryVehicle deliveryTruck = factory.CreateDeliveryTruck();

            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(deliveryBike);
            MakeDelivery(deliveryCar);
            MakeDelivery(deliveryVan);
            MakeDelivery(deliveryTruck);
            List<String> consoleEntries = endReading(writer); // end console capture

            var expectedList = new List<string>(new []
            {
                "Bike makes a delivery",
                "Car makes a delivery",
                "Van makes a delivery",
                "Truck makes a delivery"
            });

            Assert.AreEqual(consoleEntries.Count, 4, "Delivery count does not match expected deliveries");
            CollectionAssert.AreEqual(consoleEntries, expectedList, "Deliveries don't match expected
deliveries");
        }

        [TestMethod]
        public void FactoryTest_RandomVehicles()
        {
            Random random = new Random();
            var expectedList = new String[30];
            StringWriter writer = beginReading(); // begin console capture
            for (int i = 0; i < 30; i++)
            {
                int caseSwitch = random.Next(1,5);
                switch (caseSwitch)
                {
                    case 1:
                        MakeDelivery(factory.CreateDeliveryBike());
                        expectedList[i] = "Bike makes a delivery";
                        break;
                    case 2:
                        MakeDelivery(factory.CreateDeliveryCar());
                        expectedList[i] = "Car makes a delivery";
                        break;
                    case 3:
                        MakeDelivery(factory.CreateDeliveryVan());
                        expectedList[i] = "Van makes a delivery";
                        break;
                    case 4:
                        MakeDelivery(factory.CreateDeliveryTruck());
                        expectedList[i] = "Truck makes a delivery";
                        break;
                }
            }
            List<String> consoleEntries = endReading(writer); // end console capture

            CollectionAssert.AreEqual(consoleEntries, expectedList, "Deliveries don't match expected
deliveries");
        }

        #endregion

        #region Fleet tests

        [TestMethod]
        public void FleetTest()
        {
            IDeliveryVehicle deliveryBike = new DeliveryBike();
            IDeliveryVehicle deliveryCar = new DeliveryCar();
            IDeliveryVehicle deliveryVan = new DeliveryVan();
            IDeliveryVehicle deliveryTruck = new DeliveryTruck();

            Fleet fleetOfVehicles = new Fleet();

            fleetOfVehicles.Add(deliveryBike);
            fleetOfVehicles.Add(deliveryCar);
            fleetOfVehicles.Add(deliveryVan);
            fleetOfVehicles.Add(deliveryTruck);

            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(fleetOfVehicles);
            List<String> consoleEntries = endReading(writer); // end console capture
```

```
            var expectedList = new List<string>(new []
            {
                "Bike makes a delivery",
                "Car makes a delivery",
                "Van makes a delivery",
                "Truck makes a delivery"
            });

            Assert.AreEqual(consoleEntries.Count, 4, "Delivery count does not match expected deliveries");
            CollectionAssert.AreEqual(consoleEntries, expectedList, "Deliveries don't match expected
deliveries");
        }

        [TestMethod]
        public void FleetTest_RandomNumberOfTrucks()
        {
            // Add random amount of trucks to a fleet
            Random random = new Random();
            int randomNumber = random.Next(10, 100);
            Fleet fleetOfTrucks = new Fleet();
            for (int i = 0; i < randomNumber; i++)
            {
                fleetOfTrucks.Add(new DeliveryTruck());
            }
            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(fleetOfTrucks);
            List<String> consoleEntries = endReading(writer); // end console capture

            // Create String to compare Console statements with
            var expectedList = new String[randomNumber];
            for(int i = 0 ; i < randomNumber ; i++) expectedList[i] = "Truck makes a delivery";

            Assert.AreEqual(consoleEntries.Count, randomNumber, "Delivery count does not match expected
deliveries");
            CollectionAssert.AreEqual(consoleEntries, expectedList, "Deliveries don't match expected
deliveries");
        }

        [TestMethod]
        public void FleetTest_RandomVehicles()
        {
            // Adds 30 random vehicles to a fleet and makes a delivery
            Random random = new Random();
            Fleet fleetOfRandomVehicles = new Fleet();
            var expectedList = new String[30];
            for (int i = 0; i < 30; i++)
            {
                int caseSwitch = random.Next(1,5);
                switch (caseSwitch)
                {
                    case 1:
                        fleetOfRandomVehicles.Add(new DeliveryBike());
                        expectedList[i] = "Bike makes a delivery";
                        break;
                    case 2:
                        fleetOfRandomVehicles.Add(new DeliveryCar());
                        expectedList[i] = "Car makes a delivery";
                        break;
                    case 3:
                        fleetOfRandomVehicles.Add(new DeliveryVan());
                        expectedList[i] = "Van makes a delivery";
                        break;
                    case 4:
                        fleetOfRandomVehicles.Add(new DeliveryTruck());
                        expectedList[i] = "Truck makes a delivery";
                        break;
                }
            }
            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(fleetOfRandomVehicles);
            List<String> consoleEntries = endReading(writer); // end console capture

            CollectionAssert.AreEqual(consoleEntries, expectedList, "Deliveries don't match expected
deliveries");
        }

        [TestMethod]
        public void FleetTest_Empty()
        {
            Fleet fleet = new Fleet();
            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(fleet);
```

```
            List<String> consoleEntries = endReading(writer); // end console capture

            CollectionAssert.AreEqual(consoleEntries, new List<string>(), "No deliveries should be made");
        }

        #endregion

        #region Combined tests

        [TestMethod]
        public void CombinedTest()
        {
            IDeliveryVehicle deliveryBike = factory.CreateDeliveryBike();
            IDeliveryVehicle deliveryCar = factory.CreateDeliveryCar();
            IDeliveryVehicle deliveryVan = factory.CreateDeliveryVan();
            IDeliveryVehicle deliveryTruck = factory.CreateDeliveryTruck();
            IDeliveryVehicle locker = factory.CreateParcelLocker();

            Fleet fleetOfVehicles = new Fleet();

            fleetOfVehicles.Add(deliveryBike);
            fleetOfVehicles.Add(deliveryCar);
            fleetOfVehicles.Add(deliveryVan);
            fleetOfVehicles.Add(deliveryTruck);
            fleetOfVehicles.Add(locker);

            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(fleetOfVehicles);
            List<String> consoleEntries = endReading(writer); // end console capture

            var expectedList = new List<string>(new []
            {
                "Bike makes a delivery",
                "Car makes a delivery",
                "Van makes a delivery",
                "Truck makes a delivery",
                "Package is picked up from a parcel locker"
            });

            Assert.AreEqual(consoleEntries.Count, 5, "Delivery count does not match expected deliveries");
            CollectionAssert.AreEqual(consoleEntries, expectedList, "Deliveries don't match expected
deliveries");
        }

        [TestMethod]
        public void CombinedTest_RandomVehicles()
        {
            // Add random amount of vehicles to a fleet and makes a delivery
            Random random = new Random();
            int amountOfVehicles = random.Next(10, 100);
            Fleet fleet = new Fleet();
            var expectedList = new String[amountOfVehicles];
            for (int i = 0; i < amountOfVehicles; i++)
            {
                int caseSwitch = random.Next(1,6);
                switch (caseSwitch)
                {
                    case 1:
                        fleet.Add(factory.CreateDeliveryBike());
                        expectedList[i] = "Bike makes a delivery";
                        break;
                    case 2:
                        fleet.Add(factory.CreateDeliveryCar());
                        expectedList[i] = "Car makes a delivery";
                        break;
                    case 3:
                        fleet.Add(factory.CreateDeliveryVan());
                        expectedList[i] = "Van makes a delivery";
                        break;
                    case 4:
                        fleet.Add(factory.CreateDeliveryTruck());
                        expectedList[i] = "Truck makes a delivery";
                        break;
                    case 5:
                        fleet.Add(factory.CreateParcelLocker());
                        expectedList[i] = "Package is picked up from a parcel locker";
                        break;
                }
            }
            StringWriter writer = beginReading(); // begin console capture
            MakeDelivery(fleet);
            List<String> consoleEntries = endReading(writer); // end console capture
```

```
            Assert.AreEqual(consoleEntries.Count, amountOfVehicles, "Delivery count does not match expected
deliveries");
            Assert.AreEqual(consoleEntries.Count, DeliveryCounter.NumberOfDeliveries, "Delivery counter's
result differs from actual number of deliveries");
            CollectionAssert.AreEqual(consoleEntries, expectedList, "Deliveries don't match expected
deliveries");
        }

        #endregion
    }
}
```

Adapter disainimustri juhend

1. Adapter disainimuster

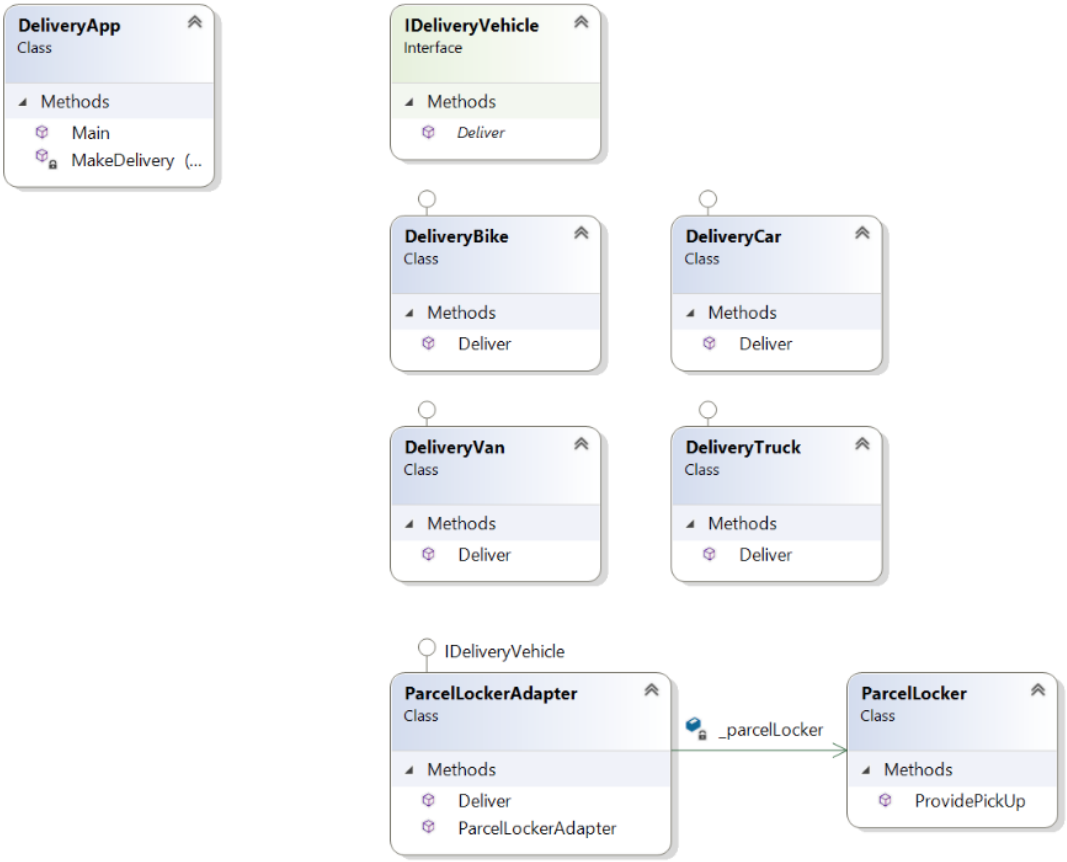
Ettevõtte tahab kasutusele võtta pakiautomaadid. Erinevalt sõidukitest puudub

```
    {
        IDeliveryVehicle deliveryBike = new DeliveryBike();
        IDeliveryVehicle deliveryCar = new DeliveryCar();
        IDeliveryVehicle deliveryVan = new DeliveryVan();
        IDeliveryVehicle deliveryTruck = new DeliveryTruck();
        IDeliveryVehicle locker = new ParcelLockerAdapter(new
ParcelLocker());

        Console.WriteLine("Deliveries:");

        MakeDelivery(deliveryBike);
        MakeDelivery(deliveryCar);
        MakeDelivery(deliveryVan);
        MakeDelivery(deliveryTruck);
        MakeDelivery(locker);
    }

    private void MakeDelivery(IDeliveryVehicle vehicle)
    {
        vehicle.Deliver();
    }
}
```

Kõigepealt loome pakiautomaadi ja pakendame (ingl.k. *wrap*) selle ParcelLockerAdapterisse, mille tulemusena käitub see justkui sõiduk. Nüüd, kui pakiautomaat on pakendatud, saab seda kasutada nagu igat teist tarnet teostavat sõidukit. Käivitades rakenduse, on tulemus järgmine:

```
Deliveries:
Bike makes a delivery
Car makes a delivery
Van makes a delivery
Truck makes a delivery
Package is picked up from a parcel locker
```

Rakenduse klassidiagrammile lisandusid ParcelLockerAdapter ja ParcelLocker klassid.



Võrdleme seda Adapter mustri UMLiga.