Tallinna Ülikool

Informaatika instituut

Objektorienteeritud veebilahenduse koostamine PHP abil

Bakalaureusetöö

Autor: Mart-Indrek Süld

Juhendaja: Jaagup Kippar

Autor: …… „ …… „ 2012

Juhendaja: …… „ …… „ 2012

Instituudi direktor: …… „ …… „ 2012

Tallinn 2012

Autori deklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

……. …….

(kuupäev) (autor)

Sisukord

Sissejuhatus

Tänapäeva kiire elutempo juures puutume infotehnoloogiaga igapäevaselt kokku. See on möödapääsmatu. Suund, mille me oleme valinud, on tekitanud olukorra, kus edukaks eksisteerimiseks on vajalik tunda interneti poolt pakutavaid võimalusi. See on osalt tingitud sellepärast, et it-sektor on viimastel aastatel teinud hüppelise kasvu ja muutunud asendamatuks, mille olemasoluta enam igapäeva toimetusi ette ei kujutata. Tegu on valdkonnaga, mis areneb justkui kiirteel ja sama plahvatuslikult, nagu on uuenenud meid igapäevaselt ümbritsevad tooted, on tõusnud ka inimeste ja ettevõtete huvi interneti ja sellega kaasneva vastu.

Oma isikliku ettevõtte võib virtuaalselt registreerida vähem kui viie minutiga. Vaja on ainult ideed ja pealehakkamist. Puudub ruumide rentimise või personalile palga maksmise vajadus. Selle väite kinnitamiseks leidub hulgaliselt edukaid ettevõtteid, mille tegevusvaldkond on ainult veebipõhine – kindlale kategooriale spetsialiseerunud veebikaubamaja, inimesi ühendav sotsiaalmeedia võrgustik või igapäevaseid uudiseid vahendav portaal. See tähendab seda, et internet on saanud ettevõtete lahutamatuks osaks, mida ei tohi ignoreerida. Elatakse informatsiooniajastul ja teave peab olema kõigile kättesaadav ja kergesti hallatav. Praegusel ajal on infotehnoloogia populaarseim kui kunagi varem ja inimeste jaoks, kes seda kõige paremini mõistavad, ka väga hästi tasustatud.

Käesolevat bakalaureusetööd ajendas kirjutama autori huvi ja igapäevane kokkupuude veebi põhise programmeerimisega. Veebiarendajana on ta teadlik, et mooduseid erinevate ideede realiseerimiseks on mitmeid, kuid arusaadava ja kergesti hallatava töö garanteerib ainult korrektsete programmeerimisvõtete praktiseerimine.

Bakalaureusetöö eesmärgiks on anda lugejale ülevaade PHP objektorienteeritusest ja sellega kaasnevatest võimalustest. Ühtlasi otsitakse töö raames vastust küsimusele, kas samas valdkonnas on eelnevalt juhendeid koostatud. Püstitatud eesmärkide saavutamiseks tutvub autor erinevate artiklite ja näidetega

Bakalaureusetöö on jaotatud neljaks peatükiks. Esimene peatükk annab ülevaate skriptimiskeelest PHP ja tutvustab lugejale objektorienteeritud programmeerimise põhimõtteid. Teises peatükis analüüsitakse olemasolevaid inglise- ja eestikeelseid

õppematerjale. Kolmandas peatükis kirjeldatakse käesoleva õppematerjali läbimiseks soovituslikke õpiteid. Neljas peatükk annab ülevaate õppematerjali testimisest.

Töös on näidetena kasutatud autori enda programmeerimisoskusi ja internetis levivaid õpetusi, mille autor on valinud enda äranägemise järgi. Arusaadavuse huvides on töö lisana CD-plaadil saadaval kommenteeritud näidisrakendused.

Koostatud objektorienteeritud veebiprogrammeerimise õppematerjal on esitatud käesoleva bakalaureusetöö lisana.

1. Skriptimiskeel PHP

PHP ehk PHP: hüperteksti preprotsessor on 1994 aastal Rasmus Leodorfi poolt loodud skriptimiskeel, mida kasutatakse peamiselt serveripoolsetes lahendustes dünaamiliste veebilehtede loomiseks. Keel toetab andmebaasidega suhtlemist, objektorienteeritud programmeerimist ja on võimeline täitma üldiste lüüsiliideste programmide ülesandeid (What is PHP, 2012).

Keel oli algselt mõeldud R. Leodorfi kodulehe hoolduse lihtsustamiseks ja pidi välja vahetama kasutusel olevad Perli skriptid. Kombineerides olemasolevat koodi enda loodud Vormi Interpreteerijaga, millest kujunes hiljem PHP välja, suudeti luua ühendusi andmebaasidega, mis võimaldas genereerida lihtsa dünaamilise sisuga veebilehekülgi. 1995 aastal avalikustati lähtekood kiirendamaks vigade leidmist ja koodi täiustamist. Versiooni nimeks sai PHP 2 ja omas juba põhilist funktsionaalsust, mis tänapäevalgi kasutusel on (Patel, 2009).

PHP on serveripoolne sktriptimiskeel, mis võimaldab lihtsalt siduda koodi HTML-ga. Tänu sellele ja avatud lähtekoodile, on temast saanud väga populaarne vahend muutuva sisuga veebilehekülgede loomiseks. Kasutajal on võimalik suhelda andmebaasidega, luua pilte, kirjutada ja lugeda faile või suhelda kaug-serveritega. Andmete töötlemine toimub serveri siseselt ja tulemus väljastatakse veebilehena. See tähendab seda, et HTML-i genereerinud kood on kõrvalistele isikutele ligipääsmatu ja nähtav ainult serverisse ligipääsu omavatele kasutajatele. Nii PHP-d jooksutav server kui ka kood on kasutatavad peamistel operatsioonisüsteemide, mis teeb skriptimiskeele äärmiselt paindlikuks. Lisaks on PHP-sse eelnevalt programmeeritud mitmeid iseärasusi, mis hõlbustavad veebilehekülje arendamist. Võrreldes teiste programmeerimiskeeltega, on koodi töötlemine väga kiire, sest erinevalt teistest platvormidest, nõuab veebilehe töötlemine ja laadimine väga vähe aega (Patel, 2009).

1.1. Objektorienteeritud programmeerimine

Objektorienteeritud programmeerimine on meetod, mis läheneb rakenduse kirjutamisele märksa laiemalt, kui ainult käskude rittapanemises. Põhimõte seisneb selles, et programmis esinevaid andmeid ja nendega teostatavaid tegevusi püütakse esitada sarnaselt meie

igapäevases elus ettetulevate intelligentsete objektidena. Kõike, mille kohta saab nimisõna kasutada, proovitakse tituleerida objektiks. Objektil on tunnused ja omadused, mida nimetatakse atribuutideks. Tegevused, mida nimetatakse meetoditeks ja objektitüüp, mida nimetatakse klassiks. Näiteks programmeerides tüüpilist rollimängu, on mängija vaadeldav eraldi andmeobjekti ehk mängija klassina. Tal on nimi (atribuut), esemete kandmise võimekus (omadus) ja erinevad tegevused tõrjumaks vastase rünnakut (meetodid) (Jentson, 1999).

Objektorienteeritud maailmas on tavaks rääkida, et objektid suhtlevad omavahel, teavitades üksteist erinevatest sündmustest. Võttes aluseks rollimängu näite võime mängija klassile vastaseks tekitada vaenlase klassi. Kui vaenlane on alustanud rünnakut mängija vastu, teavitatakse mängija objekti sellest, kes seepeale enda vastu suunatud kallaletungi tõrjuma asub.

1.2. Objektorienteeritud programmeerimine PHP's

PHP on skriptimiskeel või tuntud ka kui protseduuriline programmeerimiskeel. Ta ei vasta traditsioonilisele objektorienteeritud programmeerimiskeele mustrile, vaid pigem omab sellega sarnaseid iseärasusi. Esmane ja limiteeritud võimalustega objektorienteerituse tugi saabus neljanda versiooniga ja täielik toetus tuli alles viienda versiooniga.

Enne veebilahenduse koostama hakkamist tuleks endalt küsida, kas objektorienteeritud programmeerimise rakendamine on tingimata vajalik või saab ka muud moodi? Kui üksikute skriptide töötlemine ja väheste protseduuride läbimine on kõik, mida PHP tegema peab, siis mitte. Aga kui soovitakse midagi enamat ja töötlemisülesannete hulka kuuluvad keerukad ülesanded, siis tuleks kindlasti praktiseerida objektorienteeritud programmeerimist. Enne objektide põhise lähenemise rakendamist, tuleks järgida reeglit:

- Lihtsa ülesehitusega lehele objektorienteeritus lisab tarbetut keerukust.
- Keeruka ülesehitusega lehele objektorienteeritus lisab vajalikku lihtsust.

2. Õppematerjali võrdlev analüüs ja kirjeldus

Käesoleva õppematerjali peamine eesmärk on tutvustada lugejale objektorienteeritud programmeerimise võtteid PHP-s. Tutvustamine toimub peamiselt praktiliste näidete näol, mis peaksid juhendist huvitatule abiks olema ja soosima tulevikus iseseisvalt uute intelligentsete objektide loomist dünaamilise veebilahenduse tarbeks. Juhendis sisalduvad koodinäited on eraldiseisvad ja ei ole üksteisest sõltuvad, kuna sellisel juhul ei muutu näited liialt spetsiifiliseks ja kindlustatakse juhendi mitmekülgsus.

Õppematerjal ei ole mõeldud algajale lugejale veebiprogrammeerimise tutvustamiseks, vaid isikule, kellel on olemas varasem kokkupuude skriptimiskeele PHP-ga, tunneb selle süntaksit ja teab toimimise põhimõtteid, kuid puudub ülevaade objektorienteeritusest ja kuidas seda dünaamilise veebilehe loomisel täpsemalt rakendada. Õppematerjal alustab lihtsa ülesehitusega objektorienteeritud objektide loomisega ja liigub järk-järgult keerukamate objektide loomise poole.

Juhendi läbimiseks on vajalik PHP ja MySQL toega veebiserveri olemasolu. Serveri puudumisel, on lugejal võimalik valida tasuta teenusepakkuja nagu www.zone.ee vahel või käivitada server lokaalselt. Lokaalselt serveri käivitamiseks soovitab autor Apache Friends-i poolt pakutavat XAMPP serverit, mis on tasuta allalaetav aadressilt http://www.apachefriends.org/en/xampp.html.

2.1. Õppematerjali analüüs

Otsides õppematerjale või koodinäiteid objektorienteeritud veebilahenduse loomisest, leiab tihtipeale suurtes kogustes erinevaid koodinäiteid. Aga valdav enamus neist ei anna lugejale täielikku ülevaadet, mida kogutud teadmistega peale hakata. Eriti siis, kui pole päris selge, mille kohta infot leida üritatakse. Näited on liiga abstraktsed ja õpetuses kajastatu liiga pealiskaudne.

Peamine suund tutvustamaks objektorienteeritust, on klassi defineerimine, selle loomine ja seejärel tema meetodi väljakutsumine (vt. koodinäide 1).

```
class Inimene{
    public function tervita(){
        echo "Tere maailm! See on minu esimene objekt. ";
    }
}
$inimene = new Inimene();
$inimene->tervita();
```
Koodinäide 1 – esimese objekti loomine

Selliste näidete probleem seisneb selles, et lugeja ei saa iseseisvalt materjaliga tutvudes piisavalt mõtteainet, kuidas omandatud teadmist kodulehe loomisel rakendada. Suhteliselt raske on jõuda andmebaasiga iseseisvalt suhtleva objektini või klassini, mille eesmärk on eristada inimest arvutist. See väide leiab kinnitust ka Martin Rebase poolt loodud eesti keelse õpetuse Objektorienteeritud programmeerimine PHP-s õpetuse http://www.php.ee/4732 kommentaariumis aadressil http://www.php.ee/foorum/index.php?post=7430. Kasutajate soov oleks näha õpetust millegi kohta, mida ka reaalselt kasutada saaks. Inglise keeles leidub objektorienteeritud programmeerimise õppematerjali rohkem kui eesti keeles, kuid algajatele mõeldud näited on liiga laialivalguvad, nõuavad erialase inglise keele teadmiseid ja ei anna lugejale ülevaadet, mida antud teadmistega tulevikus teha saab.

Läbides algajale mõeldud inglise keelset objektorienteeritud PHP õpetust aadressil http://net.tutsplus.com/tutorials/php/object-oriented-php-for-beginners/ hakkab silma nii mõndagi positiivset kui ka negatiivset. Materjali saab jaotada kaheksasse peatükki:

1. Objektorienteeritud programmeerimise tutvustamine

Materjal selgitab piltlikult ja arusaadavalt lugejale objektorienteeritud programmeerimise põhimõtteid paberil oleva majaplaani ja kividest ehitatud maja kaudu. Majaplaan sümboliseerib klassi ja reaalselt olemasolev maja objekti.

2. Klasside struktuuri tutvustamine

Materjal jätkub klassi defineerimiseks vajalike märksõnade tutvustamisega ja seejärel objekti initsialiseerimisega.

3. Atribuutide defineerimine

Seejärel tutvustatakse lugejale atribuutide defineerimist ja nende kasutamist. Peatükk lõppeb illustratiivse näitega, mille käigus initsialiseeritakse objekt ja prinditakse välja atribuudi väärtus.

4. Isendimeetodite defineerimine

Materjal jätkub meetodite tutvustamise ja defineerimisega. Meetodite tutvustamisel käsitletakse ka konstruktorite ja destruktorite kasutamist koos illustreeriva näitega.

5. Klassi pärimine

Seejärel tutvustatakse lugejale lühidalt klasside pärimist, kuid puudulikuks on jäänud kirjeldav osa, millal kasutada klasside pärimist ja mis on pärimise eesmärk. Lugejale ei tutvustata klasside pärimise põhimõtet, vaid näidatakse koodinäitega kuidas klassipärimist teostada.

6. Meetodite ülekirjutamine

Seejärel tutvustatakse lugejale päritud klassi meetodi ülekirjutamist. Kuid puudulikuks on järjekordselt jäänud kirjeldav osa, millal tuleks meetodeid üle kirjutada ja mis on ülekirjutamise eesmärk.

7. Ligipääsetavuse määramine atribuutidele ja meetoditele

Objektorienteerituse tutvustamine jätkus atribuutidele ja meetoditele ligipääsetavuse defineerimisega. Kirjeldused on tehtud ruttavalt ja ühelauselised, millest oleks võinud kirjutada kindlasti pikemalt.

8. Staatilised atribuudid ja meetodid

Materjal lõpetatakse klassimeetodite ja klassiatribuutide tutvustamisega, milles on puudulikult kirjeldatud nende kasutamist. Kirjelduslik osa oleks pidanud kindlasti pikem olema, sest staatiliste meetoditest ja atribuutidest arusaamine on objektorienteeritud programmeerimises ääretult tähtis.

Läbitud inglise keelne õppematerjal annab lugejale ülevaate koos vajaminevate teadmistega objektide loomisest, kuid kirjelduslik osa on seejuures väga puudulik. Sissejuhatavad osad on väga puudulikud ja põhirõhk on koodinäidetel, mis samuti ei ole väga õnnestunud. Taolistes õppematerjalides tutvustatakse lugejale objekti kirja panemist. Sellised näited ei ole praktilised ja ei anna lugejale ideid funktsionaalsuse korrektseks jagamiseks. Käesolevas õppematerjalis on püütud eelnevalt mainitud vigasid vältida ja kasutada ainult praktilisi näiteid koos sinna juurde kuuluvate kirjeldustega. Õppematerjal on püütud teha lugeja jaoks võimalikult lihtsaks, arusaadavaks ja loogiliselt järjestatuks. Käesolevas töös sisalduvad

näited moodustatakse tüki kaupa loogilises valmimise järjekorras ja arusaadavuse huvides on töö lisades saadaval näidete terviklikud koodid. Näidete loomisel on lähtutud Kevin Skoglundi videomaterjalist PHP with MySQL Beyond the Basics, mis on leitav aadressilt http://www.lynda.com/PHP-tutorials/php-with-OOP-beyond-the-basics/653-2.html.

2.2. Õppematerjali kirjeldus

Õppematerjal hõlmab peamiselt protseduurilisest programmeerimisest tuntud tegevuste esitlemist objektorienteeritud lähenemise kaudu.

Juhendmaterjal sisaldab endas 15 peatükki, mille võib jaotada kahte suuremasse ossa.

Esimene osa, mis käsitleb esimest seitset peatükki, on sissejuhatus objektorienteeritud programmeerimisse.

Teine osa, mis käsitleb ülejäänud kaheksat peatükki, sisaldab endas praktilisi näiteid, mis keerukamaid veebilahendusi koostades programmeerijale abiks on.

1. **Sissejuhatus objektorienteeritud veebilahendusse**

Õppematerjali esimeses peatükis antakse lühiülevaade objektorienteeritud programmeerimisest ja kirjeldatakse hüvesid, mida selle rakendamine endaga kaasa toob.

2. **Klasside defineerimine**

Teises peatükis on kirjeldatud uue objekti loomise toimimise protsessi.

3. **Isendimeetodite ja –atribuutide defineerimine**

Kolmas peatükk käsitleb endas objektile meetodite ja atribuutide lisamist.

4. **Staatiliste meetodite ja atribuutide defineerimine**

Neljandas peatükis kirjeldatakse objekti staatiliste meetodite ja atribuutide defineerimist.

5. **Konstruktorite ja destruktorite kasutamine**

Viiendas antakse ülevaade objekti loomisel ja hävitamisel automaatselt väljakutsutavatest meetoditest.

6. Klasside pärimine

Kuuendas peatükis kirjeldatakse klasside pärimist ja sellega kaasnevaid hüvesid, mis väljenduvad koodi korduvkasutatavuse näol.

7. Objektide võrdlemine

Seitsmendas peatükis antakse lugejale ülevaade objektide võrdlemisest ja erinevatest operaatoritest, mida selleks on võimalik kasutada.

8. Objektorienteerituse praktiline kasutamine

Kaheksandas peatükis luuakse sissejuhatavates peatükkides õpitud võtetega hüperlingi objekt, mille eesmärk on lihtsustada linkide genereerimist.

9. Andmebaasi objekti loomine

Üheksandas peatükis luuakse sõltumatu andmebaasiga suhtlev objekt, mis lihtsustab andmebaasist päringute tegemist.

10. Sessioonide kasutamine

Kümnendas peatükis kirjeldatakse sessioonide kasutamist, mis tähendab aktiivse seansi raames andmete meeldejätmist.

11. CAPTCHA loomine

Üheteistkümnendas peatükis kasutatakse kümnendas peatükis õpitud teadmisi, loomaks objekti, mille eesmärk on eraldada inimest arvutist.

12. Repaginatsiooni kasutamine

Kaheteistkümnendas peatükis antakse lugejale ülevaade repaginatsiooni kasutamisest ja sellest, kuidas kogu andmehulga asemel kuvada ainult teatud osa.

13. Klassimeetodite paindlikum kasutamine

Kolmeteistkümnendas peatükis kirjeldatakse lugejale PHP versiooniga 5.3 kaasnevat klassimeetodite paindlikumat kasutamist.

14. Abstraktsete klasside kasutamine

Neljateistkümnendas peatükis kirjeldatakse kasutajale abstraktsete klasside defineerimist ja nende kasutamist.

15. Liideste kasutamine

Viieteistkümnendas peatükis kirjeldatakse kasutajale liideste rakendamisega kaasnevaid hüvesid.

3. Õpiteed

Käesolev õppematerjal on peamiselt mõeldud isikule, kellel on huvi dünaamilise sisuga veebilehtede loomise vastu ja omab algteadmisi PHP kohta nagu:

- Süntaksi ja koodi täitmise järjekorra tundmine
- Funktsioonide ja muutujate kasutamine erinevates skoopides;
- Programmiliidese tundmine – oskab kasutada peamisi PHP-sse sisseehitatud funktsioon.

Õppematerjali vastu võivad huvi tunda ka isikud, kes ei ole otseselt veebiprogrammeerimisega kokku puutunud, kuid tunnevad teisi programmeerimiskeeli ja soovivad ülevaadet saada objektorienteerituse rakendamisest – põhimõte jääb erinevate keelte puhul samaks, muutub ainult koodi kirjapanemise viis.

3.1. Soovitused õppematerjali läbimiseks

Õppematerjal on eelkõige mõeldud iseseisvaks kasutamiseks või loengus abimaterjalina. Käesolevas töös sisalduvad näited on universaalsed, mille tõttu saab neid erinevate projektide raames ka korduvalt kasutada.

Õppematerjali edukaks läbimiseks, on autori arvates vajalik:

- Näidetest aru saamine. Juhendis sisalduvad näited on taaskasutatavad ja leiavad kindlasti kasutust mitmetes projektides. Sellepärast on lugeja arusaamine näidete sisulisest poolest ülimalt oluline, kuna tulevikus võib tekkida vajadus koodi muutmise järele.
- Objektide arhitektuurist aru saamine. Näidetes olevate klassi meetodite ja atribuutide toimimise mõistmine aitab kaasa iseseisvalt loodavate objektide tekitamisele, mis on algusest peale hästi läbi mõeldud.
- Tähelepanelik lugemine. Läbides õppematerjali, tuleb lugejal pöörata tähelepanu kirjutatavale koodile vältimaks kergesti sissetulevaid vigu nagu meetodi asetamine klassist väljapoole või atribuudi poole pöördudes vale märksõna kasutamine.

3.2. Iseharjutamise võimalused

Õppematerjal on eelkõige mõeldud iseseisvaks läbimiseks ja materjali paremaks kinnistumiseks on „Objektorienteerituse praktilise kasutamise“ alapeatükkide lõpus, on autori poolt väljamõeldud ülesanded, mida juhendist huvitatu võiks proovida realiseerida.

Lisades on saadaval peatükkides läbitavate näidete terviklikud koodinäited, kuhu iseseisvalt lahendatavate ülesannete lahendused on juba sisseviidud.

4. Õppematerjali testimine

Õppematerjali testimine on käesoleva bakalaureuse töö olulisematest osadest, mis viidi läbi kahes etapis, mida kirjeldatakse allpool täpsemalt. Testimise üheks eesmärgiks oli ebatäpsete ja segadust tekitavate selgituste avastamine ja parandamine juhendis. Lisaks pidas autor oluliseks testimisega välja selgitada, kas õppematerjal täidab oma eesmärki ja on abiks sihtgrupile objektorienteeritud veebilahenduse koostamisel PHP-s.

Esimeses testimise etapis olid õppematerjali kasutajateks kuus isikut, kes olid varasemalt kokkupuutunud protseduurilise programmeerimisega, kuid puudusid teadmised objektorienteeritud veebilahenduste koostamisest. Esimene etapp oli autori jaoks kõige olulisem, sest esmasest tagasisidest lähtuvalt sai autor ülevaate, kas koostatud õppematerjal on lugejatele arusaadav ja täidab oma eesmärki. Lisaks viis autor peale esimest testimise etappi õppematerjalis läbi suuremad parandused ja muudatused.

Testimise teises etapis saadeti parandatud õppematerjal lugemiseks kuuele sihtgruppi kuuluvale tudengile koos tagasiside andvate küsimustega. Testijad tegid materjalis toodud näited kaasa ning vastasid seejärel küsimustele. Küsimused eeldasid põhjalikumat vastust ja saadud tagasiside abil sai autor ülevaate õppematerjalis parandust vajavatest nõrkadest kohtadest.

Testimise etappide käigus kogutud informatsiooni põhjal viis autor õppematerjalis läbi lõplikud muudatused, viies sisse parandusi ja täpsustusi.

4.1. Testimise esimene etapp

Käesoleva õppematerjali esimene etapp viidi läbi peale esimese versiooni valmimist. Eesmärgiks oli juhendis esinevate vigade parandamine ning probleemsete kohtade välja selgitamine ja paremini mõistetavaks muutmine. Ühtlasi soovis autor veenduda, kas objektorienteerituse tutvustamiseks valitud variant on ennast õigustanud. Testijate ülesandeks oli juhendis kirjeldatu läbi tegemine ja sellest arusaamine. Kui küsimuste esinemise korral õppematerjali korduval lugemisel vastuseid ei saadud, edastati need hiljem koos soovitustega õppematerjali parandamiseks tagasisidena autorile.

Õppematerjali esimesse etappi valiti testijateks kuus informaatika eriala õppinud ja sellega igapäevaselt tegelevat isikut, kes kuulusid sihtgruppi ja omasid varasemat kokkupuudet protseduurilise programmeerimisega PHP-s. Testis osalenud ei olnud teadlikud objektorienteeritud veebilahenduste koostamisest PHP-s, kuid kolm isikut tegelesid programmeerimisega igapäevaselt ja olid objektorienteeritud programmeerimisega varasemalt kokkupuutunud teistes keeltes nagu Java ja C.

Testimise esimene etapp möödus edukalt ja ületas autori esialgseid ootusi. Juhendi esimeses osas parandati väiksemad vead isendimeetodite ja –atribuutide defineerimise peatükis. Teises osas parandati sõnastust segadust tekitavates kohtades või lisati juurde selgitavaid lauseid klassimeetodite ja abstraktsete klasside peatükis. Esimese etapi lõppedes muutus juhend paremini loetavamaks ja algajale mõistetavamaks.

4.2. Testimise teine etapp

Testimise teiseks etapis moodustas autor tagasiside saamiseks üksteist küsimust, mis saadeti uuele testgrupile koos esimese etapi käigus parandatud versiooniga. Testgruppi kuulusid kuus informaatika eriala tudengit, kellest kaks olid varasemalt kokkupuutunud objektorienteeritud veebilahenduse koostamisega PHP-s. Sarnaselt esimesele etapile, pidid testijad õppematerjalis kirjeldatu läbi tegema ja lisaks vastama tagasiside andmiseks küsimustele.

Testgrupile esitatud küsimused:

1. Milliste programmeerimiskeeltega olete varasemalt kokkupuutunud?

Testis osalenute programmeerimiskeelte kogum oli väga mitmekesine: Java, C, C++, Python, Objective-C, Javascript, PHP, Actionscript, C#. Iga testis osalenu oli varasemalt kokkupuutunud vähemalt kolme programmeerimiskeelega.

2. Kas olete teadlik objektorienteeritud programmeerimisest?

Testis osalenud oli teadlikud objektorienteeritud programmeerimise põhimõtetest ja sellega varasemalt kokkupuutunud.

3. Kas olete kokkupuutunud protseduurilise programmeerimisega PHP-s?

Testgruppi kuulunud kuuest isikust ei olnud üks inimene varasemalt protseduurilise programmeerimisega PHP-s kokkupuutunud.

4. Kas juhendi esimene peatükk, sissejuhatus objektorienteeritusse, oli arusaadav? Kui midagi jäi segaseks, siis milline osa täpsemalt?

Autori arvates kõige olulisema peatükiga lugejatel probleeme ei tekkinud. Ühele isikule, kes programmeerimisega igapäevaselt kokku ei puutunud, muutus sissejuhatus objektorienteeritusse arusaadavaks alles mitmekordsel lugemisel. Sellest tulenevalt lisati sissejuhatusse kirjeldavaid lauseid juurde.

5. Kui arusaadavad olid materjalis toodud koodinäited?

Koodinäited olid testijate jaoks arusaadavad. Kuna näidetest arusaamisega ei tekkinud ühelgi testis osalenul probleeme, siis koodinäidetes muudatuste läbiviimiseks autor vajadust ei näinud.

6. Kui arusaadavad olid koodinäidetega kaasaskäivad kirjeldused?

Testis osalenute arvates olid koodinäidetega kaasaskäivad kirjeldused selged ja konkreetsed, kuid positiivsest tagasisidest hoolimata pidas autor vajalikuks parandada sõnastust klassimeetodite paindlikuma kasutamise peatükis.

7. Kas juhendi lugemisel tekkis Teil ideid loetud teadmiste rakendamiseks?

Õppematerjali lugemise ajal tekkis neljal isikul ideid loetud teadmiste rakendamiseks. Peamiselt andis lugejatele uusi mõtteid klassimeetodite paindlikuma kasutamise peatükk.

8. Kas juhendis toodud näited olid teie arvates praktilised ja võivad abiks olla objektorienteeritud veebilahenduse koostamisel?

Testis osalenute arvates olid õppematerjali valitud näited praktilised ja kasulikud. Kahe isiku arvates oli CAPTCHA näide väga hästi õnnestunud.

9. Kas teie arvates on juhend piisav andmaks ülevaadet objektorienteeritud veebilahenduse loomisest?

Testgrupi arvates oli juhend piisav andmaks ülevaadet objektorienteeritud veebilahenduse loomisest. Ühe isiku arvates oleks võinud keerulisemates peatükkides nagu abstraktsete klasside kasutamine, tuua arusaadavuse huvides rohkem näiteid.

10. Kas oleksite eelistanud õppematerjalina lugeda terviklahenduse loomist või sobis Teile õppematerjalis valitud variant?

Testgrupis osalenute vastused jagunesid pooleks. Kolm isikut oleksid eelistanud õppematerjalina lugeda objektorienteeritud tervikveebilahenduse loomist. Ülejäänud testis osalenud isikutele sobis õppematerjalis valitud variant, milleks oli näide igast valdkonnast.

11. On teil ettepanekuid õppematerjali parandamiseks?

Õppematerjali loetavuse parandamiseks soovitati autoril kujundust parandada. Lisaks pakuti välja iseseisvatele ülesannetele juurde lisada kirjeldav lahenduskäik.

Autor pidas testimise teist etappi edukaks, sest testimise käigus tulid välja esimeses etapis märkamatuks jäänud vead. Esimeses osas lisati selgitavaid lauseid juurde peamiselt klasside pärimise ja andmebaasi objekti peatükis. Samuti anti autorile asjalikke näpunäiteid õppematerjali täiustamiseks.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli tutvustada lugejale objektorienteeritud veebilahenduse koostamise võimalusi PHP-s ja analüüsida olemasolevaid materjale.

Töös tutvustati skriptimiskeelt PHP ja anti ülevaade, milleks ta mõeldud on. Lisaks kirjeldati objektorienteeritud programmeerimise põhimõtteid ja selle rakendamist PHP-s. Seejärel analüüsiti olemasolevaid eesti- ja inglisekeelseid PHP õppematerjale ja toodi välja nende nõrgad kohad, mida valminud õppematerjalis vältida püüti. Lisaks kirjeldati soovitatavaid õppematerjali läbimise meetodeid ja iseharjutamise võimalusi. Käesoleva bakalaureusetöö põhitulemuseks oli PHP-s objektorienteerituse rakendamise õppematerjal, mis koosnes kahest peamisest peatükist, sissejuhatusest ja praktilistest näidetest. Õppematerjali kasutuskõlblikkust testiti kahes etapis ja jõuti autorit rahuldava tulemuseni.

Bakalaureusetöö andis autorile arvestataval määral uusi teadmisi tänapäeval laialt kasutusel olevast skriptimiskeelest PHP ja antud keeles objektorienteerituse rakendamisest.

Autori hinnangul täitis käesolev bakalaureusetöö oma ülesande, kuna töö käigus täideti püstitatud eesmärgid.

Summary

The aim of the current Bachelor's thesis was to introduce to the reader the building opportunities of object-oriented web solutions in PHP and to analyze the existing tutorials.

The first chapter of the thesis introduced the scripting language called PHP and the purpose of it. It also included the description of the principles of object-oriented programming and its implementation in PHP. The second chapter of the thesis contained the analysis of the existing PHP teaching tutorials in Estonian and English, identifying their weak spots, which this tutorial tried to avoid. After that the recommended methods for independent practicing and learning from the tutorial were introduced. The main result of this Bachelor's thesis was a tutorial that introduces the opportunities of building object-oriented driven web applications in PHP. The tutorial consists of two main chapters: introduction and practical examples. The usefulness of the tutorial was tested in two stages and the results were satisfactory in the author's opinion.

The current Bachelor's thesis gave a considerable amount of new knowledge to the author about the scripting language PHP that is nowadays widely used, and how to implement it.

In the author's opinion, this Bachelor's thesis filled its purpose because all the objectives were achieved.

Kasutatud kirjandus

Abstract Class and Interface (2008). Viimati vaadatud 22.04.2012 aadressil http://www.sunilb.com/php/php5-tutorials-abstract-class-and-interface

Agrawal, H. (2007). OOPS in PHP 5 Tutorial – Using Interface. Viimati vaadatud 23.04.2012 aadressil http://www.hiteshagrawal.com/php/oops-in-php5-using-interface

Jentson, I. (1999). Struktuur- ja objektorienteeritud programmeerimise põhimõtted. Viimati vaadatud 24.03.2012 aadressil http://nrg.tartu.ee/algkursus/Teema12.htm

Kepple, D. (2010). Five Reasons to use Setters and Getters. Viimati vaadatud 28.03.2012 aadressil http://active.tutsplus.com/tutorials/actionscript/as3-101-five-reasons-to-use-setters-and-getters/

Lewallen, R. (2005). Advantages of an Object-Oriented Approach. Viimati vaadatud 27.03.2012 aadressil http://codebetter.com/raymondlewallen/2005/02/08/advantages-of-an-object-oriented-approach-for-new-programmers/

Mitchell, L. (2009). PHP 5.3 Feature: Late Static Binding. Viimati vaadatud 20.04.2012 aadressil http://www.lornajane.net/posts/2009/php-5-3-feature-late-static-binding

Patel, R. (2009). Facts of PHP Programming Language. Viimati vaadatud 22.03.2012 aadressil http://www.programmingfacts.com/facts-of-php-programming-language/

PHP sessions (2008). Viimati vaadatud 17.04.2012 aadressil http://www.tizag.com/phpT/phpsessions.php

Skoglund, K. (2009). PHP with MySQL Beyond the Basics. Viimati vaadatud 28.03.2012 aadressil http://www.lynda.com/PHP-tutorials/php-with-OOP-beyond-the-basics/653-2.html

What Is A CAPTCHA (2012). Viimati vaadatud 18.04.2012 aadressil http://www.google.com/recaptcha/captcha

What is PHP? (2012). Viimati vaadatud 22.03.2012 aadressil http://www.webopedia.com/TERM/P/PHP.html

Why use Object-Oriented Design. (2012). Viimati vaadatud 24.03.2012 aadressil http://www.mathworks.se/help/techdoc/matlab_oop/bri1rtu-1.html

Lisad

Lisa 1. Objektorienteeritud veebilahenduse koostamise õppematerjal

1. Sissejuhatus objektorienteeritud veebilahendusse

Luues veebilahendusi, puututakse tüüpiliselt arendusprotsessi alguses kokku disainimisega, mille eesmärk on kaardistada andmete esitlemine ja välja selgitada kuidas nendega operatsioone teostada. Protseduurilises programmeerimises edastatakse andmed funktsioonidele, kuid objektorienteeritud programmeerimises kapseldatakse andmed ja nendega seonduvad tegevused objektide sisse, mis suhtlevad omavahel vastastikku. Kui on kaks toimivat lähenemist, millest ükski pole vale, siis miks kasutada objektorienteeritud lähenemist? Protseduurilises programmeerimises keskendub disain sammudele, mis on vajalikud jõudmaks soovitud seisundisse. Enamasti kutsutakse välja funktsioonide jada, kuhu saadetakse töötlemata- ja saadakse tagasi töödeldud andmed. Objektorienteeritud programmeerimises keskendub disain:

- Süsteemi poolt kasutatavate komponentide tuvastamisele
- Analüüsile ja mustrite identifitseerimisele, mis määrab süsteemi poolt korduvkasutatavad või sarnaseid omadusi jagavad komponendid
- Sarnaste ja erinevate komponentide klassifitseerimisele

Peale eelmainitud tegevusi defineeritakse klassid, mis kirjeldavadki süsteemi poolt kasutatavaid objekte (Why Use Object-Oriented Design, 2012). Korralikult disainitud süsteem toob endaga kaasa mitmeid hüvesid (Lewallen, 2005):

- Hooldatavus – meetodid on paremini hallatavamad ja vigade tuvastamine on kergem, kuna kood on kapseldatud objektide põhiselt.
- Korduvkasutatavus – objektid on sõltumatud ja seetõttu on neid võimalik rakendada ka uutes süsteemides.
- Eraldatus – objekti väljavahetamisel puudub vajadus muuta ülejäänud koodi.

1.1. Klasside defineerimine

Objekti tekitamine algab objektitüübi ehk klassi defineerimisega ja märksõnaga `class`. Uue objekti tekitamiseks tuleb koodi kirjutada `new Klassinimi()`. Klassi nimi on teksti tüüpi ja peaks viitama objektiga seonduvale. See peaks olema võimalikult lühike, kirjeldav ja üheselt mõistetav, sest suurte projektide korral võib olla eelnevalt defineeritud ja klassifitseeritud sadu objekte ja on hea, kui koodis orienteerumisele liigselt aega ei kulutata. Näiteks ehitades objekti, mille ülesandeks on linkide genereerimise lihtsustamine, võib klassile nimeks anda Link (vt. koodinäide 2).

```
class Link{
}
$link = new Link();
```

Koodinäide 2 – URL objekti loomine

1.2. Isendimeetodite ja -atribuutide defineerimine

Protseduurilises programmeerimises puudub võimalus piirata ligipääsu funktsioonidele või muutujatele. Nad kas eksisteerivad ja on ligipääsetavad või mitte. Kuid objektorienteeritud programmeerimises on võimalik määrata objekti meetoditele ja atribuutidele ligipääsu piiranguid, mida on kolme tüüpi:

1. `public` – objekt või atribuut on ligipääsetav igalt poolt.
2. `private` – objekt või atribuut on ligipääsetav ainult objekti seest.
3. `protected` – objekt või atribuut on ligipääsetav tema klassi või alamklassi seest.

Levinud moodus objekti tekitamisel, on tema atribuudi ligipääsu piiranguks `private` määramine ja seejärel `get` ja `set` meetodite kasutamine. Selline lähenemine on kasulik mitmel juhul (Kepple, 2010):

- Atribuudid on kirjutuskaitsega – määrates atribuudi privaatseks ja omistades talle avaliku `get` meetodi, eemaldatakse atribuudi ülekirjutamise võimalus. Seda saab lugeda, kuid mitte muuta. See on kasulik, kui atribuutide väärtused ei tohi peale objekti tekitamist muutuda.

- Andmete valideerimine – enne atribuudile väärtuse omistamist, on võimalik valideerida andmete õigsust ja vajadusel määrata vaikimisi väärtused.
- Andmete kombineerimine – andmeid on võimalik tuletada lähtudes atribuutidest.
- Keeruka koodi peitmine lihtsa atribuudi taha – väärtuse määramisel võib süsteem lisaks andmete omistamisele täita veel omakorda erinevaid ülesandeid.

Objekti atribuudi poole pöördumiseks on vajalik märksõna `$this->atribuudiNimi` kasutamine. Koodinäites 3 on toodud atribuudid ning nende küsimise ja määramise meetodid.

```
class Link{
    private $aadress   = '', $failiNimi = '';
    public function setAadress($aadress){
        $this->aadress = $aadress;
    }public function getAadress(){
        return $this->aadress;
    }
    public function setFailiNimi($failiNimi){
        $this->failiNimi = $failiNimi;
    }
    public function getFailiNimi(){
        return $this->failiNimi;
    }
}
```

Koodinäide 3 – URL objekti atribuutide küsimine ja määramise meetodid

1.3. Staatiliste meetodite ja atribuutide defineerimine

Klassi atribuudid või meetodid on võimalik deklareerida sarnaselt staatiliseks sarnaselt funktsioonibloki muutujatele. See annab võimaluse andmetele või tegevustele ligipääsemiseks ilma objekti initsialiseerimata. Kuid meeles tuleb pidada seda, et objekti seest otsene ligipääs puudub. See tähendab, et `$this->atribuudiNimi` ei tööta. Objekti väliselt tuleb andmetele ligipääsemiseks kasutada `klassinimi::atribuudinimi`, objekti sees olles `self::atribuudinimi` või alamklassis olles `parent::atribuudiNimi`.

Koodinäites on 4 on defineeritud klass staatilise meetodi ja atribuudiga.

```
class Link{
    private static $url = 'http://www.postimees.ee';
    static function kuva_Link(){
        echo self::$url;
```

```
    }
}
Link::kuva_Link();
```
Koodinäide 4 – Staatilise meetodi ja atribuudiga objekt

1.4. Konstruktorite ja destruktorite kasutamine

Konstruktorid ja destruktorid on spetsiaalsed meetodid, mis kutsutakse välja automaatselt objekti loomisel või hävitamisel. Konstruktori idee seisneb sellest, et objekti tekitades kutsutakse välja kindel meetod, mille sees käivituvad objekti loomiseks vajalikud tegevused, enne kui teda reaalselt kasutama hakatakse. Erinevates programmeerimiskeeltes on konstruktori nimi sama, mis klassinimi, aga alates PHP versioonist 5, on võimalik kasutada universaalset konstruktori ja destruktori kirja panemise meetodit (Skoglund, 2009). PHP-s ei ole destruktorite kasutamine väga levinud, kuid võivad osutuda väga kasulikuks, kui objekt peab valmistuma enda hävitamiseks. Näiteks salvestama erinevaid väärtuseid andmebaasi või uuendama globaalset logifaili.

Koodinäites 5 initsialiseeritakse objekt konstruktorit kasutades.

```
class Link{
    private $url = '';
    function __construct($url){
        //PHP5 konstruktor
        $this->url = $url;
    }
    function __destruct() {
        //PHP5 destruktor.
        //salvesta url andmebaasi
    }
}
$link = new Link('http://www.postimees.ee');
```
Koodinäide 5 – URL objekti initsialiseerimine konstruktori kaudu

1.5. Klasside pärimine

Klasside pärimine on objektorienteeritud programmeerimise üks kasulikumaid omadusi. Pärimine võimaldab defineerida ühe põhiklassi ja sellest omakorda pärinevad alamklassid. Alamklass pärib põhiklassi meetodid ja atribuudid, mille ligipääsu piiranguks on määratud kas `public` või `protected`. Selle tulemusena on ühine kood defineeritud ainult korra ja

koodi dubleerimise vajadus puudub. Klassi pärimiseks on vajalik `extends` märksõna kasutamine. Näiteks `alamklass extends põhiklass`, kuid meeles tuleb pidada seda, et põhiklassi atribuudid ja meetodid ei ole nähtavad alamklassi seest `$this->` märksõna kasutades, vaid kasutada tuleb `parent::` märksõna.

Nagu eelnevalt mainitud, kanduvad alamklassi üle põhiklassi avalikud ja kaitstud meetodid. Kuid võib tekkida olukord, et päritava klassi initsialiseerimiseks on vajalik argumentide edastamine konstruktorisse. Selleks tuleb alamklassis välja kutsuda põhiklassi konstruktor märksõnaga `parent::__construct($argument, ...)`. Lisaks võib tekkida olukord, kus alamklassi jaoks ei ole põhiklassi meetodite tegevuste jada korrektne ja on vajalik meetodi sisu ülekirjutamine. Märksõnaks on ülelaadimine, mille sisu seisneb selles, et alamklassis defineeritakse põhiklassi meetod uuesti ja ülelaetavat meetodit väljakutsudes käivitatakse süsteemi poolt alamklassi, mitte põhiklassi tegevuste jada (Skoglund, 2009).

Koodinäites 6 on toodud põhiklassi pärimine ja tema meetodi ülekirjutamine.

```
class Link{
    protected $url = '';
    function __construct($url) {
        $this->url = $url;
    }
    public function kuva_link(){
        echo '<a href="'.$this->url.'">Portaal Postimees.ee</a><br />';
    }
}

class ConfirmLink extends Link{
    public function __construct($url) {
        parent::__construct($url);
    }
    //meetodi ülekirjutamine
    public function kuva_link() {
        echo "<a onclick=\"return confirm('Oled kindel?')\" href=\"".$this-
>url."\">Uudiste portaal Postimees.ee</a><br />";
    }
}
$põhiklass = new Link('http://www.postimees.ee/');
$põhiklass->kuva_link();
$alamklass = new ConfirmLink('http://www.postimees.ee/');
$alamklass->kuva_link();
```

Koodinäide 6 – Klassi pärimine ja meetodi ülelaadimine

1.6. Objektide võrdlemine

PHP-s on objektide võrdlemiseks võimalik kasutada kahte operaatorit. Nendeks on võrdlemise operaator == ja identiteedi operaator ===. Kasutades võrdlemise operaatorit, võrdleb PHP objektide atribuute, mitte seda kas nad on põhimõtteliselt üks ja sama objekt. Identiteedi operaator seevastu nõuab võrreldavatelt objektidelt seda, et nende refereeritav objekt oleks üks ja sama (Skoglund, 2009).

Operaatoreid kasutades saadakse teada, kas erinevad muutujad viitavad samale instantsile või on tegu eraldiseisvate objektidega, mis kõigest näevad sarnased välja. Järgnevas tabelis (Tabel 1) on välja toodud erinevate objektide võrdlemisel operaatorite poolt tagastatavad tõeväärtused.

Tabel 1 – PHP operaatorite tagastatavad tõeväärtused

	==	===
Viide samale objektile	Tõene	Tõene
Instantsid võrdsete atribuutidega	Tõene	Väär
Instansid erinevate atribuutidega	Väär	Väär

2. Objektorienteerituse praktiline kasutamine

Objektorienteerituse praktiseerimine aitab programmeerija tööd kergemaks teha ka elementaarsete tegevuste juures ja ei ole mõeldud koodi lihtsustamiseks ainult keerukamate ülesannetega kokku puutudes. Süsteemi tööks vajalikud korduvkasutatavad komponendid toob tavaliselt disain välja, kuid algteadmisi kasutades võib objektidena vaadelda HTML-st tuntud hüperlinke ja neid sellistena ka kohelda. Idee seisneb selles, et käsitsi parameetrite sisestamise asemel edastatakse need hüperlingi objektile staatilise meetodi kaudu kujul `Link:: build_link ('http://www.koduleht.ee', 'index.php', 'Vajuta siia', 'id=834, mode=view')`, mis tagastab automaatselt genereeritud HTML lingi `<a href="http://www.koduleht.ee/example.php?id=834&mode=view">Vajuta siia</a>`. Ehitamaks hüperlingi objekti, mis eelmainitud ülesande sooritamisega hakkama saaks, on vajalik nelja atribuudi ja nende initsialiseerimismeetodite olemasolu. Nendeks on (vt. koodinäide 7):

1. Internetiaadress
2. Failinimi
3. Kasutajale kuvatav tekst
4. Parameetrid

```
class Link{
    private $url = '';
    private $fileName = '';
    private $text = '';
    private $parameters = array();
    public function __construct($baseURL) {
        $this->url = $baseURL;
    }
    public function setText($text){
        $this->text = $text;
    }
    public function setFile($fileName){
        $this->fileName = $fileName;
    }
    public function addParameters($parameterName, $parameterValue){
        $this->parameters[$parameterName] = $parameterValue;
    }
}
```

Koodinäide 7 –Objekti atribuutide defineerimine

Peale objekti initsialiseerimist tuleb defineerida meetod, mis moodustaks toimiva lingi kasutades atribuutide väärtuseid (vt. koodinäide 8).

```
private function render(){
	$r = '';
	$baseUrl = $this->url;
	$params  = $this->parameters;
	$name    = $this->fileName;
	if(trim($name)!='')
		$r .= $baseUrl.'/'.$name;
	else
		$r .= $baseUrl;
	if(count($params)>0){
		$r .= '?';
		$i = 0;
		foreach($params as $parameterKey => $parameterValue){
			if($i>=1)
				$r .="&";
			$r.= $parameterKey .'='. $parameterValue;
			$i++;
		}
	}
	return $r;
}
public function renderLink(){
	$r = '';
	$url = $this->render();
	$t = $this->text;
	if(trim($t) == '')
		$t = 'Lingi ehitamiseks puudub tekst';
	$r .= '<a href="'.$url.'">'.$t.'</a>';

	return $r;
}
```

Koodinäide 8 – Lingi ehitamise meetodi defineerimine

Kui objekti atribuutide initsialiseerimiseks ja lingi ehitamiseks on vajalikud meetodid defineeritud, tuleb lõpetuseks kirjutada staatiline meetod, mille väljakutsudes tekitatakse automaatselt hüperlingi objekt (vt. koodinäide 9).

```
public static function build_link($url, $fileName, $text, $parametersList){
	$link = new Link($url);
	$link->setFile($fileName);
	$link->setText($text);
	$parameterPairs = explode(',', $parametersList);
	if(count($parameterPairs)>0){
		foreach($parameterPairs as $parameterPair){
			$parts = explode('=', $parameterPair);
			$key   = trim($parts[0]);
			$value = trim($parts[1]);
```

```
                $link->addParameters($key, $value);
            }
        }
        return $link->renderLink();
}
```

Koodinäide 9 – Hüperlingi objekti initsialiseerimise staatiline meetod

Alternatiivina on võimalik defineerida funktsioon, mille ülesandeks on parameetritest lähtuvalt lingi genereerimine nagu eelnevalt defineeritud objektil. Kuid võrreldes objektide põhise lähenemisega pole protseduuriline programmeerimine kaugeltki nii hästi hallatav kui seda on objektorienteeritud programmeerimine (vt. koodinäide 10).

```
function build_link($source, $linkData, $message){
      if($source == null || $linkData == null ||
      $message==null || !(count($linkData) % 2 == 0)){
            return "Lingi ehitamine ebaõnnestus - parameetrite pikkus ei
jagu kahega või null-väärtus leitud";
      }
      $link="<a href='".$source."?";
      $count = count($linkData);
      for ($i = 0; $i < $count; $i+=2) {
            $link=$link.$linkData[$i]."=".$linkData[$i+1];
            if($i!=$count-2){
                  $link=$link."&";
            }
      }
      $link=$link."'>".$message."</a>";

      return ($link);
}
```

Koodinäide 10 – Lingi ehitamise funktsioon

Iseseisev ülesanne: Päri hüperlingi objekti omadused ja lisa järgnevad võimalused:

- Määra lingile HTML-st tuntud klassi määramise võimalus.
- Määra lingile stiili määramise võimalus.
- Lingile vajutades kinnituse küsimine, kui vastav seade on aktiveeritud. Kui ei ole, siis kinnitust ei küsita ja kasutaja suunatakse automaatselt refereeritavale aadressile.

Hüperlingi objekti terviklik kood on saadaval Lisa 2: „Hüperlingi objekt“.

2.1. Andmebaasi objekti loomine

Dünaamilise sisuga veebilehtede loomisel on andmebaasiga ühenduse loomine ja sealt andmete lugemine veebiarendaja jaoks rutiinne tegevus. Lihtsustamaks päringute tegemist, on arukas kasutada korduvkasutatavat komponenti, mis täidab iseseisvalt elementaarse ülesande nagu andmebaasiga ühendumine.

Andmebaasist päringuid tehes tuleb tähelepanu pöörata andmebaasi ridadele, mis on vaadeldavad eraldi objektidena. Andmebaasiga suhtlev objekt ei ole andmebaasis olev rida. See tähendab, et andmebaasi rea jaoks peaks tekitama omakorda uue klassi ja andmebaasist päringuid tegev klass vastutab ainult andmete tagastamise eest. Lisaks peaks objekt võimaldama ka süsteemiväliselt sisestatud parameetrite valideerimist ja nende muutmist vastavalt vajadusele, et andmed oleksid kõlbulikud andmebaasi sisestamiseks.

Korrektselt toimiva andmebaasi objekti loomiseks, on vajalik nelja atribuudi olemasolu. . Nendeks on (vt. koodinäide 11):

1. Päringute tegemiseks kasutatav ühenduse atribuut.
2. Vigade tuvastamiseks kasutatav viimasena sooritatud päring.
3. PHP-versiooni tuvastus, mida kasutatakse parameetrite valideerimisel.
4. Parameetrite asendamist võimaldava meetodi olemasolu kinnitus.

```
defined('DB_SERVER') ? null : define("DB_SERVER","serverinimi");
defined('DB_NAME  ') ? null : define("DB_NAME",  "andmebaasiNimi");    }
defined('DB_USER  ') ? null : define("DB_USER",  "andmebaasiKasutajanimi");
defined('DB_PASS  ') ? null : define("DB_PASS",  "andmebaasiParool");

class MySQLDatabase {

    public  $last_query;
    private $connection;
    private $magic_quotes_active;
    private $real_escape_string_exists;

    public function __construct() {
        $this->open_connection();
        $this->magic_quotes_active = get_magic_quotes_gpc();
        $this->real_escape_string =
function_exists("mysql_real_escape_string");
    }

    public function open_connection() {
        $this->connection = mysql_connect(DB_SERVER, DB_USER, DB_PASS);
```

```
        mysql_set_charset('utf8',$this->connection);
        if (!$this->connection) {
            die("Database connection failed: " . mysql_error());
        } else {
            $db_select = mysql_select_db(DB_NAME, $this->connection);
            if (!$db_select) {
                die("Database selection failed: " . mysql_error());
            }
        }
    }
}
$database = new MySQLDatabase();
```

Koodinäide 11 – Andmebaasi objekti loomine

Seejärel tuleb juurde lisada kolm meetodit: esimene, mis omistab päringu andmebaasile, teine, mis tagastab päringu tulemuse meetodi väljakutsujale ja kolmas, mis veendub käivitatava päringu õigsuses.

Koodinäites 12 on defineeritud päringu edastamise ja päringu valideerimise meetodid.

```
public function query($sql) {
      $this->last_query = $sql;
      $result = mysql_query($sql, $this->connection);
      $this->confirm_query($result);
      return $result;
}
public function fetch_array($result_set){
        return mysql_fetch_array($result_set);
}
private function confirm_query($result) {
      if (!$result) {
            $output = 'Päring luhtus, viga: ' . mysql_error() .'<br />';
            $output .= "Käivitatud SQL lausung: ".  $this->last_query;
            die($output);
      }
}
```

Koodinäide 12 – Andmebaasist pärimine ja võimalikust veast teatamine

Viimasena tuleb lisada meetod, mis muudab päringu andmebaasile kõlblikuks. Kõlblikuks muutmine tähendab, et päringus asendatakse näiteks jutumärgid kaldkriipsudega. Antud meetodit ei pea kasutama alati, vaid siis, kui andmebaasi päring sisaldab süsteemi väliselt, näiteks aadressirealt, sisseloetud parameetrit.

Koodinäites 13 on defineeritud andmete valideerimise meetod.

```
public function escape_value($value) {
	if ($this->real_escape_string_exists) { // Käesolev PHP versioon on
v4.3.0 või uuem
		if ($this->magic_quotes_active) {
			$value = stripslashes($value);
		}
		$value = mysql_real_escape_string($value);
	} else { //Käesolev PHP versioon on vanem kui v4.3.0
		if (!$this->magic_quotes_active) {
			$value = addslashes($value);
		}
	}
	return $value;
}
```

Koodinäide 13 – Andmebaasi päringu valideerimine

Andmebaasi objekti kasutamiseks tuleb eelnevalt moodustada vastav päring. Koodinäites 14 on defineeritud klassimeetod, mis pärib andmeid andmebaasist kasutades selleks andmebaasi objekti.

```
public static function find_by_id($id = 0) {
	global $database;
	$result_set = $database->query("SELECT * FROM kommentaarid WHERE id="
					. $database->escape_value($id));

	$object_array = array();
	while ($row = $database->fetch_array($result_set)) {
		$object_array[] = self::instantiate($row);
	}
	return $object_array;
}
```

Koodinäide 14 – Andmebaasi objekti kasutamine

Lisaks põhifunktsionaalsusele võib objektile lisada veel mitmeid andmebaasile omaseid meetodeid nagu näiteks viimase päringu töötlemisega seotud informatsioon, kasutusel olevast andmebaasist omadusi tagastavad meetodid ja palju muud.

Iseseisev ülesanne: Lisa andmebaasi objektile järgnevad võimalused:

- Päringus sisalduvate ridade kogu arv.
- Viimase sissekande jaoks automaatselt genereeritud id.
- Viimase päringu poolt mõjutatud ridade arv.

Andmebaasi objekti terviklik kood on saadaval Lisa 3: „Andmebaasi objekt“.

2.2. Sessioonide kasutamine

Tavaline HTML veebileht ei edasta andmeid ühelt lehelt teisele. Teisisõnu, kogu informatsioon unustatakse, kui laetakse uus leht. See teeb veebilehe toimimise tülikaks, kui on vaja alaliselt meeles pidada informatsiooni nagu sisseloginud kliendi kasutajanimi või toodete nimekirjast valitud ese. PHP sessioon lahendab eelmainitud probleemi, lubades serverisse salvestada ajutiselt informatsiooni, mida saab hiljem küsida ja töödelda. Tähelepanu tuleb pöörata sellele, et salvestatud informatsioon on lühiajaline ja hävitatakse peaaegu koheselt peale kasutaja veebilehelt lahkumist. Sessioonide põhimõte seisneb igale veebilehe külastajale unikaalse id genereerimises, kuhu salvestatakse sessiooni käigus erinevad muutujad. Sellega välditakse andmete segunemist, kui veebilehe külastajaid on üheaegselt rohkem kui üks (PHP sessions, 2008).

Loomaks objekti, mille ülesandeks on sisselogimise ees vastutamine, on vajalik kahe atribuudi olemasolu. Nendeks on:

1. Tõeväärtuse meeles pidaja – väärtus tõene, kui kasutaja on sisseloginud, vastasel juhul väär.
2. Sisseloginud kasutaja id.

Sessiooniobjekti idee seisneb selles, et iga kord, kui objekt luuakse, kutsutakse välja meetod, mille ülesanne on veenduda, kas eelnevalt on juba sisselogitud. Kui vastav viide on leitud, on kasutaja järelikult sisseloginud (vt. koodinäide 15).

```
class Session{
    private $logged_in = false;
    public $user_id;

    function __construct() {
        session_start();
        $this->check_login(); //kas on juba sisselogitud?
        if($this->logged_in){
            //vaja teha midagi, kui on sisselogitud?
        }else{
            //vaja teha midagi, kui pole sisselogitud?
        }
    }
    private function check_login(){
        if(isset($_SESSION['user_id'])){
            $this->user_id = $_SESSION['user_id'];
            $this->logged_in = true;
```

```
        }
        else{
            unset($this->user_id);
            $this->logged_in=false;
        }
    }
}

$session = new Session();
```

Koodinäide 15 – Sessiooni objekti loomine

Seejärel tuleb lisada meetod, mis jätab sessioonipõhiselt meelde kasutaja sisselogimise. Koodinäites 16 kasutatakse sisselogimise meelde jätmiseks kasutaja-objekti, millelt on võimalik küsida kasutajale omast unikaalset numbrit.

```
public function login($user){
      if($user){
            $this->user_id = $_SESSION['user_id'] = $user->id;
            $this->logged_in = true;
      }
}
```

Koodinäide 16 – Sisselogimise salvestamine sessiooni

Peale sisselogimise meetodi defineerimist, tuleb lisada väljalogimise meetod. Koodinäites 16 on defineeritud väljalogimise meetod.

```
public function logout(){
      unset($_SESSION['user_id']);
      unset($this->user_id);
      $this->logged_in = false;
}
```

Koodinäide 17 – Sessioonist väljalogimine

Lõpetuseks tuleb lisada meetod, mis tagastab tõeväärtuse, kas kasutaja on sisselogitud või mitte (vt. koodinäide 18).

```
public function is_logged_in(){
      return $this->logged_in;
}
```

Koodinäide 18 – Sisselogimises veendumine

Sisselogimise haldamiseks on hea praktika kasutada eraldi lehte, mille ülesanne on lugeda kasutaja sisseloginuks ja alustada sessiooni, kui on sisestatud korrektne kasutajanimi ja parool (vt. koodinäide 19).

```
if($session->is_logged_in()){
    redirect_to('index.php');
}
if(isset($_POST['submit'])){ //lehel on postitus tehtud
    $username = trim($_POST['username']);
    $password = trim($_POST['password']);

    $found_user = getUserFor($username, $password);

    if($found_user){ //kasutaja leitud
        $session->login($found_user);
        redirect_to('index.php');
    }
    else{
        //sisselogimine ebaõnnestus
    }
}
```

Koodinäide 19 – Sessiooni alustamine

Sessiooni objekti terviklik kood on saadaval Lisa 4: „Sessiooni objekt“.

2.2.1. CAPTCHA loomine

CAPTCHA on tehnoloogia (*Completely Automated Public Turing test to tell Computers and Humans Apart*), mille ülesanne on genereerida kergesti lahendatavaid ülesandeid, mille lahendamisega saavad inimesed, erinevalt arvutitest, hakkama (What Is A CAPTCHA, 2012). CAPTCHA aitab tõsta veebilehe turvalisust:

- Ennetades rämpsposti kommentaarides
- Kaitstes veebilehe registreerimisankeeti
- Kaitstes veebiküsitlusi

Üks kergematest moodustest rakendamaks CAPTCHAt veebilehe turvalisuse tõstmiseks, on genereerida veebilehe külastajale kaks juhuslikku numbrit, mille summat enne ülejäänud vormi töötlemist kontrollitakse.

Koodinäites 20 on defineeritud CAPTCHA objekt, mis initsialiseerides salvestab eelnevalt genereeritud juhuslikud numbrid atribuutidena ja hävitab sessioonis olevad väärtused, sest vastasel juhul võivad need hakata segama ülejäänud kodulehe toimimist.

```
class CAPTCHA {
    private $integer_one;
    private $integer_two;

    function __construct() {
        session_start();
        if (isset($_SESSION['numberOne'])) {
            $this->integer_one = $_SESSION['numberOne'];
            unset($_SESSION['numberOne']);
        }
        if (isset($_SESSION['numberTwo'])) {
            $this->integer_two = $_SESSION['numberTwo'];
            unset($_SESSION['numberTwo']);
        }
    }

}
$captcha = new CAPTCHA();
```

Koodinäide 20 – CAPTCHA initsialiseerimine

Seejärel tuleb lisada meetodid, mille väljakutsudes genereeritakse juhuslikud numbrid, mis salvestatakse sessioonis (vt. koodinäide 21).

```
public function generate_integer_one() {
      return $this->integer_one = $_SESSION['numberOne'] = rand('1', '19');
}
public function generate_integer_two() {
      return $this->integer_two = $_SESSION['numberTwo'] = rand('1', '19');
}
```

Koodinäide 21 – Juhuslike numbrite genereerimine

Viimasena tuleb defineerida meetod, millega kontrollitakse sisestatud summat. Kui summa võrdub juhuslikult genereeritud numbrite summaga, on test läbitud ehk tagastatav tõeväärtus on tõene (vt. koodinäide 22).

```
function isAnswer($answer) {
      return ($this->integer_one + $this->integer_two == $answer);
}
```

Koodinäide 22 – Sisestatud summa kontrollimine

Valikuliselt võib CAPTCHA objektile lisada ankeedi valideerimise abimeetodi. Tegu ei ole otseselt koodi osaga, mis on vajalik eristamaks inimest arvutist ja peaks seetõttu kuuluma mujale. Kuid abstraktselt mõeldes võib objekti vastutuseks olla sisendandmete töötlemine ja sellest tulenevalt on meetodi defineerimine õigustatud (vt. koodinäide 23).

```
function isInputDataCorrect() {
      if (!isset($_POST['submit']))
            return false;
      if (trim($_POST['firstName']) == '')
            return false;
      if (trim($_POST['lastName']) == '')
            return false;
      return true;
}
```

Koodinäide 23 – Ankeedi valideerimine

CAPTCHA kasutamiseks tuleb ühildada temasse sisseehitatud funktsionaalsus olemasoleva ankeediga.

Koodinäites 24 on näidisankeet CAPTCHA kasutamisest.

```
<?php
if($captcha->isInputDataCorrect() && $captcha->isAnswer($_POST['answer'])){
  echo 'Andmed on sisestatud korrektselt! <br />';
}
else {
?>
<div class="registrationForm">
<form action="" method="post">
  <table border="0">
    <tr>
      <td>Eesnimi: </td>
        <td>
          <input type="text" name="firstName" />
      </td>
    </tr>
    <tr>
      <td>Perenimi: </td>
      <td>
        <input type="text" name="lastName" />
      </td>
    </tr>
    <tr>
      <td>
          <?php echo 'Kui palju on?: '.$captcha->generate_integer_one().' +
'
            .$captcha->generate_integer_two(); ?></td><td><input
type="text" name="answer" /></td>
    </tr>
  </table>
<input value="Vajuta" name="submit" type="submit" />
```

```
</form>
</div>
<?php
}?>
```

Koodinäide 24 – CAPTCHA kasutamine

Iseseisev ülesanne: Lisa turvalisuse suurendamise võimalus külastaja blokeerimise näol, kui ankeedil genereeritud arvude summat on viis korda valesti sisestatud.

CAPTCHA objekti terviklik kood on saadaval Lisa 5: „CAPTCHA objekt“.

2.3. Repaginatsiooni kasutamine

Veebileheküljed, mille sisu pidevalt uueneb, võivad mahuliselt väga suureks kasvada. Andmebaasis võib olla näiteks 1000 sissekannet, mida peaks kuvama, kuid kogu info esitlemine ühel lehel, teeb selle lugemise peaaegu võimatuks. Probleemi lahenduseks on repaginatsioon (*pagination*), mis tähendab sisu jaotamist väiksemateks osadeks ehk erinevateks lehekülgedeks jagamist. Seda kasutatakse näiteks foorumites sissekannete ja otsingumootorites tulemuste kuvamiseks.

Repaginatsioon vajab kahe hulga olemasolu:

- Sissekannete alamhulk käesoleval leheküljel.
- Sissekannete koguhulk.

Rakendamiseks on vaja jooksvalt arvet pidada kolme väärtuse üle:

- Sissekannete koguarv.
- Lehel kuvatavate sissekannete arv.
- Mitmendal leheküljel parasjagu ollakse.

Eelnevalt mainitud väärtusi teades ja kasutades MySQL-st pärit märksõnu `LIMIT` ja `OFFSET`, on võimalik arvutada täpne sissekannetes paiknemine ja nendes orienteerumine.

Koodinäites 25 on defineeritud repaginatsiooni objekt koos klassiatribuutidega.

```
class Pagination{
    public $current_page, $per_page, $total_count;
    public function __construct($page=1, $per_page=20, $total_count=0) {
        $this->current_page = (int)$page;
        $this->per_page     = (int)$per_page;
        $this->total_count  = (int)$total_count;
    }
}
```

Koodinäide 25 – Repaginatsiooni objekti initsialiseerimine

Seejärel tuleb lisada meetod, mille ülesanne on kuvatava järjekorranumbri ehk `OFFSET`-i arvutamine (vt. koodinäide 26).

```
public function offset(){
      //Lehel kuvatakse 20 sissekannet. Esimese lehe OFFSET on 0 (1-1)*20
      //Teise lehe OFFSET on 20 (2-1)*20
      return ($this->current_page -1) * $this->per_page;
}
```

Koodinäide 26 – Järjekorranumbri arvutamine

Lisaks tuleb defineerida meetodi, mis vastutab leheküljenumbrite koguarvu arvutamise eest (vt. koodinäide 27).

```
public function total_pages(){
      return ceil($this->total_count/$this->per_page);
}
```

Koodinäide 27 – Leheküljenumbrite koguarvu arvutamine

Viimasena tuleb lisada meetodid, mis aitavad ehitada navigeerimisriba arvutades järgmise lehekülje järjekorranumbri (vt. koodinäide 28).

```
public function previous_page(){
      return $this->current_page-1;
}public function next_page(){
      return $this->current_page+1;
}
```

Koodinäide 28 – Sirvitavale leheküljele OFFSET-i arvutamine

Repaginatsiooni kasutamiseks tuleb leheküljel, kus kuvatavaid andmeid on ühel lehel korraga näitamiseks liiga palju, sisse arvutada repaginatsiooni objekti defineerimiseks vajalikud muutujad (vt. koodinäide 29).

```
$page = !empty($_GET['page']) ? (int)$_GET['page']:1; //sirvitava lk number
$per_page = 3; //kui mitut sissekannet tahetakse ühel lehel kuvada
$total_count = getTotalRecordsCount();//Sissekannete kogu arv
$pagination = new Pagination($page, $per_page, $total_count);
$sql = "SELECT * FROM TABELINIMI ";
$sql .= "limit {$per_page} ";
$sql .= "OFFSET {$pagination->offset()}";
$records = findAllRecords($sql);
```

Koodinäide 29 – Repaginatsiooni objekti muutujate arvutamine

Viimasena tuleb defineerida navigeerimisriba, mis lubab kiiresti kuvatava sisu vahel sirvida (vt. koodinäide 30).

```
for($i=1; $i <= $pagination->total_pages(); $i++){
    if($i == $page)
        echo ' <b>'.$i.'</b>';
    else
        echo ' <a href="index.php?page='.$i.'">'.$i.'</a>';
}
```

Koodinäide 30 – Navigeerimisriba ehitamine

Iseseisev ülesanne: Täienda navigeerimisriba lisades sinna funktsionaalsust, mis võimaldab ainult nooltele (`echo '&laquo';` ja `echo '&raquo';`) vajutades jõuda sisu algusest lõppu. Selleks tuleb repaginatsiooni objektile juurde lisada kaks meetodit:

- `has_previous_page()`
- `has_next_page()`

Repaginatsiooni objekti terviklik kood on saadaval Lisa 6: „Repaginatsiooni objekt“.

2.4. Klassimeetodite paindlikum kasutamine

PHP versiooniga 5.3 kaasnes lisa, mis lubas staatilisi meetodeid senisest paindlikumalt kasutada ehk *late static binding*. Enne antud funktsionaalsuse lisandumist, tekitas probleeme viis, kuidas PHP klassid üksteisele viitasid. Erinevates klasside hierarhias on objektidel ühine funktsionaalsus, kuid ühiseks kasutamiseks vajab täpsustamist klassi päritolu pääsemaks ligi tema nimele või temas defineeritud atribuutidele (Mitchell, 2009). Näiteks ehitades ettevõttele müüdud toodete tagasiside andmise võimalusega veebipoodi, tuleb arvatavasti kokku puutuda kahe järgneva klassiga:

- Müüdava eseme klass.
- Kommentaaride klass.

Mõlemal klassil leidub ühine funktsionaalsus:

- Kõikide esemete või kommentaaride leidmine
- Unikaalse numbri põhjal eseme lisainfo või eseme kohta käivate kommentaaride leidmine
- Objekti initsialiseerimine

Enne *late static bindingut* pidi veebilehe arendaja kopeerima sama funktsionaalsusega koodi igasse klassi eraldi, kuid alates PHP versioonist 5.3 on päritaval klassil võimalik edastada oma funktsionaalsus teda pärivale klassile abstraktselt (Mitchell, 2009).

Kasutamaks *late static bindingut* tuleb defineerida ühiste meetoditega objekt. Pääsemaks ligi meetodi väljakutsuja klassi atribuutidele või meetoditele, tuleb kasutada märksõna `static::väljakutsutavaNimi` (vt koodinäide 31).

```
class Object {
    public static function find_all() {
        return self::find_by_sql("SELECT * FROM " . static::$table_name);
    }
    public static function find_by_id($id = 0) {
        $result_array = self::find_by_sql("SELECT * FROM " .
        static::$table_name . " WHERE id={$id} LIMIT 1");
        return $result_array;
    }
    public static function find_by_sql($sql = "") {
        global $database;
        $result_set = $database->query($sql);
```

```
        $object_array = array();
        while ($row = $database->fetch_array($result_set))
            $object_array[] = self::instantiate($row);
        return $object_array;
    }
    private static function instantiate($record) {
        $classname = get_called_class();
        $object = new $classname($record);
        return $object;
    }
}
```

Koodinäide 31 – Late static bindingu kasutamine

Peale ühise funktsionaalsusega klassi defineerimist, tuleb luua objektid, mis selle omakorda pärivad (vt. koodinäide 32).

```
class Item extends Object{
}
class Comment extends Object{
}
```

Koodinäide 32– Ühise funktsionaalsuse pärimine

Ühise funktsionaalsuse kasutamiseks tuleb klassimeetod välja kutsuda, peale mida kasutatakse ülesannete täitmisel väljakutsuja klassi atribuute (vt. koodinäide 33).

```
Item::find_all();
Comment::find_by_id(3);
```

Koodinäide 33 – Ühise funktsionaalsuse väljakutsumine

Iseseisev ülesanne: Lisa ühise funktsionaalsusega objektile turvameetmed, mis kontrollib vastava atribuudi olemasolu, enne päringu sooritamist.

Klassimeetodite kood on saadaval Lisa 7: „Klassimeetodite paindlikum kasutamine“.

2.5. Abstraktsete klasside kasutamine

Abstraktne klass on klass, millest ei saa teha otseselt abstraktse klassi eksemplare ehk andmekandjad, kuid mis sisaldab endas teatud funktsionaalsust, mille teostamine on kohustuslik teda pärivatele klassidele. Klassi pärija peab tagama abstraktse klassi abstraktsete meetodite täitmise, vastasel juhul muutub klassi pärija samuti abstraktseks. (Abstract Class and Interface, 2008).

Abstraktsed klassid võivad lisaks abstraktsetele meetoditele sisaldada ka tavalisi meetodeid, koos eeldefineeritud funktsionaalsusega. Abstraktsetest klassidest ei saa objekte initsialiseerida, vaid instantse võib luua ainult konkreetsetest klassidest (Abstract Class and Interface, 2008).

Defineerimaks abstraktset klassi või abstraktset meetodit, on vajalik `abstract` märksõna kasutamine . Pärimaks abstraktse klassi abstraktseid meetodeid, on vajalik klassi pärimine `extends` märksõna abil.

Näiteks ehitades veebis mängitavat rollimängu, leiavad abstraktsed klassid rakendust vastavalt mängus tehtavatele valikutele (vt. koodinäide 34).

```
abstract class Weapon {

    private $nameOfTheGun;

    abstract public function fire();

    public function __construct($nameOfTheGun) {
        $this->nameOfTheGun = $nameOfTheGun;
    }
    public function getNameOfTheGun() {
        return $this->nameOfTheGun;
    }
}
```

Koodinäide 34 – Abstraktse klassi defineerimine

Seejärel tuleb defineerida abstraktsete meetodite sisu, mis on abstraktse klassi pärimisel kohustuslik (vt. koodinäide 35).

```
class Cannon extends Weapon {
    public function fire() {
        $this->needToReload = true;
        return $this->bullet;
    }
}
```

Koodinäide 35 – Abstraktsete meetodite täitmine

Iseseisev ülesanne: Tekita magasiniga relva klass, mis enne tulistamist veendub kuulide olemasolus.

Abstraktsete klasside kood on saadaval Lisa 8: „Abstraktsete klasside kasutamine“.

2.6. Liideste kasutamine

Liides on leping sidumaks kahte üksteisest sõltumatut objekti täitmaks ühist funktsionaalsust. Liides võimaldab täpsustada, et teatud objekt on võimeline täitma teatud meetodite kogumit, aga see ei dikteeri objektile, kuidas ta seda tegema peab. See tähendab, et iga klass vastutab ise lepingus defineeritud meetodite realiseerimise eest (Agrawal, 2007).

Defineerimaks liidest, on vajalik `interface` märksõna kasutamine. Omistamaks objektile lepingu täitmist, on vajalik `implements` märksõna kasutamine. Ühe objektiga saab teostada mitmeid liideseid.

Sarnaselt abstraktsetele klassidele, ei saa liidestest objekte luua, kuid erinevalt abstraktsetest klassidest, ei tohi liidesed sisaldada eeldefineeritud funktsionaalsusega meetodeid (Agrawal, 2007).

Koodinäites 36 on defineeritud liides, mille meetodite sisu peab täitma teostaja.

```
interface invoice {
    public function getCustomerName();
    public function getBillingAddress();
    public function getTotalAmount();
    public function sendEmail();
}
class check implements invoice {
    private $name, $address, $content, $amount;
```

```
    public function getBillingAddress() {
        return $this->address;
    }
    public function sendEmail() {
        $email = new Email();
        //obtain data
        $email->send();
    }
    public function getCustomerName() {
        return $this->name;
    }
    public function getTotalAmount() {
        return $this->amount;
    }
    public function confirm() {
        if(!isset($this->name) || $this->amount<0 ||
           !isset($this->address) || !isset($this->content))
            return false;
        return true;
    }
}
```

Koodinäide 36 – Liidese defineerimine ja omistamine

Iseseisev ülesanne: Tekita uus liides, mis lisab eelnevalt defineeritud liidesele funktsionaalsust juurde nagu arve sisu, mis saadetakse kliendile.

Liideste kood on saadaval Lisa 9: „Liideste kasutamine“.

Lisa 2: Hüperlingi objekt

```
/*
Kasutamine: SmartUrl::buildLink('http://www.postimees.ee/proov', 'example.php',
'Kliki siia', 'status=print, id=834, mode=view');
Väljund HTML-is:
<a
href="http://www.postimees.ee/proov/example.php?status=print&id=834&mode=view">Klik
i siia</a>
Lisadega kasutamine: SmartUrl::buildLink('http://www.postimees.ee/proov',
'example.php', 'Kliki siia jou jou', 'status=print, id=834, mode=view', 'mainLink',
'font-size: 40px;', false);
Väljund HTML-is:
<a href="http://www.postimees.ee/proov/example.php?status=print&id=834&mode=view"
class="mainLink"  style="font-size: 40px;" >Kliki siia </a>
*/
class SmartUrl{
    private $url = '';
    private $fileName = '';
    private $text = '';
    private $parameters = array();

    private $class;
    private $style;
    private $confirmation;
    function __construct($baseURL) {
        $this->url = $baseURL;
    }
    public static function buildLink($url, $fileName, $text, $parametersList,
$class=null, $style=null, $confirmation=false){
        $smartUrl = new SmartUrl($url);
        $smartUrl->setFile($fileName);
        $smartUrl->setText($text);

        if(isset($class))
            $smartUrl->setClass($class);
        if(isset($style))
            $smartUrl->setStyle($style);
        $smartUrl->setConfirmation($confirmation);
        //parameetrite listi tulevad parameetrid kuju "status=view, mode=all,
id=123"
        $parameterPairs = explode(',', $parametersList);
        if(count($parameterPairs)>0){
            foreach($parameterPairs as $parameterPair){
                $parts = explode('=', $parameterPair);
                $key   = trim($parts[0]);
                $value = trim($parts[1]);
                $smartUrl->addParameters($key, $value);
            }
        }
        return $smartUrl->renderLink();
    }
    public function setClass($class){
        $this->class = $class;
    }
    public function setStyle($style){
        $this->style = $style;
    }
    public function setConfirmation($confirmation){
        $this->confirmation = $confirmation;
    }
    public function setText($text){
        $this->text = $text;
    }
    public function getText(){
```

```
        return $this->text;
    }
    public function setFile($fileName){
        $this->fileName = $fileName;
    }
    public function getFile(){
        return $this->fileName;
    }
    public function addParameters($parameterName, $parameterValue){
        $this->parameters[$parameterName] = $parameterValue;
    }
    public function render(){
        $r = '';
        $baseUrl = $this->url;
        $params  = $this->parameters;
        $name    = $this->fileName;
        if(trim($name)!='')
            $r .= $baseUrl.'/'.$name;
        else
            $r .= $baseUrl;
        if(count($params)>0){
            $r .= '?';
            $i = 0;
            foreach($params as $parameterKey => $parameterValue){
                if($i>=1)
                    $r .="&";
                $r.= $parameterKey .'='. $parameterValue;
                $i++;
            }
        }
        return $r;
    }
    private function renderLink(){
        $r = '';
        $url = $this->render();
        $t = $this->text;

        if(trim($t) == ''){
            $t = 'Lingi ehitamiseks puudub tekst.';
        }
        $r .= '<a href="'.$url.'"';
        if(isset($this->class))
            $r .= ' class="'.$this->class.'" ';
        if(isset($this->style))
            $r .= ' style="'.$this->style.'" ';
        if($this->confirmation)
            $r .= ' onclick="return confirm(\'Oled kindel?\')"';
        $r .='>'.$t.'</a>';

        return $r;
    }
}
```

Lisa 3: Andmebaasi objekt

```
class MySQLDatabase {

    private $connection;
    public  $last_query;
    private $magic_quotes_active;
    private $real_escape_string_exists;
    function __construct() {
        $this->open_connection();
        $this->magic_quotes_active = get_magic_quotes_gpc();
        $this->real_escape_string = function_exists("mysql_real_escape_string");
//olemas kui kasutatav PHP versioon >= v4.3.0
    }

    public function open_connection() {

        $this->connection = mysql_connect(DB_SERVER, DB_USER, DB_PASS);
        mysql_set_charset('utf8',$this->connection);
        if (!$this->connection) {
            die("Database connection failed: " . mysql_error());
        } else {
            $db_select = mysql_select_db(DB_NAME, $this->connection);
            if (!$db_select) {
                die("Database selection failed: " . mysql_error());
            }
        }
    }

    public function fetch_array($result_set){
        return mysql_fetch_array($result_set);
    }
    public function num_rows($result_set){
        return mysql_num_rows($result_set);
    }
    public function insert_id(){
        return mysql_insert_id($this->connection);
    }
    public function affected_rows(){
        return mysql_affected_rows($this->connection);
    }
    public function close_connection() {
        if (isset($this->connection)) {
            mysql_close($this->connection);
            unset($this->connection);
        }
    }
    public function query($sql) {
        $this->last_query = $sql;
        $result = mysql_query($sql, $this->connection);
        $this->confirm_query($result);
        return $result;
    }
    public function escape_value($value) {

        if ($this->real_escape_string_exists) { //Saadaval kui kasutatav PHP
versioon on v4.3.0 või kõrgem
            if ($this->magic_quotes_active) {
                $value = stripslashes($value);
            }
            $value = mysql_real_escape_string($value);
        } else { //Kasutatav PHP versioon on vanem v4.3.0
            if (!$this->magic_quotes_active) {
                $value = addslashes($value);
            }
```

```
        }
        return $value;
    }
    private function confirm_query($result) {
        if (!$result) {
            $output = 'Andmebaasi päring luhtus, viga: ' . mysql_error() .'<br />';
            $output .= "Käivitatud SQL lausung: ".  $this->last_query;
            die($output);
        }
    }
}
$database = new MySQLDatabase();
```

Lisa 4: Sessiooni objekt

```
class Session {

    private $logged_in = false;
    public $user_id;

    function __construct() {
        session_start();
        $this->check_login(); //kas on juba sisselogitud?
        if ($this->logged_in) {

        } else {

        }
    }

    private function check_login() {
        if (isset($_SESSION['user_id'])) {
            $this->user_id = $_SESSION['user_id'];
            $this->logged_in = true;
        } else {
            unset($this->user_id);
            $this->logged_in = false;
        }
    }

    public function login($user) {
        if ($user) {
            $this->user_id = $_SESSION['user_id'] = $user->id;
            $this->logged_in = true;
        }
    }

    public function logout() {
        unset($_SESSION['user_id']);
        unset($this->user_id);
        $this->logged_in = false;
    }

    public function is_logged_in() {
        return $this->logged_in;
    }

}

$session = new Session();
```

Lisa 5: CAPTCHA objekt

```
<?php
class CAPTCHA {

    private $integer_one;
    private $integer_two;
    private $counter;

    function __construct() {
        session_start();
        if (isset($_SESSION['numberOne'])) {
            $this->integer_one = $_SESSION['numberOne'];
            unset($_SESSION['numberOne']);
        }
        if (isset($_SESSION['numberTwo'])) {
            $this->integer_two = $_SESSION['numberTwo'];
            unset($_SESSION['numberTwo']);
        }
        if (!isset($_SESSION['counter'])) {
            $this->counter = $_SESSION['counter'] = 0;
        }
        else{
            $this->counter = $_SESSION['counter'];
        }
        if($this->counter >=5)
            die("Oled ületanud lubatud proovimiste arvu!");
    }
    function failed(){
        $this->counter = $_SESSION['counter'] += 1;
    }
    function isInputDataCorrect() {
        if (!isset($_POST['submit']))
            return false;
        if (trim($_POST['firstName']) == '')
            return false;
        if (trim($_POST['lastName']) == '')
            return false;
        return true;
    }

    function isAnswer($answer) {
        return ($this->integer_one + $this->integer_two == $answer);
    }

    public function generate_integer_one() {
        return $this->integer_one = $_SESSION['numberOne'] = rand('1', '19');
    }

    public function generate_integer_two() {
        return $this->integer_two = $_SESSION['numberTwo'] = rand('1', '19');
    }

}
$captcha = new CAPTCHA();
?>

<?php
if($captcha->isInputDataCorrect() && $captcha->isAnswer($_POST['answer'])){
    echo 'Andmed on sisestatud korrektselt! <br />';
}
else{
$captcha->failed();
?>
    <div class="registrationForm">
```

```
        <form action="" method="post">
            <table border="0">
                <tr>
                    <td>Eesnimi: </td>
                    <td>
                        <input type="text" name="firstName" />
                    </td>
                </tr>
                <tr>
                    <td>Perenimi: </td>
                    <td>
                        <input type="text" name="lastName" />
                    </td>
                </tr>
                <tr>
                    <td><?php echo 'Kui palju on?: ' . $captcha-
>generate_integer_one() . ' + ' . $captcha->generate_integer_two();
?></td><td><input type="text" name="answer" /></td>
                </tr>
            </table>
            <input value="Vajuta" name="submit" type="submit" />
        </form>
    </div>
    <?php
}
?>
```

Lisa 6: Repaginatsiooni objekt

```
class Pagination{
    public $current_page;
    public $per_page;
    public $total_count;

    public function __construct($page=1, $per_page=20, $total_count=0) {
        $this->current_page = (int)$page;
        $this->per_page     = (int)$per_page;
        $this->total_count  = (int)$total_count;
    }
    public function offset(){
        //Oletades et lehel kuvatakse 20 sissekannet:
        //1. lehe offset on 0 (1-1)*20
        //2. lehe offset on 20 (2-1)*20
        return ($this->current_page -1) * $this->per_page;
    }
    public function total_pages(){
        return ceil($this->total_count/$this->per_page);
    }
    public function previous_page(){
        return $this->current_page-1;
    }
    public function next_page(){
        return $this->current_page+1;
    }
    public function has_previous_page(){
        return $this->previous_page() >= 1 ? true : false;
    }
    public function has_next_page(){
        return $this->next_page() <= $this->total_pages() ? true:false;
    }
}
$page = !empty($_GET['page']) ? (int)$_GET['page']:1;
//records per page
$per_page = 3;
//total record count($total_count)
$total_count = getTotalRecordsCount();

//$photos = Photograph::find_all();
$pagination = new Pagination($page, $per_page, $total_count);
$sql = "SELECT * FROM TABELINIMI ";
$sql .= "limit {$per_page} ";
$sql .= "OFFSET {$pagination->offset()}";
$photos = findAllRecords($sql);

if($pagination->total_pages()>1){
    if($pagination->has_previous_page()){
        echo ' <a href="index.php?page=';
        echo $pagination->previous_page();
        echo '">&laquo; Eelmine</a> ';
    }
}
for($i=1; $i <= $pagination->total_pages(); $i++){
    if($i == $page)
        echo ' <b>'.$i.'</b>';
    else
        echo ' <a href="index.php?page='.$i.'">'.$i.'</a>';
}
if($pagination->total_pages()>1){
    if($pagination->has_next_page()){
        echo ' <a href="index.php?page=';
        echo $pagination->next_page();
        echo '">Järgmine &raquo;</a>';
```

```
        }
    }
```

Lisa 7: Klassimeetodite paindlikum kasutamine

```
class Object {
    public static function find_all() {
        if(!isset(static::$table_name))
            die('Atribuut $table_name ei ole defineeritud või ligipääs puudub');
        return self::find_by_sql("SELECT * FROM " . static::$table_name);
    }
    public static function find_by_id($id = 0) {
        if(!isset(static::$table_name))
            die('Atribuut $table_name ei ole defineeritud või ligipääs puudub');
        $result_array = self::find_by_sql("SELECT * FROM " . static::$table_name .
" WHERE id={$id} LIMIT 1");
        return $result_array;
    }
    public static function find_by_sql($sql = "") {
        global $database;
        $result_set = $database->query($sql);
        $object_array = array();
        while ($row = $database->fetch_array($result_set))
            $object_array[] = self::instantiate($row);
        return $object_array;
    }
    private static function instantiate($record) {
        $classname = get_called_class();
        $object = new $classname($record);
        return $object;
    }
}

class Item extends Object{
    public static $table_name = '';
}
class Record extends Object{

}
Item::find_all();
```

Lisa 8: Abstraktsete klasside kasutamine

```
abstract class Weapon {

    private $nameOfTheGun;

    abstract public function fire();

    public function __construct($nameOfTheGun) {
        $this->nameOfTheGun = $nameOfTheGun;
    }
    public function getNameOfTheGun() {
        return $this->nameOfTheGun;
    }
}

class MachineGun extends Weapon {

    public function fire() {
        if ($this->remainingAmmos > 0) {
            $this->remainingAmmos--;
            return $this->remainingAmmos;
        }
        $this->needToReload = true;
    }
}

class Cannon extends Weapon {
    public function fire() {
        $this->needToReload = true;
        return $this->bullet;
    }

}
```

Lisa 9: Liideste kasutamine

```
interface Invoice {
    public function getCustomerName();
    public function getBillingAddress();
    public function getTotalAmount();
    public function sendEmail();
}
interface invoiceData extends Invoice {
    public function getContent();
}
class check implements InvoiceData {
    private $name, $address, $content, $amount;

    public function getBillingAddress() {
        return $this->address;
    }
    public function sendEmail() {
        $email = new Email();
        //obtain data
        $email->send();
    }
    public function getCustomerName() {
        return $this->name;
    }
    public function getTotalAmount() {
        return $this->amount;
    }

    public function confirm() {
        if(!isset($this->name) || $this->amount<0 ||
           !isset($this->address) || !isset($this->content))
            return false;
        return true;
    }
    public function getContent() {
        loadContentFor($this->name);
    }
}
```