TALLINNA ÜLIKOOL

Informaatika instituut

Eclipse arenduskeskkond

Seminaritöö

Koostaja: Alexey Gudz

Juhendaja: Jaagup Kippar

Tallinn 2010

Sisukord

Sissejuhatus

Tuhandeid aastaid on inimesi huvitanud efektiivsuse küsimus – “Kuidas tööd kiiremini ja mugavamalt ära teha?” Minu meelest ükskõik millise töö efektiivseks tegemiseks on kõige olulisem valida õige tööriist. Programmeerimine ei ole erand. Java lähtekoodi võiks kirjutada isegi “NotePad” redaktoris kuid see kindlasti pole aegasäästev viis. Ilmtingimata, kõige võimsam töörist on “IDE”, ehk “Integrated Development Environment”. Selline arenduskeskkond võib pakkuda erinevaid abivahendeid programmeerijale – “oma kompilaator, silur, funktsioonide loetelu, koodi värvimine ning treppimine ja isegi “WYSIWYG” režiim**,** kus võib erinevaid komponente mugavalt paigutada” on ainult osa neist. Tekib küsimus: “Millise arenduskeskkonna valida?” Neid on üsna suur valik internetis, igal on oma head ja halvad küljed. Mina hakkan rääkima ühest neist, nimelt “Eclipse IDE”st mida ise soovitan. Käesolev seminaritöö on pühendatud selle programmile.

Ma valisin Eclipse arenduskeskkonna teema, sest olen ise huvitatud sellest programmist. Arvan, et see on kõige parem töörist igale Java programmeerijale ning tahan seda tõestada lugejatele. Teine oluline põhjendus on eestikeelse Eclipse abimaterjale täielik puudumine.

Töö eesmärgiks on Eclipse keskkonna uurida, tutvustada lugejaid selle programmiga, tema eelistega ja võimalustega, näidata miks Eclipse on hea valik ning abistada algajaid startimisega.

Kõigepealt alustame ülevaatest olemasolevate abimaterjalide kohta.

1 Olemasolevate abimaterjalide ülevaade

Abimaterjale Eclipse IDE kohta ei ole raske netis leida, aga kahjuks enamus on vananenud ja eesti keeles need täiesti puuduvad.

Alljärgnevad materjalid on, minu meeles, mõned huvitavamatest ja kasulikumatest Eclipse arenduskeskkonna õppimiseks

1.1 Eclipse And Java: Free Video Tutorials

http://eclipsetutorial.sourceforge.net/ (Mark Dexter: 2010)

Tõsine ja samal ajal sõbralik ning arusaadav infoallikas kõikidele Java programmeerimisest Eclipse keskkonnas. Flash videod sisaldavad baasteadmised ja katavad veel „Workbench“, „Persistence“ ja “Debugger“ teemasid. Alternatiivselt on pakutud seal ka lugemise variant “PDF“ formaadis. Info on, kahjuks, vanema Eclipse versiooni kohta ja seda enam ei uuendata.

Kerge ja mugav variant programmi pea funktsionaalsusega tutvuda ja sobib nendele kes ei taha üle pingutada

1.2 Proffessional Eclipse 3 for Java Developers

(Berthold Daum: 2008)

Väga detailne ja sügav raamat Eclipse kohta. Autor seletab nii kõike võimalike funktsioone, nuppe ja aspekte, kui ka efektiivse programmeerimise metoodikat, tihti juhtuvaid probleeme ja muud. Raamat sisaldab piisavalt infot, et Eclipse eksperdiks saada. Selle tõttu on see allikas soovitatud nendele, kes on tõsiselt huvitatud Eclipse keskkonna põhjalikust õppimisest.

1.3 Eclipse Documentation

http://www.eclipse.org/documentation/ (Eclipse Documentation: 2010)

Ametlik dokumentatsioon, mis sisaldab kõike võimaliku informatsiooni Eclipse kohta. Info on kõige uuem ja täpsem siin, kuid see kindlasti pole hea koht esma-tutvumiseks IDE’ga. Ise soovitan kasutada seda allikat ainult mingi konkreetse asja uurimiseks.

Sobib pigem edasijõudnutele tõsisema teemade uurimiseks.

1.4 Eclipse Video Tutorial

http://jonah.cs.elon.edu/dpowell2/Courses/EclipseTutorial/EclipseTutorial.htm (Dave Powell: 2008)

Veel üks videomaterjale sisaldav allikas. Eesmärgiga tutvustada uusi Java programmeerijaid Eclipse keskkonnaga pakutakse Flash-videod koos näidistega, mis katavad olulisemad teemad nagu „Projekti loomine“, „Koodi loomine, redakteerimine ja käivitamine“, „Projektide importimine“, „Debuggimine“, „Koodi dokumenteerimine Javadoc'iga“ ja muud. Seal kasutatav Eclipse versioon on ka kahjuks vananenud.

Võrreldes Mark Dexter'I video juhendiga, see on kindlasti väiksem, kuid sisaldab kõik tähtsamad teemad. Selle tõttu arvan et see on parem valik nendele, kes tahavad kiiremini oma tööd alustada.

1.5 Eclipse IDE Tutorial

http://www.vogella.de/articles/Eclipse/article.html (Lars Vogel: 2008)

Tüüpiline “step-by-step” teksti põhinev juhend piltidega. Väga mugav lugemiseks ja täiesti arusaadav. Kuigi artikkel on lühike ja mõeldud programmiga tutvumiseks, autor annab viited oma teistele juhenditele mis katavad silumist, JSP rakenduste-, pluginate- ja RCP rakenduste loomist. Pakutakse ka paar soovitust, näiteks “sisse lülitada semikooloni genereerimist”. Info on värske ja autor uuendab seda.

Peaks sobima nendele, kes eelistavad lugeda, samm-sammult asju uurides ja kohe katsetades.

2 Installeerimine

Eclipse keskkonna installeerimine on minu arvates nii lihtne, kui see üldse võimalik on. Aga siin on üks väike nüanss – tuleb enne masinale Java Runtime Environment panna kui see pole veel olemas.

Viimase JRE versiooni võib saada siit:

“http://java.com/en/download/manual.jsp” (Java Downloads for All Operating Systems: 2008)

Kui see on tehtud, siis võib jätkata.

- Laadime viimast Eclipse versiooni, mis on hetkel 3.6 ehk “Helios”, alla siit http://www.eclipse.org/downloads/ (Eclipse Downloads: 2010). Esmatutvustuse tegemise eesmärgiks piisab baas pakendit nimega “Eclipse IDE for Java Developers”.
- Pakkime lahti arhiivi soovitud kohta (näiteks: “C:\Program Files\eclipse”).
- Käivitame “eclipse.exe” faili.
- Valime kausta kuhu kõik meie projektid ja nendega seotud failid lähevad.
- Nüüd peame nägema tervitusekraani, kus võib ka palju kasulike info saada.
- Paneme akna kinni või vajutame “Workbench” nuppu.

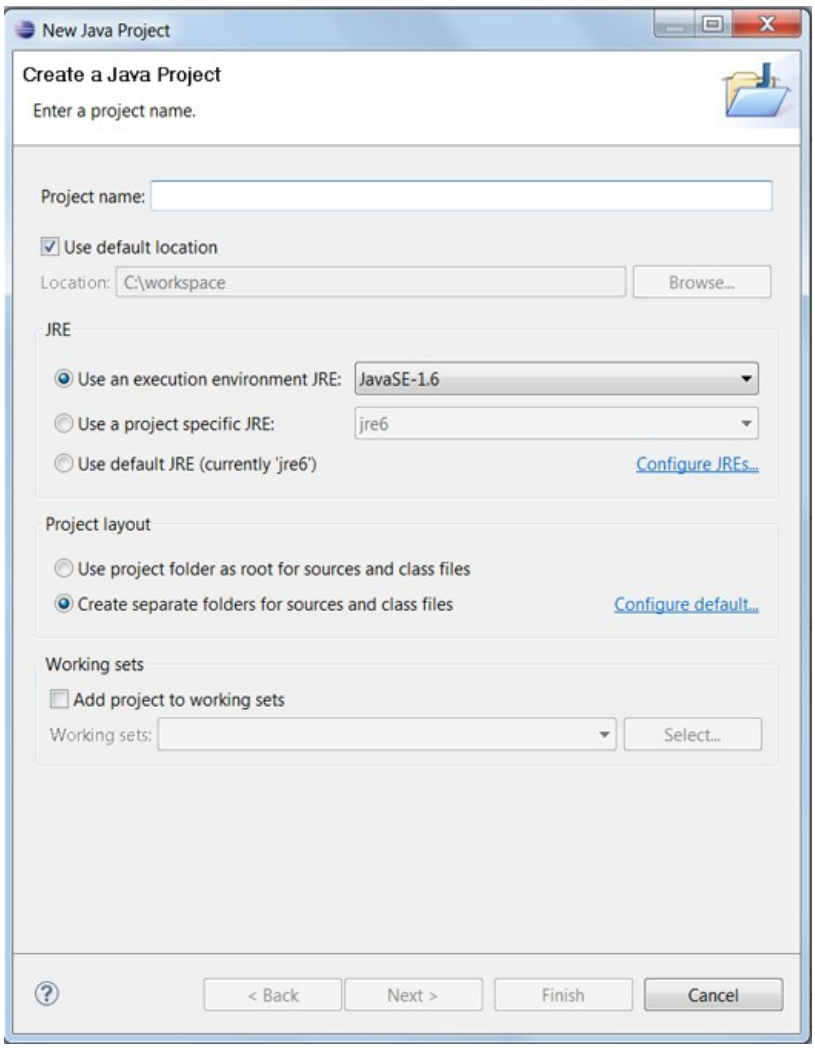
Nüüd oleme valmis kirjutada oma esimest Java programmi Eclipse keskkonnas.

3 Lihtsate programmide käivitamine

Alustame klassikalise “HelloWorld” programmi kirjutamist. Kõigepealt meil on vaja uus projekt mis käitub nagu konteiner, kuhu selle tööga seotud java failid ja muud ressursid lähevad.

File -> New -> Java Project

Peab ilmuma niisugune aken:



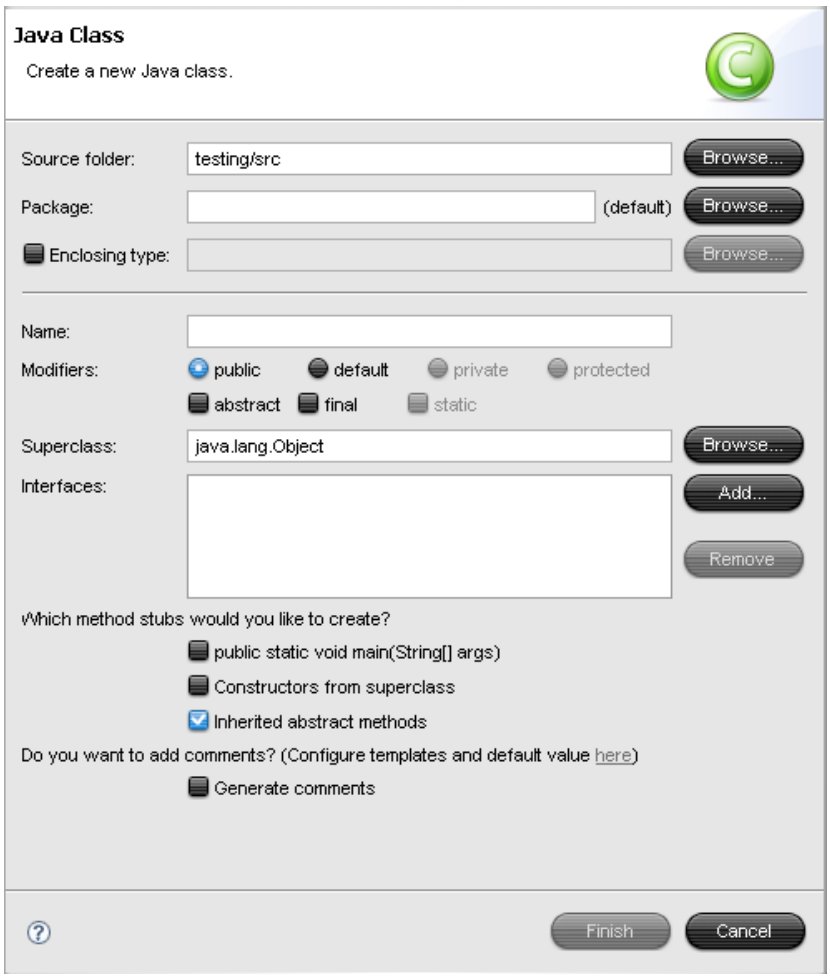
Pilt 1: Uue projekti loomine

- Project name: Nimetame meie projekti
- JRE: Kui on mingil põhjusel vaja valida teist JRE versiooni, siis seda tehakse siin.
- Project layout: Valime kas hoida kõik failid projekti kaustas või luua eraldi “bin” ja “src” kaustad (soovitatud).
- Working sets: Eclipse’is on võimalik projekte rühmitada. Siin võib valida millisse gruppi soovime selle projekti lisada.
- Kui vajutame “next”, pakutakse veel paar valikuid kus saab, näiteks, määrata teisi projekte mida meie projekt vajab. See pole meie jaoks hetkel oluline.
- Vajutame “finish”.

Siis, kui projekt on valmis, loome java klassi:

File -> New -> Class

Peab näitama sellist akent:



Pilt 2: Uue klassi loomine

- Source folder: Soovi korral võib muuta kuhu läheb lähtefail
- Package: Hea java programeerija paneb seotud java failid mingi paketti, kuid praegu meil seda pole vaja.
- Enclosing type: Võime luua meie uut klassi teise klassi sees (“Class nesting”).
- Name: Klassi nimi, paneme siia “HelloWorld”.
- Modifiers: Valime klassi sissepääs.
- Superclass: Siin võib määrata vanema klassi ehk lisada “extends”
- Interfaces: Siin võib määrata interface’id (liidesed) mida meie klass kasutab (realiseerib) ehk lisada “implements”

- Eclipse pakub meie jaoks luua veel “main” funktsiooni, konstruktorid ülemklassist ja/või abstraktsed meetodid implementeeritud "interface"idest
- Ja viimaseks, võiks lisada kommentaare, vaikimisi lisatakse ainult autori nime.

Peaks olema juba selgeks saanud, et Eclipse tihti pakub meie jaoks mingi koodi genereerida. See on üks Eclipse IDE aspektidest, mis teevad Java programmeerimise mugavamaks, aga sellest räägime hiljem.

Oleme nüüd valmis ise koodi kirjutama:

```
public class HelloWorld {
	public static void main(String[] args) {
		System.out.println("Hello World!");
	}
}
```

Käivitame:

Run -> Run

Alternatiivselt võib kasutada "CTRL + F11" kiirklahvi. Sellega tutvume natuke hiljem.

Hurraa! Meie esimene Eclipse keskkonnas loodud ja käivitatud programm!

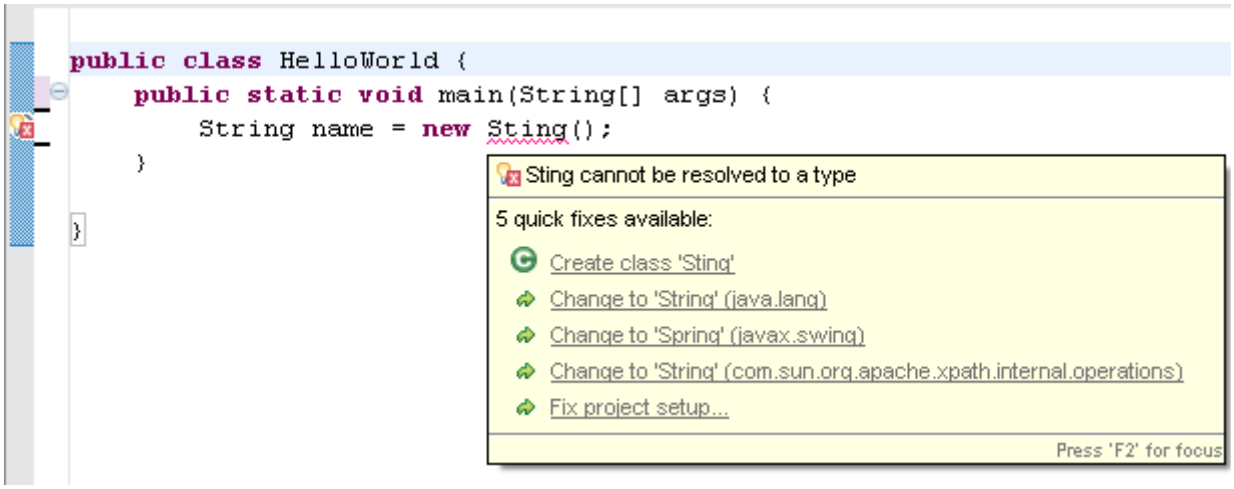
4 Eclipse eelised ja puudused

Aga miks tegelikult Eclipse kasutada tavalise tekstiredaktori ja “javac” käsureakompilaatori asemel? Kuidas ja milles on siis Eclipse parem? Proovin seda siin seletada.

4.1 Reaalaja vigade näitamine:

Programm kriipsutab alla kohad mis tekkitavad kompileerimisvead punase joonega, kuid teeb seda pidevalt reaalajas. See elimineerib vajadust mitu korda lähtefaili kompileerida ja vead otsida ehk aitab aega säästa.

Näiteks:



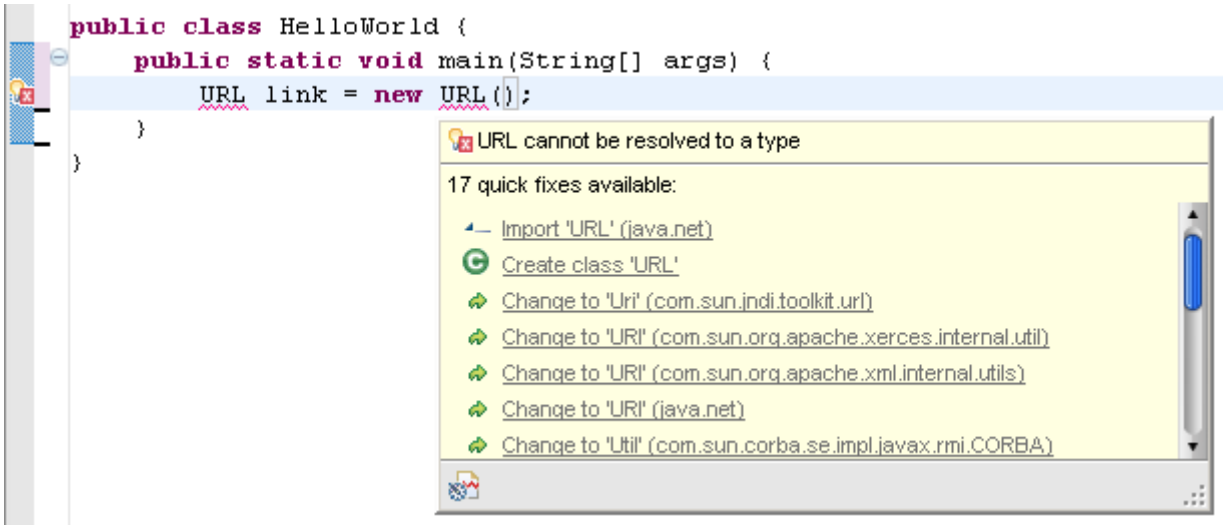
Pilt 3: Eclipse näitab vigu

Pildil võib näha, et programm nii näitab meie vigu, kui ka pakub parandamise variante.

4.2 Koodi genereerimine

Nagu oli juba eelnevalt mainitud, Eclipse tihti pakub meile kasulike koodi juppe, kas nuppude ja kiirklahvide kaudu või teisel viisil. Tegelikult, võiks öelda, et Eclipse genereerib koodi meie jaoks pidevalt, luues märke nagu ‘ “ () {} meile paaridena.

Kõige parem näide on parandamise lingid mis ilmuvad kui programm näitab vea infot reaalajas:



Pilt 4: Eclipse pakub probleemi lahendusi

“Import ‘URL’” lingi klõpsudes, importeeritakse automaatselt vajalikke pakette.

On võimalik genereerida ka plokid “try-catch”, “if”, funktsioonid nagu “toString”, “get”, “set” ja palju muid juppe.

On väga raske nimetada kõik olukorrad, kus Eclipse kirjutab koodi meie jaoks.

4.3 Autotäitmine

Ilmtingimata, see on üks nendest aspektidest mis väga kiiresti muutuvad asendamatuks.

Töötab sarnaselt “TAB”-iga “Command Prompt”is – sisestame ainult osa koodist ja vajutame CTRL + SPACE. IDE ise pakkub kõik võimalikud variandid, mis algavad sisestatud koodi fragmentiga. Juhul, kui on loogiliselt ainult üks võimalik koodilõik, programm ise sisestab seda. Näiteks, kui kirjutame “Dimension d = Toolkit.getD” ja kasutame CTRL + SPACE kombinatsiooni, programm ise kirjutab meie jaoks “Dimension d = Toolkit.getDefaultToolkit()”.

See on kasulik mitte ainult aega säästmiseks, aga aitab ka juhul, kui me oleme mingi funktsiooni nime lihtsalt unustanud.

4.4 Treppimine

Iga programmeerija teab, et kui kood kasvab suureks, muutub selle lugemine ka keerulisemaks. Et kood jääks loetavaks, on vaja teda kogu aeg treppida. See tähendab “TAB” ja võib olla “SPACE” nuppude pidev lammutamine. Meie õnneks, Eclipse hoolitseb selle eest. Ta automaatselt paneb koodi jupid õigele kohale.

4.5 Koodi värvimine

Treppimine on üks asi, mis lihtsustab suure koodi lugemist. Teine, ja minu arvates olulisem, on koodi värvimine. Sõltuvalt koodilõikude tüübist, programm värvib seda vastavalt. Võtmesõnad on märgistatud lilla värviga, kommenteeritud osa on roheline, stringid – punased, jne.

4.6 Koodi pakkimine

Veel üks väike boonus, mida arenduskeskkond pakub, on erinevate koodi osade peitmise võimalus. Programm näitab “pluss” märke erinevatel kohtadel, näiteks, esimese paketi importimine või iga funktsiooni koodi ridades. Selle klõpsudes Eclipse kas paneb koodi kokku või lahti. Tänu sellele koodi suurus väheneb.

4.7 Eclipse puudused

Tihti koos võimsusega kasvab ka keerukus. Eclipse võimaldab meil efektiivsemalt koodi kirjutada aga selleks me peame programmi valdama. Et teda põhjalikult õppida tuleb kulutada palju aega.

Erinevalt tavalisest tekstiredaktorist, Eclipse nõuab rohkem ressursse ehk tugevamat protsessorit ja operatiivmälu, aga kui arvuti ei ole 7 ja rohkem aastat vana, see ei peaks probleemi tekitama.

5 Eclipse üllatused

Neid, kes on harjunud tekstiredaktoris Java lähtefaile kirjutama ja kasutavad Eclipse arenduskeskkonna esimest korda, ootavad programmile spetsiifilised probleemid. Pole midagi rasket, tuleb lihtsalt natuke uurida. Sellega ma aitan.

5.1 Eclipse ei lase java klassi luua

Kõige esimene problemaatiline olukord millega Eclipse IDE algajad kokku puutuvad tekkib kui kasutaja proovib lisada uut java faili, aga pole mingit java projekti loodud. Iga töö peab olema organiseeritud ja Eclipse hoolitseb selle eest. Uue projekti loomisel tekitakse uued kaustad. Java failid hoitakse „workspace/projekti_nimi/src" kaustas ja kompileeritakse "bin" kausta. Järelekult, kui puuduvad konteinerid, pole kuhugi uut Java faili panna.

5.2 Main meetodile argumentide edastamine

Programmeerijad, kes kirjutavad java koodi tekstiredaktoris reeglina kompileerivad seda "javac" ja käivitavad "java" käsude abil „Command Prompt"'is või „Bash"'is. Argumendid lihtsalt lisatakse "java" käsu lõppu. Kuidas siis edastame neid arenduskeskkonnas kui kompileeritud java programmi käivitatakse nuppu vajutamisega. Tegelikult, pole midagi rasket, tuleb natuke otsida. Kui avame "Run --> Run Configurations" akent ja valime vasakul vastavat java faili, peame nägema "Arguments" sektsiooni. Paneme soovitud argumendid "Program Arguments" tekstivälja. Lihtne!

5.3 Töökataloogi leidmine

Veel üks tüüpiline situatsioon, kus tekkib segadus, on süsteemis asuva failile viitamine. Käivitades meie „.java" faili "java" käsu abil, töökataloogiks on sama kaust kus fail ise

asub. Kui, näiteks, loome seal „Pildid“ kausta ja viskame sinna „test.jpg“ pildi, õige viide failile on „Pildid/test.jpg“.

Eclipse’ga on asi teistmoodi. Java failid asuvad „bin“ kataloogis, aga töökataloogiks on tegelikult projekti kaust. Veenduda võib selle koodi abil:

```
String cd = System.getProperty(“user.dir”);
System.out.println(cd);
```

See tähendab, et kui viskame “test.txt” faili “src” kausta, viide failile on “src/test.txt”. Kui soov või vajadus tekkib, vaikimisi töökataloogi on võimalik muuta „Run --> Run Configurations --> Arguments” sektsioonis.

5.4 Mälu lõppemise vead

Oma kogemuses on tekkinud olukordi, kus Eclipse arenduskeskkond viskab “Out of memory: Heap space” vigu. Juhtub tavaliselt, kui programm vajab rohkem mälu kui on lubatud. Vältida seda võiks kasutades java virtuaalmasina “-Xms” ja “-Xmx” argumente. Kas parandame numbrid “eclipse.ini” failis programmi kaustas või loome „eclipse.exe“ otseteed ja lisame parameetri “-vmargs -Xmx512m” või mingi muud, kus “-Xms” on alg- ja “-Xmx” on maksimaalne mälu.

6 Eclipse pluginad

Paar aastat tagasi oli Eclipse peamiselt java programmeerimise arenduskeskkonnaks. Nüüd on programm muutunud uskumatult paindlikuks ja võimaldab lahendada väga erinevaid programmeerimisülesande tänu suure hulka lisa pluginaid.

Eclipse pluginad on arenduskeskkonnaga integreerivad lisandid, mis laiendavad programmi võimalused. Aastate jooksul on Eclipse ühiskond loonud umbes tuhat pluginat. Kasutades ja kombineerides neid võib luua endale ideaalse programmeerimistööriista.

Tänu Eclipse Helios versiooni MarketPlace kliendile, mida leiame „Help“ sektsioonis, pluginate installeerimine on lihtsalt paari nupu vajutamine ja juba minuti pärast on meil soovitud lisand olemas.

Kõik pluginaid on teiste inimeste töö tulemused. Oma pluginaid saame luua meie ka. Selleks on vaja uue projekti loomisel valida „Plug-in Development Plug-in

Project“. Saame oma soovitud funktsionaalsuse lisada, näiteks uut menüüd või vaadet, või olemasolevat muuta nagu tundub parem.

Igal juhul, Eclipse arenduskeskkonna paindlikkus pluginate tänu väärib lugupidamist.

6.1 Pluginate tüübid

Mõned pluginad võimaldavad meid kirjutada koodi teistes keeltes. “Pydev”: http://pydev.org/ (Pydev: 2010), näiteks, muutub Eclipse „Python“i arenduskeskkonnaks. On suhteliselt palju ka neid, mis annavad erinevate raamistikute toetust. Java “Spring IDE“: http://marketplace.eclipse.org/content/spring-ide (Spring IDE 2.3.2: 2010), javaskripti „jQueryWTP“: http://marketplace.eclipse.org/content/jquerywtp (jQueryWTP 0.36: 2010) ja „prototypeWTP“: http://marketplace.eclipse.org/content/prototypewtp (prototypeWTP 0.2: 2010), PHP „Drupal for Eclipse PDT“: http://marketplace.eclipse.org/content/drupal-eclipse-pdt (Drupal for Eclipse PDT 0.2.1: 2010) on mõned head näited.

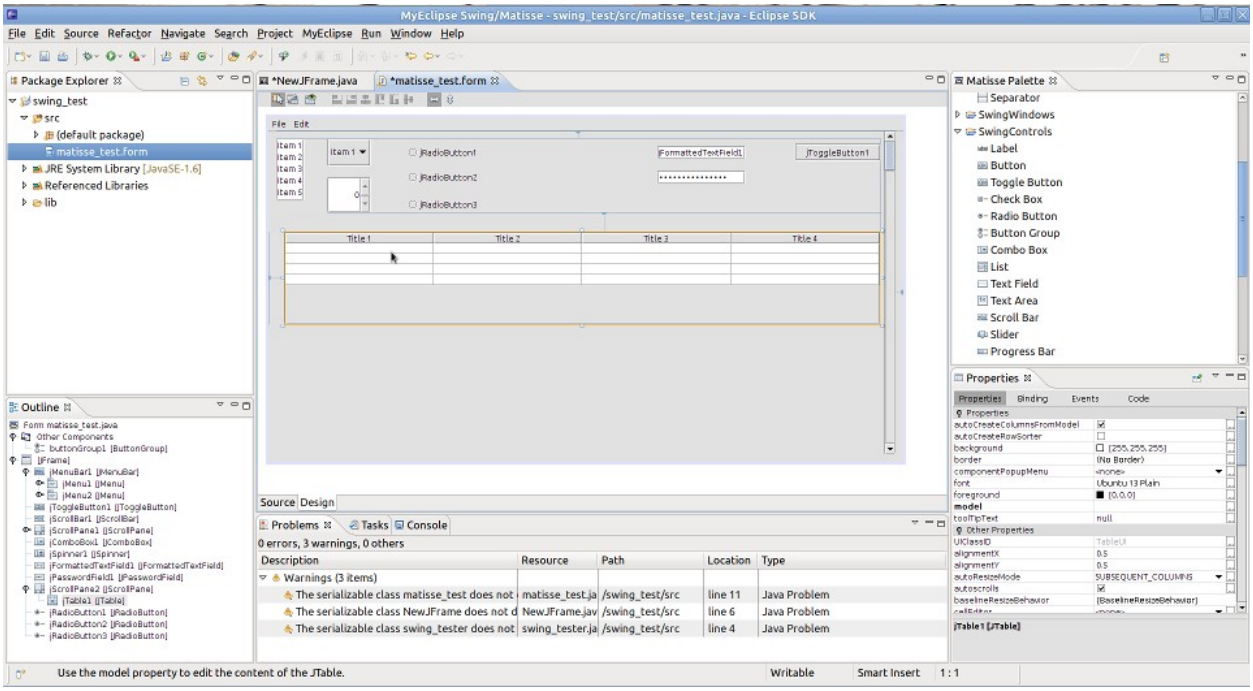
Pluginate võimed ei ole piiratud just sellega. „Subclipse“: http://subclipse.tigris.org/ (Subclipse: 2010) ja „Subversive – SVN Team Provider“: http://www.eclipse.org/subversive/ (Subversive – SVN Team Provider: 2010) annavad „Subversion“ toetust Eclipse’le. On isegi niisugused pluginad nagu “UMLet”: http://www.umlet.com/ (Free UML Tool for Fast UML Diagrams: 2010) ja “eUML2 Studio Edition”: http://marketplace.eclipse.org/node/399 (eUML2 Studio edition 3.5.0: 2010) mille abil saame Eclipse programmis UML diagramme koostada. Üks kasulikumatest, “FindBugs Eclipse Plugin”: http://marketplace.eclipse.org/content/findbugs-eclipse-plugin (FindBugs Eclipse Plugin 1.3.9: 2010), aitab leida suure hulga erinevaid defekte java koodis. Isegi nendele kes kasutavad Google WebToolkit veebirakenduste loomiseks on “Google Plugin for Eclipse”: http://code.google.com/eclipse/ (Google Plugin for Eclipse: 2010) olemas. Eclipse pluginate maailm on nii rikas.

6.2 Pluginate võrdlus

Nagu näha, samas kategoorias võib leida väga sarnaseid pluginaid. Seoses sellega proovin mõni neist ise katsetada ja omavahel võrrelda. Kasutan Linuxi Eclipse 3.5.2 versiooni.

Swing GUI Designer 8.6 vs Jigloo SWT/Swing GUI Builder 4.6

Matisse4MyEclipse, ehk Swing GUI Designer, mida saab leida aadressil http://marketplace.eclipse.org/content/swing-gui-designer (Swing GUI Designer 8.6: 2010), on Jens Eckels'i kommertslik plugin. Installeerimine on suhteliselt lihtne, samal lehel on juhised. Esimene asi mida ma märkasin, et see plugin kasutab oma "MyEclipse Swing/Matisse" perspektiivi, mida tuleb kindlasti valida. Kahjuks, oma olemasoleva klassi peale selle plugina WYSIWYG redaktorit rakendada ei saanud - on vaja luua uut eri objekti nimega "Matisse Form". Edasi asi läks ilusti — hakkasin edukalt Swing'i komponente paigutama lihtsalt vedades neid paletist "Design" sektsiooni. "Source" sektsioonis saab muidugi lähtekoodi ise kirjutada. Põhimõtteliselt sama funktsionaalsus nagu NetBeans'is.

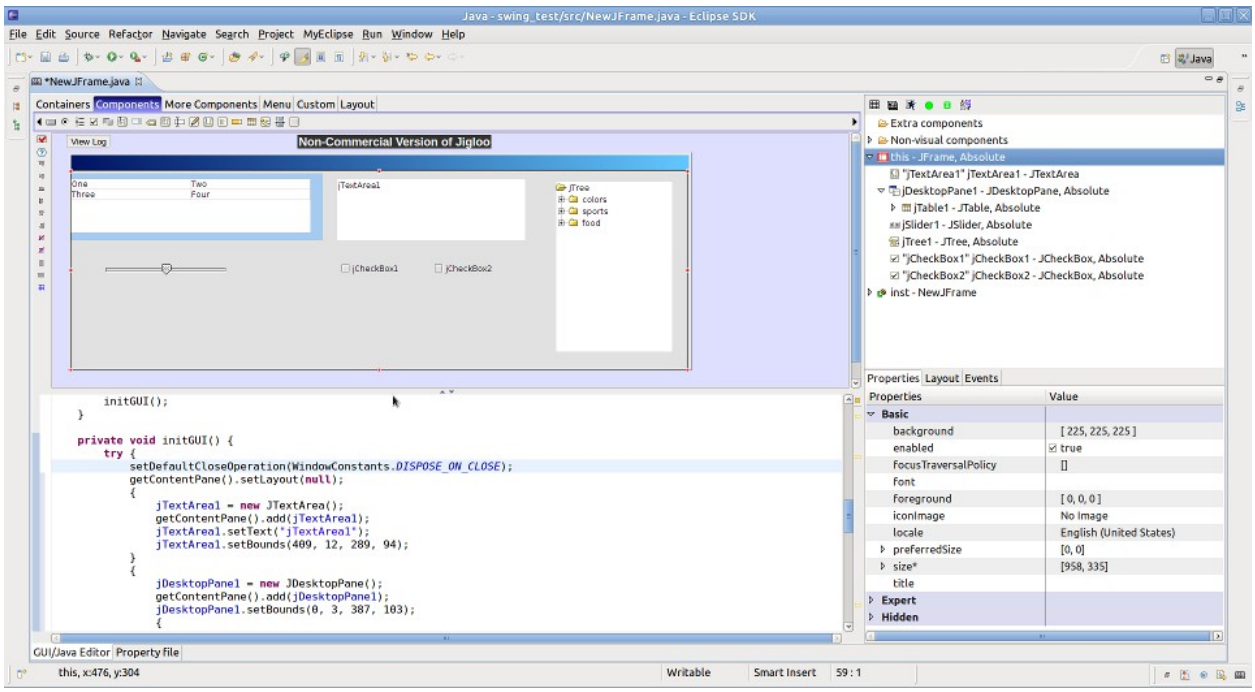


Pilt 5: Matisse4MyEclipse WYSIWEG redaktor

Kuigi NetBeans GUI builder tundub mugavama, autor ikka tegi väga head tööd selle WYSIWYG redaktori Eclipse'ga integreerimises.

Nüüd katsetan Jonathan Kinnersley' "Jigloo SWT/Swing GUI Builder", http://marketplace.eclipse.org/content/jigloo-swtswing-gui-builder (Jigloo SWT/Swing GUI Builder 4.6: 2010). Plugina tööle panek läks sama hästi. Erinevalt Matisse4MyEclipse'ist, plugin ei kasuta eri perspektiivi, aga sarnaselt tuleb luua eri objekti. Nii koodi kui ka visuaalne osa on ühel ja samal lehel — peaaken on lihtsalt jagatud pooleks. WYSIWYG funktsionaalsus jääb samaseks.

Disaini ja mugavuse mõttes, minu meeles, Jigloo kaotab. Ja veel üks miinus, mida tasub mainida, on suur ja ärritav "Non-Commercial Version of Jigloo" silt disaini osas.

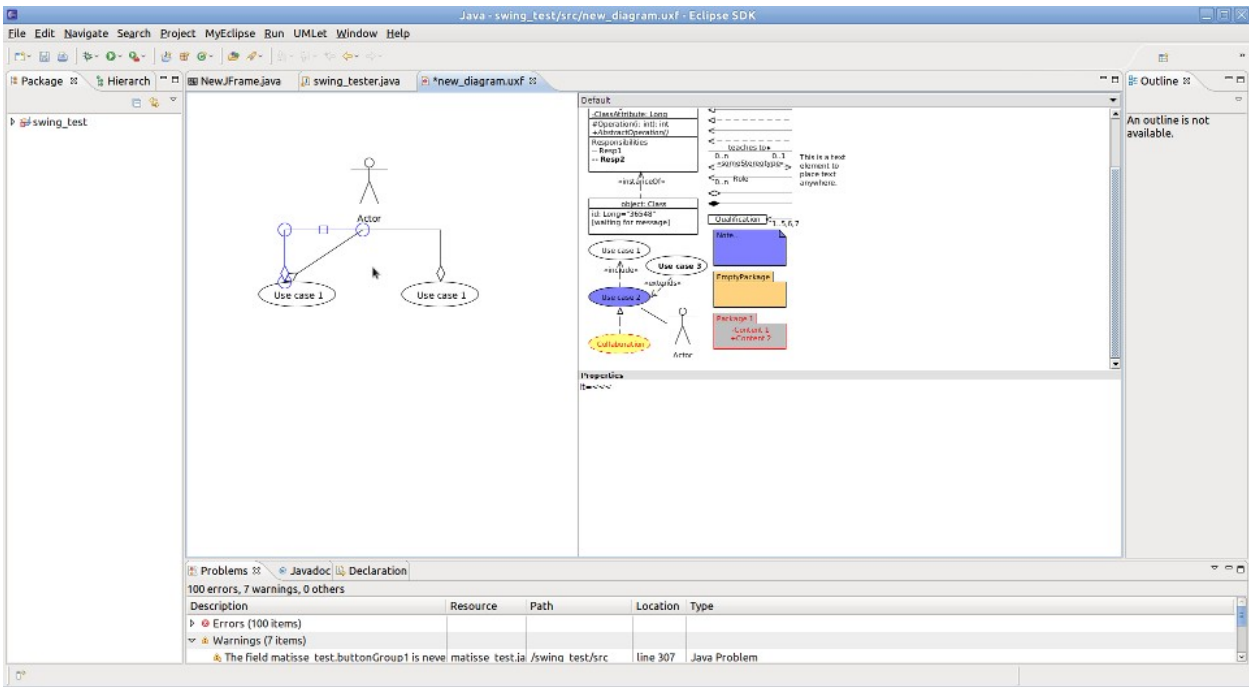


Pilt 6: Jigloo WYSIWEG redaktor

Üldiselt, mõlemad on väga head pluginad Swing komponentide kergeks loomiseks "What you see is what you get" reziimis, aga Matisse4MyEclipse oli minu jaoks mugavam.

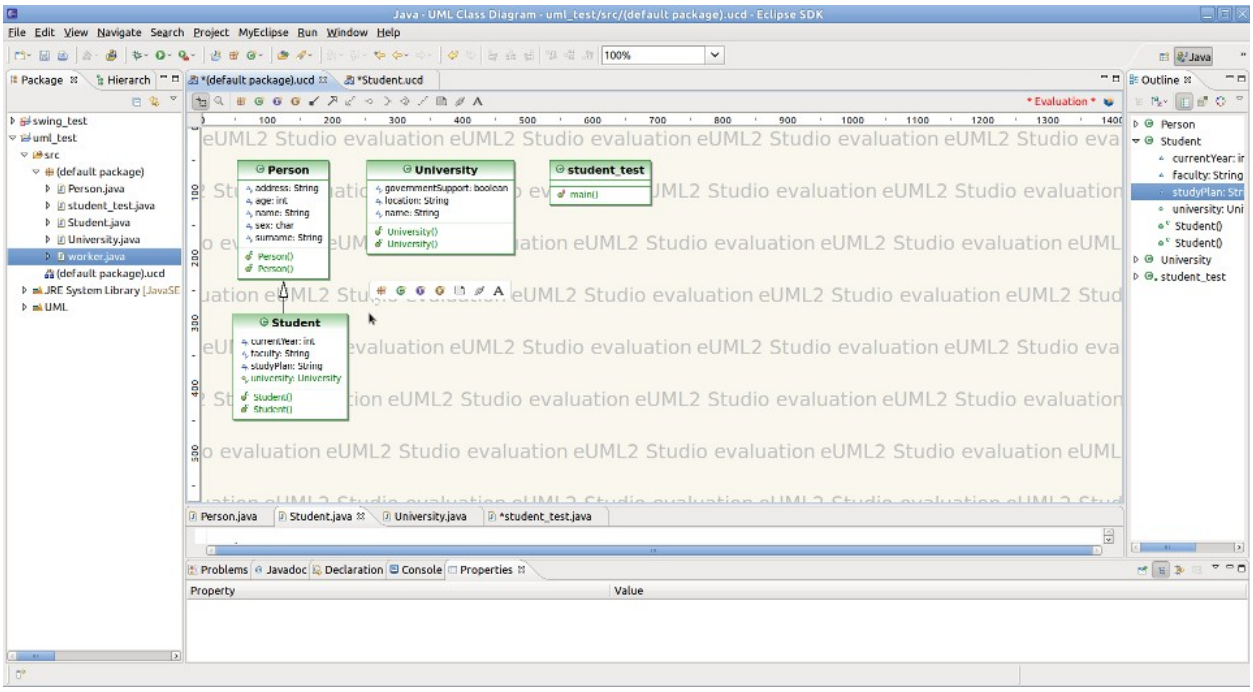
UMLet vs eUML2

"UMLet - UML Tool for Fast UML Diagrams 10.4", aadressil http://www.umlet.com (Free UML Tool for Fast UML Diagrams: 2010), on nagu nimi pakub — kiir vahend UML diagrammide koostamiseks. Installeerida seda Eclipse pluginana tuleb, kahjuks, käsitsi. Diagrammi koostamise alustamiseks on vaja luua uut "UMLet diagram" objekti. Nagu näha, see UML plugin ei pakku mingeid raskusi — keerukuse mõttes on see väga lihtne, aga saame ikkagi koostada misiganes diagrammi. Üleval saab valida diagrammi tüübi, näiteks, "Class", "Use-Case", "Sequence" või mingi muu. Tulemuse eksportida saame erinevate piltide-, "eps", või "pdf" formaadina. Minu nõuetele see plugin täiesti vastab.



Pilt 7: UMLet abil Eclipse'is diagrammide koostamine

Teiselt poolt, Soyatec eUML2, http://marketplace.eclipse.org/content/euml2-studio-edition (eUML2 Studio edition 3.5.0: 2010), pakub midagi erinevat. Installeerisin automaatselt Eclipse Updater'i kaudu ja katsetasin "Studio" versiooni. Plugin pakkus ainult 2 diagrammi tüübi — "Class" ja "Sequence". Valmistasin "Person", "Student" ja "University" üksteisega seotud klassid ja proovisin luua selle plugina "Class" diagrammi. Tulemusena nägin korraliku diagrammi mis vastab minu java klassidele.



Pilt 8: eUML2 Class diagrammi genereerimine

Plugin laseb meid ka midagi diagrammis muuta ja lisada, aga tundus mulle, et ei saa midagi kasulikku sellega teha. Oma diagrammide joonistamiseks see plugin üldse ei sobi.

UMLet on hea vahend kiirdiagrammide Eclipse keskkonnas joonistamiseks ja eUML2 — "Class" diagrammide genereerimiseks, aga mingi professionaalse UML programmiga nagu "Visual Paradigm for UML" neid, kahjuks, ei saa võrrelda.

FindBugs-eclipse-plugin 1.3.9 vs CheckStyle Plug-In 5.2.0

Valmistasin ette väikesi java koodi juppe, mis on vigu täis.

```
import java.awt.LayoutManager;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import javax.swing.JPanel;

import javax.swing.colorchooser.*;// pole kasutatud

public class analyzer_test extends JPanel {

	String test="test";

	public LayoutManager getLayot() { //viga, superklassi meetodi ülekirjutamiseks peaks nimi getLayout olema.. siin on uus meetod

		return this.getLayout();

	}

	public static void main(String[] args) {

		System.out.println(test); // viga, test muutuja ei ole static

		String test2="Hello";

		String test3="Helo";

		if(test2==test3) System.out.println("Equal");// stringid ei tohi niimodi võrrelda
```

```
            BufferedReader br=new BufferedReader(new FileReader("test.txt"));// siin võib Exception tekkida

            try {

                BufferedReader br2=new BufferedReader(new FileReader("test.txt"));

            } catch (FileNotFoundException e) {}//tühi catch blok

            int numbers[]={3, 7, 15};

            int nr = numbers[3];//numbers massiivis on ainult [0], [1] ja [2] elemendid olemas

        }

}
```

Mõni neist Eclipse näeb enne kompileerimist, mõni mitte, aga vaatame mida need koodi-analüüsimis pluginad ütlevad.

Kõigepealt proovin Lars Ködderitzsch'i "Checkstyle Plug-in 5.2.0", http://marketplace.eclipse.org/content/checkstyle-plug (Checkstyle Plug-in 5.2.0: 2010). Installeerisin selle tasuta plugini mugavalt Eclipse updater'i kaudu. Kohe sain teada, et plugin ei pakku uut perspektiivi või objekti — lisatakse ainult "Checkstyle" funktsioone popup menüüsse. Käivitades "Check Code with Check Style", plugin märgistab kõik tema meeles stilistiliselt valed kohad ja "Clear Check Style violations" kustutab neid. Pakutakse ka "Apply CheckStyle fixes" funktsiooni, mis parandab need kohad.

Minu üllatuseks, plugin genereeris mitukümmend märkust. Kuigi enamik on triviaalsed nagu "üleliigsed või puuduvad space märgid", plugin ikka teeb suurepärast tööd stilistiliselt korrektse koodi kirjutamise propageerimisega. Üks asi oli ebamugav — kontroll ei toimu reaalajas, iga kord pidin märkused kustutama ja "Check Code with Check Style" funktsiooni uuesti käivitama.

Erinevalt "CheckStyle" pluginast, Andrei Loskutov'i "FindBugs" aadressil http://marketplace.eclipse.org/content/findbugs-eclipse-plugin (FindBugs Eclipse Plugin 1.3.9: 2010) peaks olema hea lisand tõsisema defektide leidmiseks. Installeerimine oli ka automaatne. Esimesel nägemusel, see koodi analüüsimis plugin töötab sama moodi

— kontroll funktsiooni "Find Bugs" käivitame ka popup menüüst. Tegelikult, kui sain õigesti aru, selle plugina kontroll töötab täiesti teistmoodi — vaadetakse üle kompileeritud bytecode'i, mitte lähtekoodi. Enne selle java koodi kompileerimist ta ei leidnud ühtegi vigu, pärast — 3 tükki. Kaks neist olid põhimõtteliselt samad hoiatused mida Eclipse ise teeb, et mingid muutujad ei ole kasutatud, aga üks oli uus. "FindBugs" näitas tühja "catch" bloki kohal vigu "BAD_PRACTICE" kategoorias ja andis soovituse "In general, exceptions should be handled or reported in some way, or they should be thrown out of the method". Kuigi see plugin minu väikeses koodijupis palju ei leidnud, tundus see lisand ikka professionaalsem — pluginal on oma "FindBugs" perspektiiv "Bug Explorer" aknaga, kus hoitakse leitud vigu .Autor väidab, et tema plugin leiab ühe defekti iga 1000-2000 koodirea kohta, ja mina teda usaldan.

Siin, minu arvates, pole vaja nende vahel valida: esimene, "CheckStyle Plug-In", on suurepärane abi korrektse puhta koodi kirjutamise jaoks, aga teine, "FindBugs-eclipse-plugin" — suures projektis tõsisema defekti otsimiseks.

6.3 Plugina loomise katsetus

Eelnevalt mainisin oma plugina kirjutamise võimalus. Nüüd tutvume sellega natuke lähedamini – proovime kirjutada oma esimest plugina. Selle kohta on üks hea artikkel: http://www.vogella.de/articles/EclipsePlugIn/article.html (Lars Vogel: 2010).

Eclipse'le plugina kirjutamine on tegelikult üsna suur ja raske töö. Java koodi kirjutamine ei ole ainuke mure – tuleb ka õigesti "plugin.xml" faili vormistada, sõltuvused määrata ja palju muid. Kõik see vajab sügavaid Eclipse struktuuri ja API ja teadmisi. Meie õnneks pakkub Eclipse meile selleks "PDE", ehk "Plug-in Development Environment" – tööristade komplekti Eclipse sees, mis aitavad oma plugina loomisega.

Et "PDE" võimalused rakendada, tuleb luua "Plug-In Project" tavalise projekti asemel. Meile pakkutakse mitu sablooni, mille valides, saame lihtsa töötava plugina. Valime katsutamiseks "Hello, World Command". Nagu näha, Eclipse PDE genereerib kõik vajalikud failid ja näitab mugavad selgitustega vormid erinevate paraametrite muutmiseks. Java failideks on meil kaks klasse valmistatud – "Activator.java", mis vastab plugina elutsükli eest ning "SampleHandler.java", kus asub funktsionaalne kood.

Selle projekti käivitades saame Eclipse programmi, aga ühe erinevusega – peamenüüs on lisa nupp nimega "Sample Menu" kus asub "Sample Command". Selle klõpsutades saame akna sõnadega "Hello, Eclipse world". Saime täiesti töötava ja täiesti mõttetu

Eclipse plugina. Nüüd muudame seda natuke kasulikumaks – teeme sellest plugina, mis ütleb meile kui kaua oleme Eclipse programmi kasutanud.

Kõigepealt muudame selle uue menüü- ja funktsiooni nimed sobivamaks. Kõik asjad mida lisame Eclipse UI'le vormistatakse "plugin.xml" failis "extension"na. Kõik on seal juba olemas, tuleb lihtsalt "Sample Menu" etiketti asemel panna, näiteks "myTimePlugin"le ja "Sample Command" asemel - "Eclipse kasutamis aeg".

Nüüd raskem osa – funktsionaalsus. Et aru saada, palju aega Eclipse töötab, on vaja teada Eclipse programmi ning meie funktsiooni käivitamise ajad. Kuna Eclipse stardib kõikide pluginatega, meie pluginal on põhimõtteliselt samasugune startimisaeg. See tähendab, et meie "Activator" klassile tuleb kirjutada koodi juppe mis fikseeritaks aega.

Deklareerime:

```
public static Calendar cal;

public static Date startTime;
```

ning lisame "start" meetodile

```
cal = Calendar.getInstance();

startTime = cal.getTime();
```

Siin on veel üks moment millele pöördume tähelepanu - "Activator.java" peab implementeerima "org.eclipse.ui.IStartup" interface'i ja "plugin.xml" peab sisaldama selle "org.eclipse.ui.IStartup" extension'i.

Nüüd tuleb fikseerida meie "Eclipse kasutamise aeg" käsu käivitamisaja, arvutada välja kui palju aega on möödunud ning kasutajale seda näidata.

"SampleHandler.java" klassis deklareerime:

```
public Date startTime;

public Date curTime;

public String eclipseUsage;

public Calendar cal;
```

ning

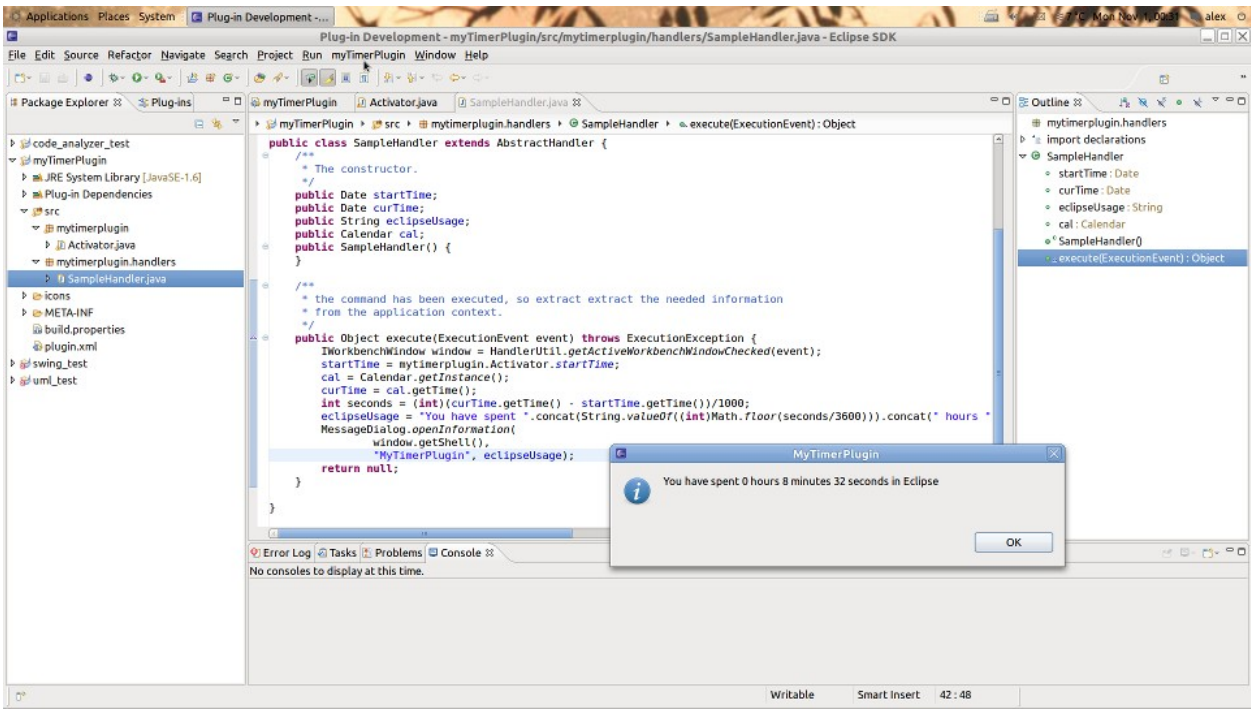
```
MessageDialog.openInformation(
        window.getShell(),
        "PluginTest",
        "Hello, Eclipse world");
```

asemel lisame "execute" meetodile:

```
startTime = mytimerplugin.Activator.startTime;
cal = Calendar.getInstance();
curTime = cal.getTime();
int seconds = (int)(curTime.getTime() - startTime.getTime())/1000;
eclipseUsage = "You have spent ".concat(String.valueOf((int)Math.floor(seconds/3600))).concat(" hours ").concat(String.valueOf((int)Math.floor(seconds % 3600)/60)).concat(" minutes ").concat(String.valueOf(seconds % 3600 % 60)).concat(" seconds in Eclipse");
MessageDialog.openInformation(
        window.getShell(),
        "MyTimerPlugin",
        eclipseUsage);
```

kus "mytimerplugin" on mul selle projekte paketi nimi. Mõlemas failis peavad ka olema "java.util.Calendar" ning "java.util.Date" paketid importitud.

Saame nüüd ööelda, et meie esimene Eclipse plugin on valmis, selle käivitamisel ja "myEclipsePlugin" --> "Eclipse kasutamis aeg" vajutades öeldakse meile mitu tunnid, minutid ning sekundid on meil Eclipse avatud.



Pilt 9: Meie myTimePlugin'a lõpptulemus

Tegelikult, meil on veel üks samm – kui soovime et teised saaksid meie pluginat ka kasutada, tuleb seda eksportida. Selleks "plugin.xml" "overview" ekraanil vajutame "export wizard", valime sobiva projekti ning paneme suvalise kausta, kus genereeritakse "plugins" kaust meie plugina "jar" failiga. Kui paneme selle "jar" faili meie või keegi teise Eclipse "plugins" kausta siis Eclipse hakkab selle plugina kasutama.

Pluginad tavaliselt lisavad mingi nupu, menü või perspektiivi, aga sellega võimalused ei ole piiratud. See oli üks väike ja lihtne plugina katsetus - kui tahame midagi tõsisemat ja kasulikumat kirjutada, tuleb Eclipse dokumentatsiooni põhjalikult uurida.

7 Kiirklahvid

Kui soovime maksimaalse efektiivsusega Eclipse IDE kasutada ja aega säästa, siis tuleb kasuks tutvuda kiirklahvidega. Kuna neid on palju, valisin ainult kõige tihedamini kasutatavaid.

CTRL+SPACE:	Autolõpetamine
CTRL+F11	Projekti käivitamine
F11	Debugger

CTRL+ /	Valitud koodi juppe kommenteerimine
CTRL+O	Kiir skeem/ülevaade
CTRL + /(jagamise märk)	Koodi laiendamine / kokkupanek
CTR+L	Konkreetsele reale minek
CTRL+I	Treppimine
CTRL+S	Säilimine
CTRL+F	Otsimine
CTRL+SHIFT+O	Puuduvate paketide importimine

Täieliku loetelu võib näha CTR+SHIFT+L kiirklahvi abil.

Kokkuvõte

Iga töö jaoks loovad inimesed mingid tööriistad või vahendid, mille abil saaks tööd kiiremini ja mugavamalt ära teha. Programmeerimisega on sama moodi. Eclipse IDE on, ilmtingimata, see asendamatu töövahend Java programmeerimises, mille abil saame kõrgemat efektiivsust saavutada.

Tänu Eclipse arenduskeskkonnale ei ole enam vaja hoida kõikide Java klasside, meetodite ja konstantide nimed mälus - Eclipse teab kõike neid meie eest. Koodi kirjutamine ja lugemine läheb palju kiiremini tänu niisuguste programmis kasutavate tehnoloogiatele nagu koodi värvimine, treppimine ja pakkimine. Erinevate koodilõikude genereerimine, kompileerimiseelsete vigade leidmine, automaatne täitmine ja muidugi kiirklahvid aitavad aega säästa. Kõige võimsam Eclipse eriomadus on tema paindlikus. Kasutades ja kombineerides soovitud pluginaid me saame ehitada endale sellise tööriista, mis sobib meile kõige paremini. Kui tekkib huvi või vajadus, saame kirjutada ja lisada Eclipse'le oma funktsionaalsust.

Muidugi on sellel arenduskeskkonnal ka oma puudused: programm on aeglasem kui tekstiredaktor, võtab piisavalt ressursse, aga kõige olulisem - nõuab suhteliselt palju õppimist. Aga, igal juhul, Eclipse on seda väärt.

Loodan, et see seminaritöö aitas programmeerimise tööriista endale valimisega, et keegi lugejatest sai selles paindlikust ja võimsast arenduskeskkonnast huvitatud, ja et siit saanud info oli kasulik programmiga töötamise alustamiseks.

Kasutatud kirjandus

1. Berhold Daum, Proffessional Eclipse 3 for Java Developers, Wrox: 2004.
2. Eclipse Documentation, URL http://www.eclipse.org/documentation/ 30.10.2010.
3. Mark Dexter, 2007, Eclipse And Java: Free Video Tutorials, URL http://eclipsetutorial.sourceforge.net/ 20.08.2010
4. Lars Vogel, 2010, Eclipse Plugin Development Tutorial, URL http://www.vogella.de/articles/EclipsePlugIn/article.html 31.10.2010.
5. Dave Powell, 2006, Eclipse Video Tutorial, URL http://jonah.cs.elon.edu/dpowell2/Courses/EclipseTutorial/EclipseTutorial.htm 20.08.2010.
6. Lars Vogel, 2010, Eclipse IDE Tutorial, URL http://www.vogella.de/articles/Eclipse/article.html 15.09.2010.
7. Eclipse MarketPlace, URL http://marketplace.eclipse.org 30.10.2010.
8. Java Downloads for All Operating Systems, URL http://java.com/en/download/manual.jsp 20.08.2010.
9. Pydev , URL http://pydev.org/ 28.10.2010.
10. Spring IDE 2.3.2, URL http://marketplace.eclipse.org/content/spring-ide 28.10.2010.
11. jQueryWTP 0.36, URL http://marketplace.eclipse.org/content/jquerywtp 28.10.2010.
12. prototypeWTP 0.2, URL http://marketplace.eclipse.org/content/prototypewtp 28.10.2010.
13. Drupal for Eclipse PDT 0.2.1, URL http://marketplace.eclipse.org/content/drupal-eclipse-pdt 28.10.2010.
14. Subclipse, URL http://subclipse.tigris.org/ 28.10.2010.

15. Subversive – SVN Team Provider, URL http://www.eclipse.org/subversive/ 28.10.2010.

16. Free UML Tool for Fast UML Diagrams, URL http://www.umlet.com/ 30.10.2010.

17. eUML2 Studio edition 3.5.0, URL http://marketplace.eclipse.org/node/399 30.10.2010.

18. FindBugs Eclipse Plugin 1.3.9, URL http://marketplace.eclipse.org/content/findbugs-eclipse-plugin 30.10.2010.

19. Google Plugin for Eclipse, URL http://code.google.com/eclipse/ 28.10.2010.

20. Swing GUI Designer 8.6, URL http://marketplace.eclipse.org/content/swing-gui-designer 30.10.2010.

21. Jigloo SWT/Swing GUI Builder 4.6, URL http://marketplace.eclipse.org/content/jigloo-swtswing-gui-builder 30.10.2010.

22. Checkstyle Plug-in 5.2.0, URL http://marketplace.eclipse.org/content/checkstyle-plug 30.10.2010.