5. domaća zadaća; JAVA, akademska godina 2013/2014; FER

Najprije pročitajte uputu na posljednje dvije stranice. Ova domaća zadaća sastoji se od dva zadatka. Čitava uputa je na hrvatskom izuzev standardnog dijela na kraju.

1. Izrada jednostavnog računalnog sustava

Sve razrede koje napišete u okviru rješenja ovog zadatka potrebno je smjestiti u odgovarajuće podpakete paketa `hr.fer.zemris.java.tecaj_2.jcomp`.

1.1 Priprema

Na Ferku u repozitoriju postoji kategorija "Dodatci domaćim zadaćama"/"Uz domaću zadaću 05" i tamo su dostupne sljedeće datoteke:

```
parser.jar
models.jar
primjeri.zip
```

Preuzmite ove datoteke. Arhiva `parser.jar` sadrži implementaciju parsera za asemblerski kôd pisan prema formatu koji je opisan u nastavku ovog dokumenta. Arhiva `models.jar` sadrži izvorne i izvršne kodove potrebnih sučelja; ako Vam Eclipse automatski ne bude pokazivao dokumentaciju za sučelja definirana u ovoj arhivi, trebate mu samo reći da su izvorni kodovi u toj istoj arhivi. Napravite novi Eclipse projekt. U njemu napravite poddirektorij `lib` te u taj direktorij iskopirajte `parser.jar` i `models.jar`. Potom ih uključite u Eclipseov *Build-Path* kako smo to napravili na jednom od prethodnih predavanja s bibliotekom `prikaz.jar`. Opremite projekt i `build.xml` datotekom. Potom tu datoteku modificirajte tako da postane svjesna postojanja direktorija `lib`. Najprije na mjestu gdje definirate varijable `src`, `build` i slično dodajte još i definiciju varijable `lib`:

```
<property name="lib" location="lib"/>
```

Potom modificirajte *classpath* koji se koristi prilikom prevođenja izvornog koda programa iz:

```
<path id="compile.path">
 <pathelement location="${build.classes}"/>
</path>
```

u

```
<path id="compile.path">
 <pathelement location="${build.classes}"/>
 <fileset dir="${lib}">
  <include name="*.jar"/>
 </fileset>
</path>
```

Ovo je potrebno kako bi prevodioc prilikom prevođenja Vašeg koda imao pristup do potrebnih `.class` datoteka koje koristite u programu a koje su smještene u jar ahive koje ste upravo smjestili u direktorij `lib`.

Datoteka `primjeri.zip` sadrži dva primjera na koja ćemo se pozvati na kraju ovog dokumenta – za sada je pričuvajte negdje sa strane.

1.2 Zadatak

Cilj ovog zadatka je upoznati se još malo bolje sa sučeljima i njihovom uporabom. Stoga ćete tijekom izrade ovog zadatka napraviti jednostavnu implementaciju "mikroprocesora" koji može izvršavati jednostavne programe. Primjerice, neki jednostavan program mogao bi izgledati ovako:

```
#Ovaj program 3 puta ispisuje "Hello world!"

            load r0, @brojac        ; ucitaj 3 u registar r0
            load r1, @nula          ; ucitaj 0 u registar r1
            load r7, @poruka        ; ucitaj poruku u r7
@petlja:    testEquals r0, r1       ; je li r0 pao na nulu?
            jumpIfTrue @gotovo      ; ako je, gotovi smo
            decrement r0            ; umanji r0
            echo r7                 ; ispisi na konzolu poruku
            jump @petlja            ; skoci natrag u petlju
@gotovo:    halt                    ; zaustavi procesor

#podaci koje koristimo u programu

@poruka:    DEFSTR "Hello world!\n" ; poruka na jednoj mem. lokaciji
@brojac:    DEFINT 3                ; broj 3 na drugoj mem. lokaciji
@nula:      DEFINT 0                ; broj 0 na trecoj mem. lokaciji
```

U okviru ovog zadatka računalni sustav i njegove komponente modelirane su kroz niz sučelja. Parser za program napisan asemblerskim jezikom prikazanim u prethodnom primjeru priložen je u JAR datoteci i ne trebate ga raditi. Parser, međutim, ne zna koje sve instrukcije postoje i kako ih treba izvršavati; ono što parser razumije je sintaksa asemblerskog jezika: što su komentari, gdje se nalazi naziv instrukcije, gdje su parametri i slično. Primjerice: pogledajte prvi redak koji sadrži naredbu `load`. Analizom tog retka, parser će zaključiti da je potrebno stvoriti naredbu koja se zove `load` te koja ima dva operanda: prvi je registar r0 a drugi je adresa memorijske lokacije (tj. cijeli broj) na kojoj se nalazi zapisana "varijabla" brojač (potražite ga u kodu i uočite da je njegova početna vrijednost postavljena na 3). Jednom kada pročita redak, prevodioc će pokušati pronaći razred koji implementira sučelje `Instruction` i koji predstavlja implementaciju ove instrukcije. Potom će pozvati konstruktor tog razreda kako bi dobio primjerak koji će predstavljati tu konkretnu instrukciju, i njega će pohraniti u memoriju mikroračunala. Vaš će zadatak, između ostaloga, biti i pisanje implementacija svih potrebnih instrukcija.

O mikroračunalu

Naše se računalo sastoji od uobičajenih dijelova: registara i memorije. Od registara, na raspolaganju su nam registri opće namjene, programsko brojilo te jedna zastavica. Za razliku od konvencionalnih procesora, naši registri opće namjene (kao i sama memorija) umjesto jednostavnih brojeva mogu pamtiti proizvoljne Java objekte. Memorija, dakle, nije niz okteta, već niz objekata. Na svaku lokaciju možemo staviti referencu na proizvoljno veliki objekt.

Zadatak 1.2.1.

U paketu `hr.fer.zemris.java.tecaj_2.jcomp` nalaze se sučelja `Computer`, `Memory` te `Registers`. Proučite ova sučelja (svako sučelje dodatno je dokumentirano) i za svako napišite jedan razred koji ih implementira. Ova sučelja dostupna su u biblioteci `models.jar` i ako ste podesili *Build-Path*, morali biste ih vidjeti u *package exploreru*. Razrede nazovite kao i samo sučelje i nadodajte `Impl` na kraju (tako će Vaš razred `ComputerImpl` implementirati sučelje `Computer`). Ove implementacije smjestite u paket `hr.fer.zemris.java.tecaj_2.jcomp.impl`. Za čuvanje registara opće namjene te za memoriju koristite obično polje. Implementacija memorije trebala bi imati konstruktor koji prima broj memorijskih lokacija (tj. veličinu memorije):

```
public MemoryImpl(int size) { ... }
```

a implementacija registara trebala bi imati konstruktor koji prima broj registara koji će procesoru biti na raspolaganju:

```
public RegistersImpl(int regsLen) { ... }
```

Jednom kada su definirani razredi potrebni za rad našeg računala, možemo pogledati kako se to računalo može koristiti. U paketu `hr.fer.zemris.java.tecaj_2.jcomp` stvorite razred `Simulator` i u njegovu metodu `main` stavite sljedeći kôd.

```
// Stvori računalo s 256 memorijskih lokacija i 16 registara
Computer comp = new ComputerImpl(256, 16);

// Stvori objekt koji zna stvarati primjerke instrukcija
InstructionCreator creator = new InstructionCreatorImpl(
      "hr.fer.zemris.java.tecaj_2.jcomp.impl.instructions"
);

// Napuni memoriju računala programom iz datoteke; instrukcije stvaraj
// uporabom predanog objekta za stvaranje instrukcija
ProgramParser.parse(
   "examples/asmProgram1.txt",
   comp,
   creator
);

// Stvori izvršnu jedinicu
ExecutionUnit exec = new ExecutionUnitImpl();

// Izvedi program
exec.go(comp);
```

Prethodni primjer sastoji se od nekoliko cjelina:

1. stvaranje novog računala,
2. stvaranje objekta pomoću kojeg će parser stvarati primjerke instrukcija,
3. prevođenje programa zapisanog u tekstualnoj datoteci i njegovo punjenje u memoriju računala,
4. stvaranje izvršne jedinice (tj. sklopa koji će izvoditi program) te
5. pokretanja izvršne jedinice.

Ako ste riješili zadatak 1.2.1, prva cjelina će raditi. U drugoj liniji poziva se parser iz JAR datoteke čiji je zadatak obraditi datoteku s programom. Kako sam parser ne zna na koji način stvoriti odgovarajuću instrukciju, potrebna mu je pomoć. Tu u igru ulazi razred `InstructionCreatorImpl` čija Vam je implementacija također unaprijed dana i dostupna je u istom paketu u kojem se nalazi i sam parser. Zadaća ovog razreda je stvoriti primjerak razred koji implementira instrukciju koju je parser pronašao u kodu. Parser s ovim razredom može razgovarati jer sam razred implementira sučelje `InstructionCreator` (pogledajte izvorni kod tog sučelja). Svaki puta kada parser otkrije neku instrukciju, poziva metodu:

```
public Instruction getInstruction(
  String name, List<InstructionArgument> arguments
);
```

Prvi argument je naziv pronađene instrukcije a kao drugi argument se predaje lista argumenata instrukcije koje je parser pronašao u datoteci. Npr. kod obrade retka u kojem piše:

```
      mul r0, r1, r2
```

ime će biti "`mul`", a lista će sadržavati tri objekta (za svaki registar po jedan; za detalje pogledati izvorni kod sučelja `InstructionArgument`).

Kako bi Vam se olakšao rad, uporabom ponuđenog razreda `InstructionCreatorImpl` dovoljno (*i nužno*) je razred koji implementira svaku instrukciju smjestiti u paket koji se predaje u konstruktoru od korištenog razreda `InstructionCreatorImpl`. Pri tome sama implementacija instrukcije mora imati jedan javni konstruktor koji prima listu argumenata (drugi argument metode `getInstruction`). Primjerice, instrukcija `mul r0, r1, r2` koja uzima sadržaje registara `r1` i `r2`, množi ih i rezultat pohranjuje u `r0` može se implementirati ovako:

```
package hr.fer.zemris.java.tecaj_2.jcomp.impl.instructions;

import java.util.List;
import hr.fer.zemris.java.tecaj_3.dz2.Computer;
import hr.fer.zemris.java.tecaj_3.dz2.Instruction;
import hr.fer.zemris.java.tecaj_3.dz2.InstructionArgument;

public class InstrMul implements Instruction {
	private int indexRegistra1;
	private int indexRegistra2;
	private int indexRegistra3;

	public InstrMul(List<InstructionArgument> arguments) {
		if(arguments.size()!=3) {
			throw new IllegalArgumentException("Expected 3 arguments!");
		}
		if(!arguments.get(0).isRegister()) {
			throw new IllegalArgumentException("Type mismatch for argument 0!");
		}
		if(!arguments.get(1).isRegister()) {
			throw new IllegalArgumentException("Type mismatch for argument 1!");
		}
		if(!arguments.get(2).isRegister()) {
			throw new IllegalArgumentException("Type mismatch for argument 2!");
		}
		this.indexRegistra1 = ((Integer)arguments.get(0).getValue()).intValue();
		this.indexRegistra2 = ((Integer)arguments.get(1).getValue()).intValue();
		this.indexRegistra3 = ((Integer)arguments.get(2).getValue()).intValue();
	}

	public boolean execute(Computer computer) {
		Object value1 = computer.getRegisters().getRegisterValue(indexRegistra2);
		Object value2 = computer.getRegisters().getRegisterValue(indexRegistra3);
		computer.getRegisters().setRegisterValue(
				indexRegistra1,
				Integer.valueOf(
					((Integer)value1).intValue() * ((Integer)value2).intValue()
				)
		);
		return false;
	}
}
```

Zadatak 1.2.2.

Napišite implementacije instrukcija:

`load rX, memorijskaAdresa`

koja uzima sadržaj memorijske lokacije (dobit će to kao broj u drugom argumentu) i pohranjuje taj sadržaj u registar `rX` (index će dobiti kao broj u prvom argumentu),

`echo rX`

koja uzima sadržaj registra `rX` i ispisuje ga na ekran (pozivom metode `System.out.print()`), te

`halt`

koja zaustavlja rad procesora.

Zadatak 1.2.3

Napišite razred koji implementira sučelje `ExecutionUnit`. Za pseudo-kod pogledajte u izvorni kod tog sučelja. Modificirajte razred `Simulator` tako da kao jedini argument naredbenog retka prima stazu do datoteke s asemblerskim kodom programa koji je potrebno prevesti i pokrenuti. Prilagodite metodu `main` tako koristi tu datoteku, a ne onu koja je bila "hardkodirana".

Stvorite datoteku `examples/prim1.txt` sljedećeg sadržaja:

```
        load r7, @poruka    ; ucitaj poruku u r7
        echo r7             ; ispisi na konzolu poruku
        halt                ; zaustavi procesor

@poruka:  DEFSTR "Hello world!\n"
```

Prilikom prevođenja ovog programa, parser će u memoriju na lokaciju 0 pohraniti instrukciju `load` (odnosno primjerak vašeg razreda `InstrLoad`), na lokaciju 1 instrukciju `echo`, na lokaciju 2 instrukciju `halt` te na lokaciju 3 string `"Hello world!\n"` (za ovo posljednje je zaslužna direktiva `DEFSTR` – *define string*; integeri se u memoriju pohranjuju s `DEFINT`). Direktive `DEFSTR` i `DEFINT` ne pišete Vi – to parser sam zna obaviti. Također, kod instrukcija koje primaju memorijsku lokaciju, u argumentima Vašeg konstruktora parser Vam neće dati ime te lokacije (tipa `@poruka`) već ćete dobiti stvarnu lokaciju (broj).

Sada biste trebali moći pokrenuti program `Simulator`, i na ekranu dobiti ispis `"Hello world!"`. Ako nešto ne radi, sada je pravo vrijeme za otkriti u čemu je problem.

Konačni cilj

Cilj je napraviti procesor koji će moći "izvrtiti" kodove priložene u primjerima u ZIP datoteci; primjere (odnosno čitav direktorij `examples` iskopirajte u Eclipseov projekt, tako da u direktoriju projekta uz `src`, `bin` i `lib` imate i `examples`). Prvi primjer nekoliko puta ispisuje istu poruku a drugi generira dobro poznati slijed brojeva. Da bi to ostvarili, još Vam trebaju neke instrukcije koje morate ostvariti.

Zadatak 1.2.4

Instrukcija	***Opis***
`add rx, ry, rz`	`rx` ← `ry` + `rz`
`decrement rx`	`rx` ← `rx` - 1
`increment rx`	`rx` ← `rx` + 1
`jump lokacija`	`pc` ← lokacija
`jumpIfTrue lokacija`	ako je `flag=1` tada `pc` ← lokacija
`move rx, ry`	`rx` ← `ry`
`testEquals rx, ry`	postavlja zastavicu `flag` na `true` ako su sadržaji registara `rx` i `ry` isti, odnosno na `false` ako nisu.

Ostvarite ove instrukcije i provjerite rad programa oba priložena programa.

Napišite još i sljedeću instrukciju:

Instrukcija	***Opis***
`iinput lokacija`	Čita redak s tipkovnice. Sadržaj tumači kao `Integer` i njega zapisuje na zadanu memorijsku lokaciju. Dodatno postavlja zastavicu `flag` na `true` ako je sve u redu, odnosno na `false` ako konverzija nije uspjela ili je drugi problem s čitanjem. `[lokacija]` ← pročitani `Integer`

Sada u datoteku `examples/prim2.txt` napišite asemblerski kod programa čijim ćete pokretanjem dobiti ponašanje prikazano u sljedećem primjeru (korisnikovi unosi su prikazani crveno), ispisi programa crno.

```
Unesite početni broj: perica
Unos nije moguće protumačiti kao cijeli broj.
Unesite početni broj:
Unos nije moguće protumačiti kao cijeli broj.
Unesite početni broj: 3.58
Unos nije moguće protumačiti kao cijeli broj.
Unesite početni broj: -23
Sljedećih 5 brojeva je:
-22
-21
-20
-19
-18
```

2. Naredba dir

Sve razrede koje napišete u okviru rješenja ovog zadatka potrebno je smjestiti u paket `hr.fer.zemris.java.tecaj.hw5` odnosno podpakete (po vlastitom nahođenju).

Napišite program `Dir`, oslanjajući se na metode razreda `java.io.File` koji smo površno upoznali na prošlom predavanju. To je komandno-linijski program koji prima minimalno jedan argument: putanju do nekog direktorija. Zadaća programa je ispisati sadržaj tog direktorija (ne ići rekurzivno u poddirektorije!). Nakon prvog argumenta, program prima jedan ili više specifikatora koji definiraju način na koji podatke treba sortirati, informacije koje treba ispisati (i kojim redosljedom) te filtere koje treba primijeniti prije ispisa. Specifikatori koji određuju sortiranje navode se uz argument -s:X (za svaki uvjet jednom se navodi ovakav specifikator; X označava željeni način sortiranja). Potrebno je podržati sljedeće načine sortiranja.

Specifikator	**Opis**
s	Sortira se po veličini; pri tome se i za datoteke i za direktorije poziva metoda `length()` i sortira prema onome što ta metoda vrati; manja veličina ide prije.
n	Sortira se po imenu; leksikografski manje ime ide prije; opet se ne radi razlika između direktorija i datoteka.
m	Sortira se prema vremenu zadnje modifikacije (metoda `lastModified()`); ranije vrijeme je prije. Ne radi se razlika između direktorija i datoteka.
t	Sortira se prema vrsti datotečnog objekta: direktoriji su "manji" od "datoteka" odnosno dolaze prije.
l	Sortira se po duljini imena. Kraća imena dolaze prije. Ne radi se razlika između direktorija i datoteka.
e	Sortira se prema izvršivosti. Datoteke koje se mogu izvršiti (metoda `canExecute()`, ma koliko smislen i ne bio njezin rezultat na pojedinim operacijskim sustavima). Izvršive datoteke dolaze prije neizvršivih.

Ako ispred slova specifikatora dođe znak minus, poredak mora biti obrnut od onoga koji specifikator definira. Primjerice, ako damo sljedeći niz specifikatora kao argumente naredbenog retka:

```
-s:-t -s:s -s:-l
```

ispis bi trebao biti sortiran tako da su najprije sve datoteke a potom svi direktoriji; unutar istog tipa sortiranje je po veličini (manje je ranije ispisano), a unutar iste veličine, najprije su ispisani objekti s duljim imenom (pojam *ime* naravno uključuje i ekstenziju).

Specifikatori ispisa određuju koje će se informacije ispisati o svakom objektu. Navode se kao argumenti koji započinju s `-w:`. Specifikatori koje treba podržati su sljedeći:

Specifikator	**Opis**
n	Ime objekta. Ispisuje se poravnato po lijevoj strani.
t	Tip objekta. Za direktorije se ispisuje `d`, za datoteke `f`.
s	Veličina objekta (u oktetima). Ispisuje se desno poravnato.
m	Ispisuje vrijeme zadnje modifikacije formata: 2014-12-27 17:21:15. Isječak koda kojim ovo možete postići dan je u nastavku.
h	Ako je objekt skriven, ispisuje se slovo "h", inače ništa.

```
Date date = new Date(f.lastModified());
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String formatiraniDatum = sdf.format(date);
```

Primjerice, želimo li da svaki redak ispisa sadrži samo naziv objekta i tip objekta, naveli bismo:

```
-w:n -w:t
```

Želimo li da najprije bude ispisan tip objekta a potom njegov naziv, naveli bismo:

```
-w:t -w:n
```

Specificiranje filtera navodi se uporabom argumenata koji započinju s `-f:`. Potrebno je podržati sljedeće filtere.

Specifikator	**Opis**
sSIZE	Prihvaća samo objekte čija je veličina manja ili jednaka SIZE. Npr: `s1024` prihvaća objekte koji su veliki do 1 kB.
f	Prihvaća samo objekte koji su datoteke (nisu direktoriji).
lSIZE	Prihvaća samo objekte čija je duljina imena manja ili jednaka SIZE. Npr. `l5` bi odbacilo objekt imena "datoteka.txt" jer je duljina imena 12.
e	Prihvaća samo objekte čije ime ima ekstenziju. Objekt imena "README" bi bio odbijen, ali objekta imena "README.txt" bi bio prihvaćen.

Ako ispred slova filtera dođe znak minus, prihvaćaju se oni objekti koje filter ne prihvaća. Primjerice, da bismo zatražili ispis samo onih datoteka čija je veličina veća od 1000 okteta i manja ili jednaka 2000 okteta, morali bismo navesti filter:

```
-f:f -f:-s1000 -f:s2000
```

Uočite: kod navođenja više kriterija filtriranja, objekt se prihvaća samo ako zadovoljava sve filtere.

Prilikom pokretanja programa Dir, prvi argument mora biti direktorij koji se želi ispisati. Nakon toga može slijediti proizvoljna mješavina raznovrsnih specifikatora; pri obradi je važno uzeti u obzir samo relativni poredak istovrsnih specifikatora. Sljedeći poziv je OK:

```
java -cp bin hr.fer.zemris.java.tecaj.hw5.Dir -s:t -w:t -f:-l3 -w:s -w:n -f:l7 -s:-l
```

i ekvivalentan je pozivu:

```
java -cp bin hr.fer.zemris.java.tecaj.hw5.Dir -s:t -s:-l -w:t -w:s -w:n -f:-l3 -f:l7
```

ali nije ekvivalentan pozivu:

```
java -cp bin hr.fer.zemris.java.tecaj.hw5.Dir -s:-l -s:t -w:t -w:s -w:n -f:-l3 -f:l7
```

jer se u ovom posljednjem zahtjeva drugačije sortiranje.

Pokušajte koncipirati Vaše rješenje tako da glavna metoda koja dohvaća sve datoteke i radi filtriranje, sortiranje i ispis ne zna ništa o konkretnim podržanim načinima filtriranja, sortiranja te o informacijama koje je potrebno ispisati. Primjerice, želimo li dodati još jednu informaciju: ispis je li ime objekta palindrom ili ne, ne biste smjeli doći u situaciju da morate išta mijenjati osim:

1. dodavanja prikladno enkapsuliranog koda koji radi tu provjeru te
2. modificiranja koda koji obrađuje argumente naredbenog retka (on naravno mora znati da uvodite

novo slovo, i koja se funkcionalnost krije iza njega).

Metoda koja provodi cjelokupni postupak dohvaćanja svih objekata, filtriranje, sortiranje i ispis zbog ovog se zahtjeva ne smije apsolutno nikako promijeniti.

Ispis svih traženih informacija treba riješiti tako da se generira tablica minimalno potrebne širine. Primjerice, ako je traženi ispis `-w:t -w:n -w:s`, ispisana tablica s datotekama/direktorijima bi trebala izgledati ovako (na nekom fiktivnom primjeru).

```
+---+-------------+---------+
| d | bin         |       0 |
| f | program.exe | 1463174 |
| d | lib         |       0 |
| f | readme.txt  |    5712 |
| f | abc.txt     |      12 |
| f | a.bat       |     125 |
| f | ccc.bin     |    3227 |
| d | src         |       0 |
| d | build       |       0 |
+---+-------------+---------+
```

Ako prilikom pokretanja programa nije dan niti jedan specifikator ispisa, postupiti kao da je dano `-w:n`. Uočiti da je legalno više puta zatražiti ispis iste informacije. Primjerice, `-w:n -w:n` bi rezultiralo s dva stupca u kojima je ispisano ime.

Ako prilikom pokretanja programa nije dan niti jedan specifikator sortiranja, ispisati objekte onim redoslijedom kojim su dobiven od operacijskog sustava.

Ako se prilikom obrade argumenata naredbenog retka naiđe na pogrešku, to je potrebno dojaviti korisniku (a ne srušiti program uz stack-trace preko ekrana). Ako se program pokrene s objektom koji nije direktorij, to dojavite korisniku na odgovarajući način.

Important notes

Solve all of the problems in a single Eclipse project. Configure Eclipse to use two source directories: `src/main/java` for your source files and `src/test/java` for sources files of unit tests.

You are required to write the adequate number of unit tests for developed instructions in problem 1 (all but the last-one which should read from standard input). In order to do so, for each instruction your will need a functional computer. However, using it, you wont be able to determine how many times was which register read and similar information. Instead, you are encouraged to read about *mockito*:

https://code.google.com/p/mockito/

There are also many useful articles on writing tests which rely on this framework: For example, take a look at this one:

http://www.vogella.com/tutorials/Mockito/article.html

Place mockito's jar into your project's lib directory and configure ant script to use it when it runs tests. When uploading you project, you will create a ZIP which will include the content of lib directory as well.

For each instruction test, create a mock computer and mock memory (where needed) and registers (where needed). Preset registers you expect the instruction will read to some known value, execute the instruction and verify that the instruction has actually read from these registers and that the result was stored as expected (get yourself familiar with mock methods `.when(...).thenReturn(...)` and `.verify(...)`).

You must equip your project with `build.xml` script so that the project can be build from the command line. In that script, you must integrate all of the quality-checking tools which I have described in the book (by integrate, I mean have a new target for each tool so that you can call each tool separately). It should be possible to run at least the following targets: `init`, `compile`, `compile-tests`, `run-tests`, `quality`, `reports`, `clean`.

Target `quality` must run unit tests and all of the quality checks (i.e. *checkstyle*, *pmd*, *findbugs*): make it just a target which depends on all tool-targets. Unit tests must be run with the code coverage analysis. Target `reports` is a wrapper that will run all of the unit tests, the quality checks and the javadoc generation.

All of the classes in all three problems should have appropriate javadoc.

Configuring ant build script

In order to get *portable* build script, you should not define paths to additional tools directly in `build.xml` file (for example, variables such as `checkstyle.home`, `pmd.home` etc). Instead, create a new file `config.properties` in the same directory as `build.xml`. It must be a text file which defines all of the necessary home variables. Here is a sample content:

```
checkstyle.home=d:/usr/checkstyle-5.6
checkstyle.jar = checkstyle-5.6-all.jar
checkstyle.style = sun_checks_2.xml
pmd.home=d:/usr/pmd-bin-5.0.2
findbugs.home=d:/usr/findbugs-2.0.2
junit.home=d:/usr/junit-4.11
jacoco.home=d:/usr/jacoco-0.6.3
xalan.home=d:/usr/xalan-j_2_7_1
```

Then delete in `build.xml` declarations for those variables (*and only that!*) and just add the following line:

```
<property file="config.properties"/>
```

Now, *`ant`* will read file `config.properties` and will define variables as defined in that file.

Also change taskdef declaration for checkstyle into:

```
<taskdef uri="antlib:com.puppycrawl.tools.checkstyle"
   resource="checkstyletask.properties" classpath="${checkstyle.home}/${checkstyle.jar}"
/>
```

Update target "cs" so that it looks like:

```
<cs:checkstyle config="${checkstyle.home}/${checkstyle.style}" failOnViolation="false">
```

Please note. You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries which is not part of Java standard edition (Java SE) unless explicitly provided by me. This means that from this point on, you can use Java Collection Framework classes or its derivatives (moreover, I recommend it). Document your code!

In order to solve this homework, create a blank Eclipse Java Project and write your code inside. You must name your project's main directory (which is usually also the project name) `HW05-`*yourJMBAG*; for example, if your JMBAG is 0012345678, the project name and the directory name must be `HW05-0012345678`. Once you are done, export the project as a ZIP archive and upload this archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. Deadline is April 13th 2014. at 8 PM.