8. homework assignment; JAVA, Academic year 2013/2014; FER

Tekst ove zadaće dan je na hrvatskom. Prije rješavanja s adrese http://java.zemris.fer.hr/nastava/opjj skinite upravo stavljenu verziju knjige (book-2014-04-30.pdf) i proučite/isprobajte poglavlje 17 (studija slučaja). Jednom kada to savladate, na isti način pristupite rješavanju sljedećeg zadatka. Za dani primjer u knjizi izvorni kodovi su dostupni na webu – ne morate sve pretipkavati (komentar je na kraju poglavlja).

Zadatak 1.

Zamislite trenutak u kojem ste gotovi s domaćom zadaćom. Pokrećete WinRAR, generirate datoteku zadaca.RAR, odlazite na Ferka, uploadate datoteku i umjesto da Vas recenzent sruši jer niste uploadali ZIP, sustav Vam javlja poruku: "Format datoteke je pogrešan – upload se ne prihvaća.". Srećom, ostalo Vam je dovoljno vremena, prepakiravate zadaću u ZIP i ponavljate upload, samo da bi Vas sustav obavijestio kako Vam u ZIP-u nedostaje datoteka "build.xml", ili pak direktorij "src/main/java" jer ste previdom ostavili defaultne postavlje Eclipse-a koji je stvorio samo direktorij "src".

U okviru ovog zadatka napravit ćete biblioteku koja radi upravo to: prima tekstovni opis provjera (tj. naš "program"), primjerak razreda `File` koji predstavlja uploadanu arhivu te ime koje je prilikom uploada datoteka imala na korisnikovom računalu. Izvorno ime dobit ćete kao poseban argument jer će u tipičnom scenariju primjerak razreda `File` koji predstavlja uploadanu datoteku biti zapravo neka privremena datoteka u koju je web-poslužitelj pohranio ono što je korisnik poslao kako bi potom na poslužitelju odgovarajući dio koda odlučio što će s time napraviti.

Kompletan program koji bi trebao raditi dostupan je kao zasebna datoteka uz ovaj dokument (u Ferku u repozitoriju) pa ga najprije proučite (`provjere.txt`).

Sve što slijedi u nastavku pretpostavlja da su iz programa uklonjeni svi komentari i prazni retci te da su svi preostali retci trimani.

U datoteci s programom, svaka je naredba dana u jednom retku. Postoje tri vrste naredbi.

1. Naredbe koje prekidaju izvođenje programa: **terminate**.
2. Naredbe koje definiraju sadržaj pomoćnih varijabli: **def**.
3. Naredbe koje rade neku provjeru: **exists**, **filename**, **format**, **fail**.

Naredba **def** prima dva argumenta: naziv varijable te sadržaj varijable. Naziv može biti proizvoljan tekst koji se sastoji isključivo od slova, brojeva, podvlaka te točki. Sadržaj je uvijek zadan kao string (napisan je pod navodnicima, ne postoje nikakve escape-sekvence ali radi mehanizam supstitucije varijabli). Njegova semantika je uvijek ista: sadržaj varijable koja se definira naredbom **def** predstavlja stazu (putanju) pa se interno može pamtiti nekom prikladnijom strukturom a ne nužno kao jedan veliki String.

Naredbe koje rade neku provjeru imaju općeniti format kako je prikazano u nastavku:

```
imeNaredbe argument1 ... argumentn @"Poruka neuspjeha" {
  naredba
  ...
  naredba
}
```

pri čemu su posljednja dva elementa opcionalna (poruka pogreške i blok naredbi definiran unutar {}) a ako su prisutni, može biti prisutan samo jedan od njih. Poruku neuspjeha lako je razlikovati od argumenata jer niti jedan argument ne može biti string koji je uveden znakom "@"). Blok naredbi, ako postoji, započinje vitičastom zagradom u istom retku u kojem je i naredba, a završava vitičastom zagradom koja mora biti jedina u retku i koja je s njome uparena (moguće je da u bloku opet postoji naredba koja otvara svoj blok pa treba pripaziti što zatvara što). Argumenti mogu biti razdvojeni jednim ili više razmaka ili tabova.

Interno, podsustav koji izvodi program treba održavati registar (tj. kolekciju) prijavljenih pogrešaka. Svaku naredbu iz ove porodice naredbi treba smatrati jednim testom. Ako je test uspješan, naredba ne radi ništa osim ako nema definiran blok naredbi – u tom slučaju se izvodi taj blok naredbi.

Ako je test neuspješan, ako postoji definirana poruka o pogrešci, tada se ona dodaje u kolekciju prijavljenih pogrešaka i prelazi se na sljedeću naredbu. Ako naredba ima definiran blok naredbi, tada se neuspjeh ignorira, ništa se ne prijavljuje (i blok se naravno ne izvodi) i ide se na sljedeću nareebu. Ako ne postoji definirana poruka o pogrešci niti blok naredbi, sustav sam u kolekciju poruka o pogreškama dodaje neku generičku poruku (ali smislenu) i opet prelazi na izvođenje sljedeće naredbe.

Ponašanje se, dakle, razlikuje ovisno o tome je li test uspio ili nije; ako je, gleda se postoji li pridruženi definirani blok naredbi pa se on izvodi a ako test nije uspio, gleda se postoji li definirana poruka o pogrešci.

Ispred svakog testa moguće je dodati znak ! koji mu invertira značenje.

Naredba **exists** prima dva argumenta: vrstu objekta koji traži te stazu objekta. Vrsta može biti "f", "fi", "fil", "file" (odnosi se na datoteku) odnosno "d", "di", "dir" (odnosi se na direktorij). Sve ostalo predstavlja pogrešku.

Evo nekoliko primjera:

```
exists f "build.xml"
```

- provjerava postoji li u arhivi u korijenskom direktoriju datoteka `build.xml`. Ako postoji, ne radi ništa, ako ne postoji, registrira pogrešku s nekim generičkim tekstom (tipa: "Datoteka build.xml nije prisutna.").

```
!exists d "bin"
```

- provjerava da ne postoji u arhivi u korijenskom direktoriju direktorij `bin`. Ako ne postoji, ne radi ništa, ako postoji, registrira pogrešku s nekim generičkim tekstom (tipa: "Direktorij bin je pronađen a ne bi smio biti.").

```
exists d "bin" {
  fail "Jao, jao! Nismo li rekli da bin se smijete pakirati?"
}
```

- ako direktorij *bin* postoji, krenut će se u izvođenje definiranog bloka; ako ne postoji, ništa se ne prijavljuje i ide se na sljedeću naredbu.

```
exists f "build.xml" @"Datoteka build.xml je obavezna! Bez nje: ocjena 1."
```

- ako datoteke nema, prijavljuje se zadana poruka.

```
exists d "src" @"Fali Vam direktorij src!" {
  exists d "src/main"
}
```

- provjerava postoji li direktorij *src*; ako ne postoji, ispisuje definiranu poruku; ako postoji, kreće u izvođenje zadanog bloka naredbi (u bloku je samo jedna naredba koja provjerava postoji li direktorij "*src/main*").

Naredba **filename** provjerava je li ime predane datoteke u skladu s prvim argumentom. Usporedba može biti osjetljiva na velika i mala slova ili ne mora biti, ovisno o tome kako je zadan prvi argument (vidi primjer u datoteci).

Naredba **format** provjerava je li predana datoteka zadanog formata, kako je definirano njezinim prvim argumentom. Trenutno morate podržati samo jedan format: "ZIP" ali kod morate napisati tako da je kasnije lagano dodavati nove formate.

Naredba **fail** predstavlja test koji je po definiciji neuspješan.

Supstitucija varijabli

Ako se u trenutku izvođenja programa pojavi potreba za supstitucijom varijable koja ne postoji (primjerice xyz), izvođenje skripte se treba prekinuti uz poruku "Interna pogreška: varijabla xyz ne postoji.".

Imena naredbi mogu se sastojati od slova, brojeva i podvlaka. Na početku imena je opcionalan uskličnik koji predstavlja negaciju testa. Taj uskličnik konceptualno ne pripada imenu ali ako postoji, mora biti napisan tik uz ime bez razmaka između njega i prvog znaka imena. Prvi znak imena naredbe ne smije biti brojka.

Imena varijabli mogu se sastojati od slova, brojeva, podvlaka i točki. Primjeri imena (bez navodnika): "dir", "dir1", "src.dir", "__dir". Prvi znak ne smije biti brojka.

Postoji više vrsta stringova.

- Obični: `"bla bla ovo je string"`.
- Neosjetljivi na velika i mala slova: `i"ovo je string"`.
- Poruke pogrešaka: `@"ovo je poruka pogreške"`.

Niti u jednom stringu nema mehanizma escapeanja što znači da nije moguće zapisati string koji bi u sebi imao tabove, entere, navodnike i slično. String također ne smije sadržavati vitičaste zagrade, znak dvotočke niti znak dolara, osim ako to nije mjesto na kojem se traži supstitucija varijable ili pak pretvorba paketa u stazu. Konkretno, sljedeći string je nevaljan:

"bla bla { itd."

Da bi nešto bilo supstitucija varijable, mora započeti s "${" (napisani tako jedan do drugoga) i mora završavati s "}". Ono što je unutar toga nakon trimanja praznina s početka i kraja mora biti sintaksno legalno ime varijable ili je takav string nevaljan.

"Vaš jmbag je ${ jmbag }." je u redu; jmbag je sintaksno legalan naziv varijable.
"Vaš jmbag je ${ jmb ag }." nije u redu; nakon trimanja s lijeva i desna nije preostao sintaksno legalan naziv varijable.

U stringovima se također ne smije koristiti znak backslash-a, a kao separator direktorija univerzalno se koristi znak "/".

Napomena uz definiranje varijabli: naredba **def** uvijek gazi eventualno prethodno definiranu vrijednost varijable, i sve ovako definirane varijable treba tretirati kao globalne (drugim riječima, ako ih **def** definira u nekom bloku, one su ipak vidljive od tog trenutka na dalje).

Organizacija rješenja

Detaljna organizacija koda ovog rješenja ostavlja se Vama. U nastavku će samo biti prikazan jedan primjer uporabe biblioteke koju razvijate i koji mora raditi.

Iz tog primjera možete očitati dva razreda koja će predstavljati ulaznu točku u Vašu biblioteku.

```
package hr.fer.zemris.java.filechecking.demo;

import java.io.File;
import java.util.HashMap;
import java.util.Map;
import hr.fer.zemris.java.filechecking.FCFileVerifier;
import hr.fer.zemris.java.filechecking.FCProgramChecker;

public class FCDemo {

	public static void main(String[] args) {
		File file = new File("C:/tmp/TMP12349876.tmp"); // zadaj neku ZIP arhivu
		String program = ucitaj("provjere.txt"); // učitaj program iz datoteke
		String fileName = "0012345678-DZ1.zip"; // definiraj stvarno ime arhive

		FCProgramChecker checker = new FCProgramChecker(program);
		if(checker.hasErrors()) {
			System.out.println("Predani program sadrži pogreške!");
			for(String error : checker.errors()) {
				System.out.println(error);
			}
			System.out.println("Odustajem od daljnjeg rada.");
			System.exit(0);
		}

		Map<String,Object> initialData = new HashMap<>();
		initialData.put("jmbag", "0012345678");
		initialData.put("lastName", "Perić");
		initialData.put("firstName", "Pero");

		FCFileVerifier verifier = new FCFileVerifier(
					file, fileName, program, initialData);
		if(!verifier.hasErrors()) {
			System.out.println("Yes! Uspjeh! Ovakav upload bi bio prihvaćen.");
		} else {
			System.out.println("Ups! Ovaj upload se odbija! Što nam još ne valja?");
			for(String error : verifier.errors()) {
				System.out.println(error);
			}
		}
	}

	private static String ucitaj(String fileName) {
		// Implementirajte ovo citanje!
		// Vratiti sadržaj datoteke:
		return " -- sadržaj datoteke, ne ovo -- ";
	}

}
```

Ovaj program prepravite tako da sva tri podatka prima preko argumenata komandne linije:

- args[0]: staza do zip datoteke,
- args[1]: naziv datoteke koji je bio kod korisnika
- args[2]: staza do datoteke s programom koji opisuje testove

Sav kod biblioteke treba biti u paketu `hr.fer.zemris.java.filechecking` ili u njegovim potpaketima.

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 this project.

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`, `jar`, `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 should have appropriate javadoc.

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) `HW08-`*yourJMBAG*; for example, if your JMBAG is 0012345678, the project name and the directory name must be `HW08-0012345678`. 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 May 6th 2014. at 23:59.