3. laboratorijska vježba

Zadatak 1.

Napravite novi projekt i u njega prekopirajte implementaciju razreda `SimpleHashtable` (razred je opisan u prethodnim uputama). Smjestite ovaj razred u paket `hr.fer.oop.lab3.topic1`. Ako tako niste prethodno napravili, promijenite implementaciju tako da razred `TableEntry` bude definiran kao javni statički ugniježđeni razred razreda `SimpleHashtable`. Ako još ne razumijete razliku između unutarnjih razreda i statičkih ugniježđenih razreda (nestatički vs. statički razred), razriješite sve nedoumice – pitat ćemo Vas to (opet).

Jednom kada ste napravili prethodne pripreme, modificirajte definiciju razreda `SimpleHashtable` tako da definirate da razred implementira sučelje `Iterable`, kako je prikazano u nastavku.

```
public class SimpleHashtable implements Iterable { … }
```

Zbog ove promjene u razred ćete morati dodati metodu tvornicu koja će proizvoditi iteratore koji se mogu koristiti za obilazak po svim parovima koji su trenutno pohranjeni u tablici, i to redosljedom kojim se nalaze u tablici ako se tablica prolazi od slota 0. Ideja je osigurati da možete napisati sljedeći isječak koda.

```
package hr.fer.oop.lab3.topic1;

public class Primjer {

	public static void main(String[] args) {
		// create collection:
		SimpleHashtable examMarks = new SimpleHashtable(2);

		// fill data:
		examMarks.put("Ivana", Integer.valueOf(2));
		examMarks.put("Ante", Integer.valueOf(2));
		examMarks.put("Jasna", Integer.valueOf(2));
		examMarks.put("Kristina", Integer.valueOf(5));
		examMarks.put("Ivana", Integer.valueOf(5)); // overwrites old grade for Ivana

		for(Object entry : examMarks) {
			SimpleHashtable.TableEntry pair = (SimpleHashtable.TableEntry)entry;
			System.out.printf("%s => %s%n", pair.getKey(), pair.getValue());
		}

	}
}
```

Ovaj kod trebao bi rezultirati sljedećim ispisom:

```
Ante => 2
Ivana => 5
Jasna => 2
Kristina => 5
```

Zadatak 2.

U paket `hr.fer.oop.lab3.topic1.shell` smjestite razred `MyShell`. Taj razred sadrži metodu `main` i predstavlja ljusku (slično kao Command prompt na Windowsima odnosno Bash na Linuxu) koja s korisnikom komunicira preko tipkovnice i ekrana. Jednom pokrenuta, ljuska s tipkovnice čita redak po redak i svaki redak tumači kao jednu naredbu (redak je sve što se unese do pritiska tipke ENTER). Naredba se ne može protezati na više od jednog retka.

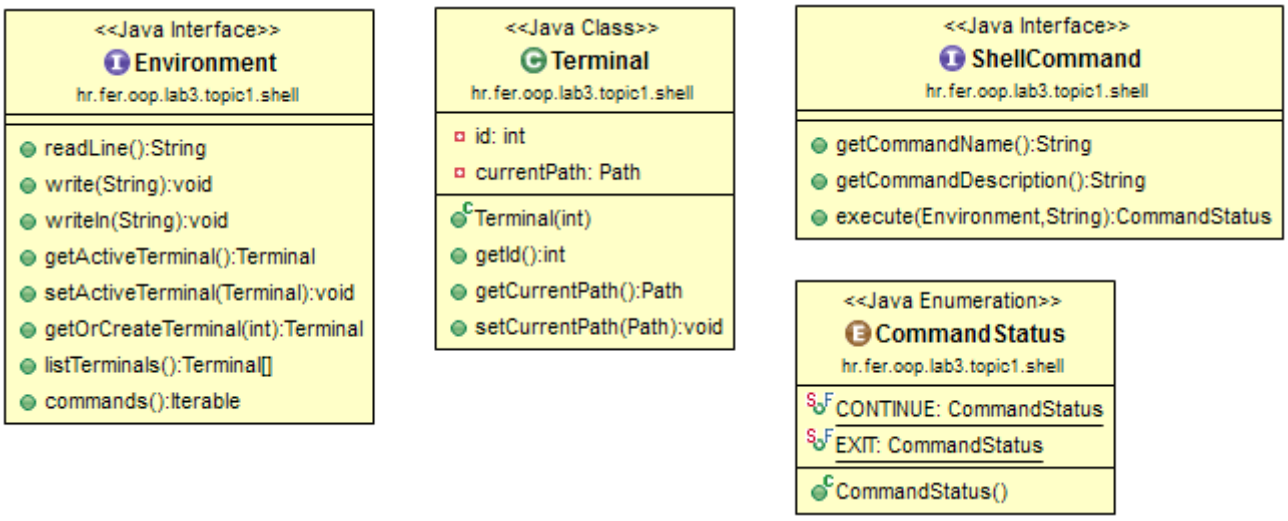
Ljuska korisniku nudi proizvoljan broj "virtualnih" terminala (ali u jednom trenutku aktivan je samo jedan, i svi terminali dijele isti ekran i tipkovnicu, pri čemu ljuska naredbe predaje aktivnom terminalu). Svaki terminal ima jedinstveni identifikator (broj) koji korisnik definira prilikom stvaranja terminala. Po pokretanju ljuske, ljuska sama stvara inicijalni terminal s rednim brojem 1. Svaki terminal pamti koji je trenutni direktorij korisnik specificirao za taj terminal. Inicijalni radni direktorij terminala je onaj u koji se razriješi `Paths.get(".")`. Prompt terminala ispisuje redni broj terminala, znak dolar ($), trenutni apsolutnu stazu trenutnog direktorija ljuske, znak veće (>) i jedan razmak. Potom korisnik unosi naredbu i pritiskom na ENTER ta se naredba izvršava.

Naredba **`terminal N`** aktivira terminal čiji je identifikator *N*. Ako takav terminal ne postoji, naredba ga stvara. Naredba **`cd staza`** kao trenutni direktorij postavlja direktorij zadan kao argument staza (to može biti i relativna staza koja se onda razrješava s obzirom na trenutnu stazu terminala). Naredba **`help`** ispisuje sve naredbe i njihov opis dok naredba **`quit`** prekida rad ljuske. Primjer rada s ljuskom prikazan je u nastavku.

```
Welcome to MyShell! You may enter commands.
1$D:\eclipse_workspaces\oop\SimpleHashtableProject> cd ..
Current directory is now set to D:\eclipse_workspaces\oop.
1$D:\eclipse_workspaces\oop> terminal 2
Changed current terminal to 2.
2$D:\eclipse_workspaces\oop\SimpleHashtableProject> cd D:\
Current directory is now set to D:\.
2$D:\> terminal 1
Changed current terminal to 1.
1$D:\eclipse_workspaces\oop> quit
Thank you for using this shell. Goodbye!
```

Po pokretanju, ljuska ispisuje pozdravnu poruku, prikazuje prompt i čeka. Aktivan je terminal 1 koji je jedini inicijalno stvoren od ljuske i prikazan je trenutni direktorij za taj terminal. Korisnik zadaje naredbu "cd .." (prikazano crveno); uočite, staza je relativna. Izvođenjem naredbe mijenja se trenutna staza terminala. Korisnik potom zadaje naredbu "terminal 2" kojom traži aktiviranje terminala 2; kako taj ne postoji, naredba ga stvara i potom aktivira (vidi se iz sljedećeg prompta). Uočite da je trenutna staza novog terminala ponovno jednaka inicijalnoj stazi – ne onoj koja je zadnja postavljena u prethodnom terminalu. Ostatak naredbi dan je kako biste po ovom primjeru istestirali radi li Vaš program dobro.

Kako su organizirani razredi? Slika u nastavku prikazuje jedan dio razreda i sučelja.

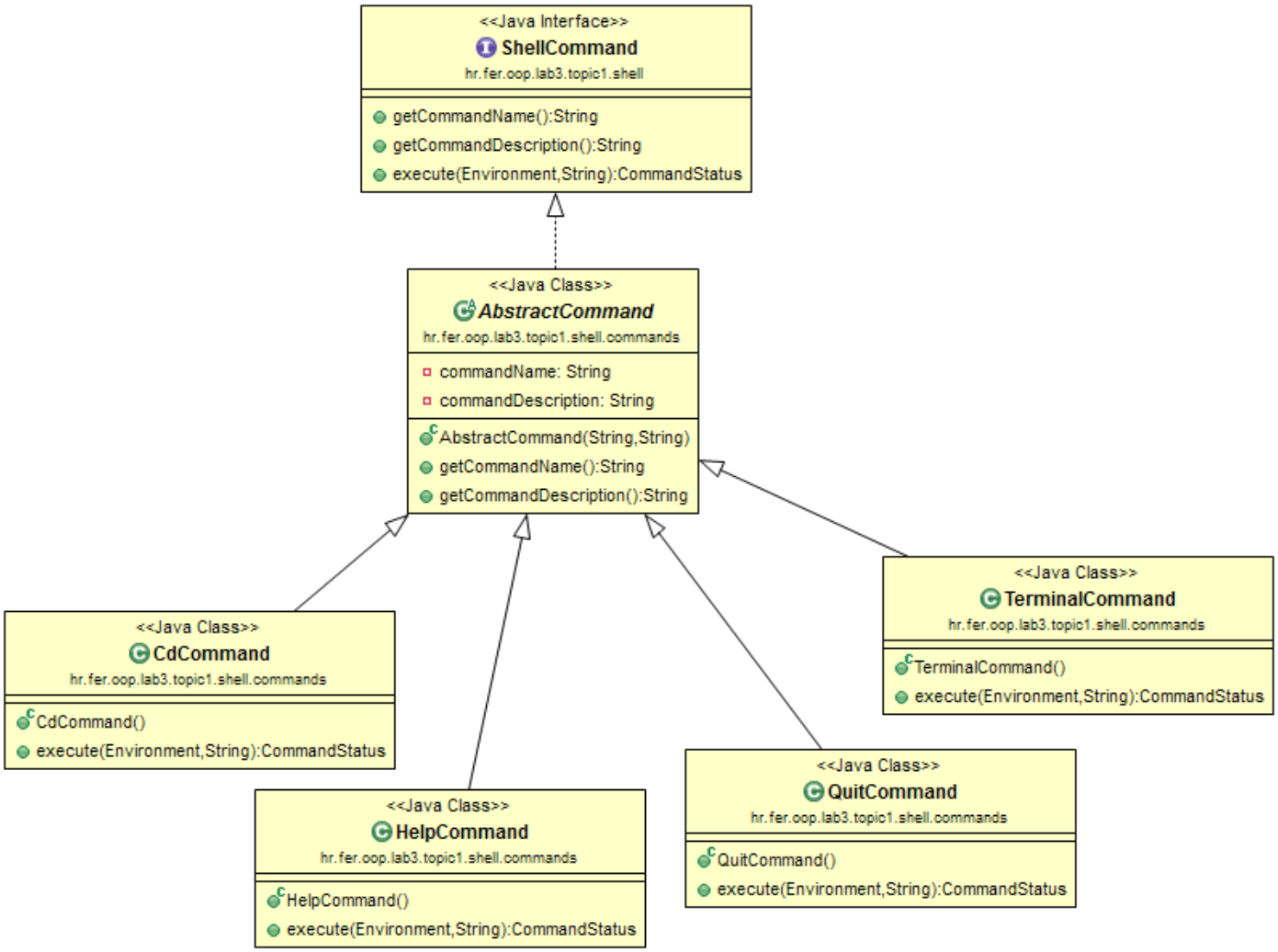


Sučelje `Environment` modelira okruženje ljuske: nudi funkcionalnost čitanja redaka s tipkovnice, zapisivanja teksta na ekran (bez automatskog dodavanja ENTERa na kraju ili s); nudi metodu za dohvat aktivnog terminala, metodu za promjenu aktivnog terminala, metodu koja dohvaća terminal prema predanom identifikatoru (i po potrebi stvara novi ako takav još ne postoji), metodu koja lista sve postojeće terminale te metodu tvornicu za Iterable objekt preko kojeg se može dobiti iterator koji obilazi po svim naredbama (objekti tipa `ShellCommand`).

Razred Terminal opisuje jedan terminal. Svaki terminal ima svoj identifikator te trenutni direktorij.

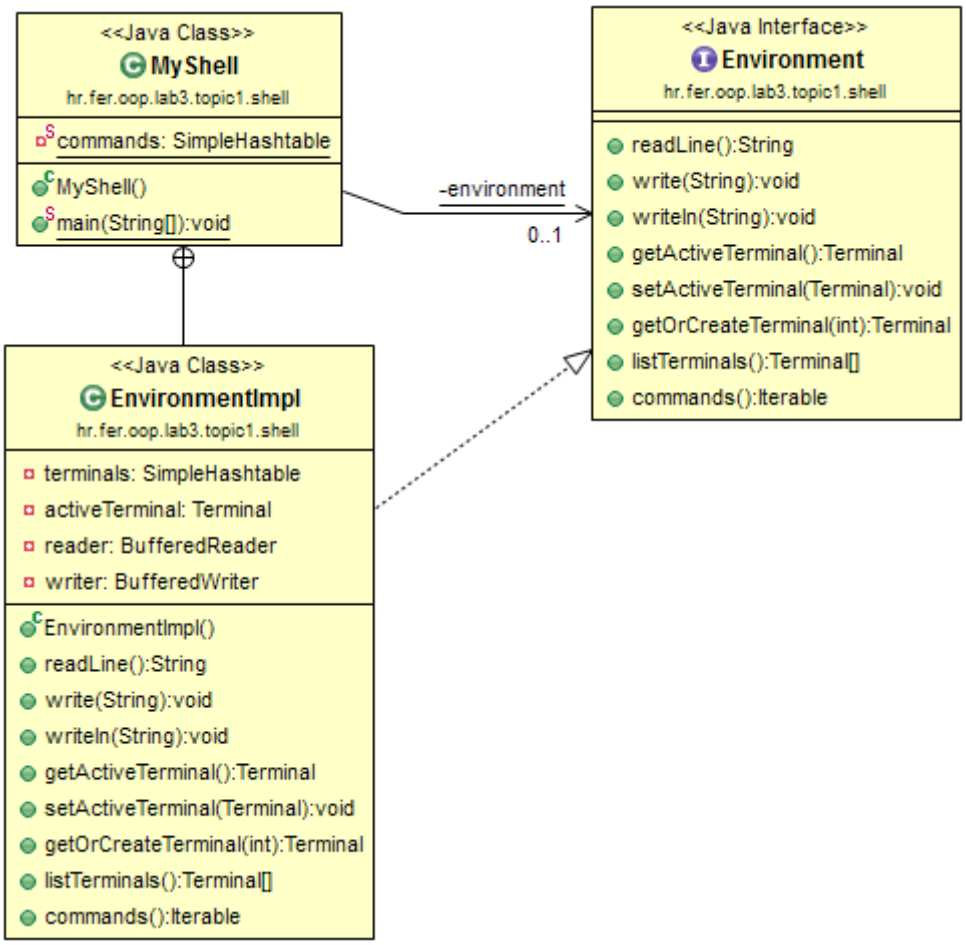
Sučelje `ShellCommand` opisuje jednu naredbu. Naredba ima ime (CD, QUIT, TERMINAL, …), opis te metodu `execute` čijim se pozivom naredba izvodi. Što treba napraviti po završetku izvođenja, naredba ljuski govori putem povratne vrijednosti koja je tipa `CommandStatus` – enumeracija pri čemu `CONTINUE` znači da ljuska treba nastaviti s izvođenjem i korisnika pitati za novu naredbu, a `EXIT` znači da ljuska treba završiti s radom. Metoda `execute` kao argumente mora dobiti referencu na okruženje u kojem se naredba izvodi (taj objekt stvara i održava sama ljuska) te string koji sadrži sve što je korisnik unio kao naredbeni redak nakon uklanjanja same naredbe (primjerice, ako je korisnik zadao "copy a.txt b.txt", drugi argument će biti "a.txt b.txt").

Hijerarhija naredbi prikazana je na slici u nastavku.



Razred `AbstractCommand` definira mehanizam čuvanja imena i opisa tako da to ne mora svaka naredba kopirati.

Konačno, pogledajmo i samu ljusku (slika u nastavku).



Ljuska je predstavljena razredom `MyShell`, koji ima statički ugniježđeni razred `EnvironmentImpl` koji implementira sučelje `Environment`. Ljuska ima statičku tablicu raspršenog adresiranja `commands` u koju su dodani po jedan primjerak svake naredbe (ključ je naziv naredbe, vrijednost je referenca na primjerak same naredbe). Da bi se osiguralo da korisnik može naredbe pisati i velikim i malim slovima (ili kombinacijom), neka ključevi budu imena naredbi pisani velikim slovima. Ljuska također ima statičku referencu na jedan primjerak okruženja (statička članska varijabla `environment`) koja se inicijalizira na primjerak razreda `EnvironmentImpl`. Iterator u tom razredu ima pristup statičkoj tablici koja čuva naredbe pa može izvesti traženi iterator.

Za inicijalizaciju naredbi ljuske iskoristite statički inicijalizacijski blok. Kostur ljuske prikazan je u nastavku.

```
package hr.fer.oop.lab3.topic1.shell;

…

public class MyShell {

	private static SimpleHashtable commands;

	static {
		commands = new SimpleHashtable();
		ShellCommand[] cc = {
			new HelpCommand(),
			new QuitCommand(),
			new CdCommand(),
			new TerminalCommand()
		};
```

```
            for(ShellCommand c : cc) {
                commands.put(c.getCommandName(), c);
            }
        }

        public static class EnvironmentImpl implements Environment { … }

        private static Environment environment = new EnvironmentImpl();

        public static void main(String[] args) throws IOException {
            environment.writeln("Welcome to MyShell! You may enter commands.");

            while(true) {
                environment.write( … prompt …);
                String line = environment.readLine();
                String cmd = naredba...;
                String arg = predani argumenti...;
                ShellCommand shellCommand = dohvati iz tablice naredbu...
                if(shellCommand==null) {
                    environment.writeln("Unknown command!");
                    continue;
                }
                izvrsi naredbu; ako vrati EXIT => break;
            }

            environment.writeln("Thank you for using this shell. Goodbye!");
        }
}
```

Zadatak 3.

Dodajte u ljusku sljedeće naredbe.

type filename
- na zaslon ispisuje sadržaj datoteke

filter img*.png
- pretražuje trenutni direktorij i sve njegove poddirektorije za datotekama čije ime odgovara zadanom uzorku; za svaku pronađenu datoteku na salon ispisuje punu stazu do te datoteke. Od zamjenskih znakova može se pojaviti znak * i to samo jednom (ali na bilo kojem mjestu: na početku, u sredini ili na kraju); taj znak je zamjena za 0 ili više proizvoljnih znakova. Pri podudaranju imena velika i mala slova se **ne razlikuju** (tj. uzorak "img*.png" je primjenjiv na "img-split.png" kao i na "ImG-Svemir.Png").

copy staza1 staza2
- kopira datoteku (i samo datoteku, ako se zada direktorij, prijaviti pogrešku) zadanu prvim argumentom u direktorij staza2 (ako je to direktorij; tada se ime datoteke preuzima iz staze1) ili pod imenom koje definira staza2 (ako je roditelj od staza2 postojeći direktorij; tada se pretpostavlja da je zadnja komponenta novo ime datoteke). Ako ne postoji staza2 niti roditelj, onda prijaviti pogrešku. Kopiranje izvesti direktno koristeći binarne tokove (**ne koristiti** pomoćne metode razreda Files).

xcopy staza1 staza2
- staza1 mora biti postojeći direktorij; staza2 ili njezin roditelj mora biti postojeći direktorij. Naredba rekurzivno kopira strukturu direktorija počev od staza1 u staza2 (ako je to postojeći direktorij) odnosno u roditelja ali pod novim imenom ako je roditelj direktorij.

Naredba xcopy zahtjeva još malo pojašnjenja. Pretpostavimo da postoji:

```
D:\
  dir1
    B.txt
    dir2
      A.txt

D:\
  dirA
```

Naredba `xcopy D:\dir1 D:\dirA` mora rezultirati ovime:

```
D:\
  dirA
    dir1
      B.txt
      dir2
        A.txt
```

dok naredba `xcopy D:\dir1 D:\dirA\dirB` mora rezultirati ovime (dir1 preimenuje u dirB pri kopiranju):

```
D:\
  dirA
    dirB
      B.txt
      dir2
        A.txt
```

Pri implementaciji ove naredbe sami ste zaduženi za fizičko kopiranje datoteka (uporabom binarnih tokova); nije dopušteno napraviti kopiranje uporabom gotovih funkcija.