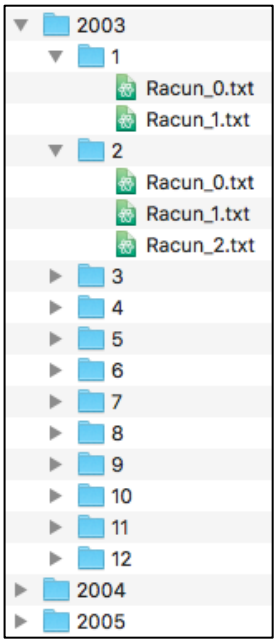
5. laboratorijska vježba (11 bodova)

Važna napomena: *u svim zadacima potrebno je napisati Javadoc komentare za svaki razred te generirati dokumentaciju. Svi nazivi razreda, metoda i varijabli moraju biti na engleskom. Sav napisani programski kod mora biti napisan u skladu s konvencijama imenovanja varijabli, metoda i razreda (varijable i metode: malo početno slovo,* camel-case*; razredi i sučelja: veliko početno slovo,* camel-case*; konstante: uobičajeno sve veliko i razdvajanje podvlakom) te ostalim pozitivnim praksama (uključivo i korektno uvlačenje redaka; smisleno razdvajanje više različitih semantički grupiranih redaka praznim redcima, pravilnim razmještajem otvorene i zatvorene vitičaste zagrade i slično). Za više informacija pogledajte* ***http://www.oracle.com/technetwork/java/codeconventions-150003.pdf***

Uvod

U ZIP-arhivi racuni.zip (http://www.fer.unizg.hr/_download/repository/racuni.zip) pohranjena je hijerarhijska datotečna struktura u kojoj se nalaze datoteke s računima. Svaka datoteka je tekstualna i strukturirana je na sličan način, a pohranjena na lokaciju kojoj odgovara vrijeme kreiranja u datoteci pohranjenog računa. Izgled strukture je:



Potrebno je dohvatiti spomenutu arhivu sa spomenute lokacije i dekomprimirati je u vršnu datoteku projekta u kojem se rješava zadatak. Time se dobiva mapa računi u kojoj su podmape imenovane po godinama, a sadrže podmape imenovane po mjesecima kada su pojedini računi kreirani. Svaka mapa za mjesec sadrži tekstualne datoteke od kojih svaka sadrži podatke jednog računa.
Svaki račun formatiran je na sličan način i slijedi sljedeći uzorak:

```
Račun br XXXXXX
Kupac: može i ne mora biti naveden
-----------------------------------------------------------------------
Naziv              cijena/komad      komada      cijena total  PDV
Artikl 1                    $/#         ###               $$$   $$
Artikl 2                    $/#         ###               $$$   $$
Artikl 3                    $/#         ###               $$$   $$
...
Artikl n                    $/#         ###               $$$   $$
```

```
                                                    ----------
UKUPNO                                                 $$$$$

Podnožje s dodatnim napomenama.
```

Datoteka sadrži znakove na hrvatskom jeziku prema UTF-8 kodiranju. Za rješavanje zadataka kreirati novi projekt (kao i dosad) te u njemu rješavati prvi zadatak u paketu `hr.fer.oop.lab5.first`, drugi u `...second`, treći u `...third`.

Zadatak 1 – Rad s tokovima podataka, FileVisitor i Decorator

U ovom zadatku potrebno je prvo pročitati sadržaj svih datoteka, a potom ga ispisati u jednu datoteku. Za čitanje koristite klase `FileInputStream`, `BufferedInputStream`. Za obilazak svih datoteka će se pisati `FileVisitor` klasa koja će obići hijerarhiju datoteka te svaku iščitati redak po redak.

Zadatak se izvodi u dva dijela: u prvom se kreira podrška za čitanje jedne datoteke i ispis te datoteke u novu (prepisivanje sadržaja), a u drugom dijelu potrebno je ostvariti prepisivanje svih tekstualnih datoteka predanog direktorija u jedinstvenu tekstualnu datoteku.

Prvi dio

Napravite klasu `MyByteWriter` koja u konstruktoru prima `InputStream` objekt s ulaznim tokom kojeg će se zapisati u novu datoteku i objekt `Path` kojim je opisana nova datoteka. `MyByteWriter` sadrži metodu

```
public void run();
```

čijim pozivom započinje rad. Implementacija metode `run()` treba biti jednostavna petlja koja čita s danog ulaza (koristiti `BufferedInputStream` dekorator i polje okteta kao *buffer*) te pročitane znakove ispisuje u zadanu datoteku. Po završetku čitanja (kada `InputStream` dođe do kraja ulaznih podataka) zatvoriti sve tokove i završiti izvođenje metode.

Kreirati klasu `SingleFileProgram` s metodom `main`. Ovdje treba nad odabranom datotekom (npr. `<root>/racuni/2015/1/Racun_0.txt`) kreirati odgovarajući `InputStream` te s tim objektom i novim objektom `Path` za izlaznu datoteku (npr. `<root>/singleout.txt`) stvoriti objekt klase `MyByteWriter`. Potom pozvati metodu `run()` i provjeriti da je kreirana očekivana datoteka.

Drugi dio

U drugom dijelu zadatka potrebno je prepisati sadržaj svih datoteka iz hijerarhije *racuni* u jednu veliku tekstualnu datoteku. Kako bi se obišlo podstablo datotečnog sustava potrebno je stvoriti klasu `MyByteReader` koja implementira sučelje `FileVisitor`. Klasa `MyByteReader` u konstruktoru prima instancu klase `OutputStream` kojeg dekorira `BufferedOutputStream`-om i postavlja kao vlastiti atribut. U metodi koja se izvršava pri posjeti datoteke potrebno je najprije provjeriti da je datoteka odgovarajućeg tipa (txt). Ako je zaista pronađena tekstualna datoteka, potrebno ju je prepisati u `BufferedOutputStream` koji je vlastiti atribut klase `MyByteReader`. To je potrebno implementirati stvaranjem `FileInputStream`-a dekoriranog `BufferedInputStream`-om nad pronađenom datotekom, kojim se čita sadržaj u polje okteta (buffer), nakon čega se iz *buffera* sadržaj zapisuje na spomenuti izlaz (`BufferedOutputStream`). Poslije prepisivanja svake posjećene datoteke potrebno je zatvoriti otvoreni `BufferedInputStream`.

Kreirati klasu `MultipleFileProgram` s metodom `main`. U `main` metodi potrebno je najprije stvoriti instancu klase `Path` koja predstavlja odredišnu tekstualnu datoteku u koju će biti prepisani svi računi. Osigurati da ta datoteka zaista i postoji (program treba stvoriti novu datoteku danog imena ako takva već ne postoji). Zatim je potrebno stvoriti još jednu instancu klase `Path` koja predstavlja direktorij u čijem će se podstablu datotečnog sustava tražiti tekstualne datoteke. Nad odredišnom datotekom potrebno je stvoriti `OutputStream` kojeg se zatim predaje kao argument pri stvaranju instance *visitora* `MyByteReader`. Stvoreni *visitor* zatim izvodi metodu `walkFileTree` kojom prepisuje sve tekstualne datoteke iz danog direktorija u jedinstvenu tekstualnu datoteku.

Zadatak 2 – Tokovi znakova i kolekcije

Ovaj zadatak rješava se korištenjem Javine podrške za tokove znakova (*character streams*). Iako se sadržaju svake datoteke može pristupiti korištenjem tokova s oktetima kao u prošlom zadatku, s obzirom da su datoteke s kojima se radi tekstualne (kodirane u UTF-8), za rad sa sadržajem datoteka lakše je koristiti klase za rad sa tokovima znakovima.

Za ilustraciju: ako je potrebno provjeriti započinje li neka linija u datoteci riječju „Riječ“ moramo znati kako je taj niz kodiran. Banalna usporedba konstante „Riječ“ pohranjene u memoriji sa oktetima iz datoteke oktet-po-oktet ne mora nužno dati točan rezultat jer se kodiranje može razlikovati.

	R	i	j	e	č
UTF-8	52	69	6A	65	C48D(**)
CP-1250 (Windows hr)(*)	52	69	6A	65	E8
UTF-32 (**)	00000052	00000069	0000006A	00000065	0000010D

(*) npr. Tekstualna datoteka kreirana u Notepad programu s hrvatskim znakovima
(**) dodatno ovisi o poretku okteta – Little- ili Big-endian

Stoga je lakše raditi s nizovima znakova i prepustiti postojećoj programskoj potpori brigu o različitim kodiranjima.

Za rješavanje ovog zadatka najprije je potrebno kreirati klasu `Artikl` koja će enkapsulirati osnovne podatke o pojedinom artiklu. `Artikl` ima dva svojstva – `naziv` i `cijena` koja se zadaju konstruktorom i mogu se samo čitati. Dva artikla su jednaka ako im je naziv jednak (pretpostavlja se da će im i cijena biti jednaka), a prirodni poredak artikala je abecedno po nazivu. Pošto je preopterećena metoda `equals`, potrebno je preopteretiti i metodu `hashCode` tako da bude s njom konzistentna (*consistent with equals*).

Zatim je potrebno implementirati novu klasu sličnu klasi `MyByteReader` iz prethodnog zadatka (neka se u ovom slučaju zove `MySecondByteReader`). Klasa `MySecondByteReader` treba imati kolekciju artikala koja se inicijalizira u konstruktoru. Čitanje datoteka u ovom zadatku treba ostvariti koristeći ulazni tok znakova (`java.io.Reader`). To se može ostvariti dohvaćanjem `InputStream`-a, kojeg se zatim omata instancom `InputStreamReader`-a i instancom `BufferedReader`-a. U metodi za obilazak datoteka potrebno je proći kroz sve račune direktorija 'racuni', te spomenutu kolekciju napuniti artiklima koji su se pojavili na računima. Kako bi se to postiglo potrebno je čitati s ulaza redak po redak. Retke zaglavlja i podnožja treba odbaciti (svi redci koji počinju s `"Račun br."`, `"Kupac"`, `"---"` ili su prazni, vidi skicu računa). Za svaki redak koji sadrži neku stavku treba kreirati instancu klase `Artikl` sa zadanim podacima. Svaki artikl treba pohraniti u kolekciju – odabrati primjerenu kolekciju u kojoj neće postojati dvije instance istog artikla.

U glavnom programu (klasa `Program` u paketu drugog zadatka) potrebno je instancirati `MySecondByteReader` *visitora*, te pozvati metodu `walkFileTree` sa stvorenim *visitorom* i instancom `Path` klase koja predstavlja direktorij 'racuni'.

Nakon čitanja svih datoteka i popunjavanja cijele kolekcije potrebno je kolekciju ispisati abecednim redom u dvije datoteke – jednu datoteku kodiranu prema ISO-8859-2 (hrvatski, gotovo identičan CP-1250), a jednu prema UTF-8 kodiranju. Neka se zovu `cjenik.88592.txt` i `cjenik.utf8.txt`.

Za upis sadržaja u datoteke koristiti klase `BufferedWriter` i `FileWriter`.

Zadatak 3 – Dekorator toka podataka i JUnit

U ovom zadatku cilj je pokazati kako se korištenjem operacije xor nad ponorom podataka može ostvariti enkripcija, odnosno dekripcija sadržaja datoteke. Izvođenjem operacije xor nad bitovima ponora i nekog zadanog ključa, stvara se datoteka čiji je sadržaj nečitljiv korisniku koji ne posjeduje informaciju o korištenom ključu pri izvođenju operacije xor. Korisnik koji zna koji se ključ koristio pri izvođenju operacije xor, provodi identičan postupak nad nastalom šifriranom datotekom-šifratom (operaciju xor s poznatim ključem), kako bi dešifrirao tu datoteku, tj. stvorio datoteku identičnu polaznoj.
Za izvođenje ovog zadatka potrebno je najprije implementirati klasu `MaskStream` koja nasljeđuje klasu `OutputStream` i preko konstruktora prima referencu na postojeći `OutputStream` kojem će prosljeđivati izmijenjene oktete. Također, u konstruktoru je potrebno postaviti i ključ tipa `byte`. Metodu `write` treba nadjačati tako da primljeni argument tipa `int`, xor-a s ključem i zapisuje u izlazni tok.
Zatim je potrebno implementirati klasu `MyCriptByteWriter` koja nasljeđuje klasu `MyByteWriter` iz prvog zadatka. U njoj je potrebno nadjačati metodu `run`, koja se razlikuje od nadjačane metode po tome što `OutputStream` omata instancom `MaskStream`-a (kojoj dodatno predaje proizvoljno odabrani ključ).
U glavnom programu (klasa `Program` u paketu trećeg zadatka) potrebno je stvoriti tri instance klase `Path` koje predstavljaju: datoteku koja će biti šifrirana, datoteku u koju će biti zapisan šifrat i datoteku u koju će biti zapisana dešifrirana kopija početne datoteke. Potom treba provesti šifriranje jednom instancom `MyCriptByteWriter`-a kojoj se kao argument predaje ulazni tok nad odabranom datotekom koja se želi šifrirati i instanca klase `Path` u koju će šifrat biti zapisan. Dešifriranje se provodi novom instancom `MyCriptByteWriter`-a kojoj su argumenti ulazni tok podataka nad datotekom u koju je zapisan šifrat i instanca klase `Path` u koju će biti dešifriran šifrat.

Kako bi se uvjerili da je dešifriranjem zaista dobivena datoteka identična polaznoj, u ovom zadatku treba napisati `JUnit` testove (dodatna klasa `Test` u paketu trećeg zadatka). Potrebno je napisati dva testa kojim se utvrđuje da je veličina polazne i konačne datoteke jednaka, te da su pročitani redci polazne i konačne datoteke jednaki. U ovu svrhu preporuča se korištenje odgovarajućih metoda klase `Files` (`readAllLines i size`).