Bacskai Attila

Dokumentáció

Felhasználói:

Ennek a programnak elég egyszerű a kezelőfelülete. A felhasználó kiválaszthatja, hogy egy fájlból szeretné beolvasni az adatokat, vagy kézzel szeretné beírni. A program hatékony futása érdekében, a felhasználónak csakis kétbetűs szavak lehet megadni, speciális karakterek nélkül. Ha véletlenül kerül bele speciális karakter (.:?\_), akkor azokat a program kiveszi belőle. Ha nem 2 betűs szavak kerülnek bele, akkor azokat két betűnkként felbontja és azokkal dolgozik tovább. Ha csak 1 betűt kap, akkor azt egyszerűen el fogja dobni, így ha páratlan hosszúságú szót ad a felhasználó, akkor az utolsó betűt eldobja. Ha a felhasználó fájlból akar beolvasni, akkor egy txt fájlnak a nevét kell megadni „.txt” nélkül. Ha viszont a felhasználó kézzel akarja bevinni az adatokat, akkor ezt addig teheti meg amíg nem ad be egy üres sort.

Ezek után a program működéséből a felhasználó már csak a végeredményt látja, ami a szavak illetve a szavak előfordulásának, kulcs érték párjai.

Programozói:

A program 8 osztályból, és egy tesztosztályból áll. A főprogramban vannak deklarálva a tárolók, amik az egész osztályban látható statikus listák. Csak egy darab getter metódus van, az is csak a tesztelés miatt.

A hívások a main fv-ból indulnak ki. A main-ben először van egy bekérés a felhasználó felől. Ha úgy dönt a felhasználó, hogy fájlból akarja olvasni az adatok, akkor be kell adnia a fájl nevét. A program ehhez hozzáfűt egy „.txt” és meghívja a read fv-t. Ebben a függvényben van egy végtelenített ciklus ami addig olvas, amíg nem olvas be egy null-t vagyis vége a fájlnak. Az olvasás soronként történik, amit azonnal fel is darabol és hozzáadja a data nevű listázhoz. Ezzel vége a metódusnak, így visszatér a mainbe és folytatódnak a hívások.

A program további részéből a felhasználó már csak a végeredményt érzékeli. A következő fv hívás a startmap metódus. Ebben a metódusban van egy számlálós ciklus, ami négyesével lép. A cikluson belül egy try blokkban hoz létre 4 szálat. Azért lép négyesével a ciklus, mert az első szál az i-edik elemt a következő szál az i+1 kapja és így tovább. A szálak azért vannak try blokkban, mert nembiztos hogy mindegyik el tud indulni, de ha nem tud, akkor sem szakítja meg a program futását. A négy szál négy különböző listába (thread1,thread2…) szedi szét a data listát. Minden szál egy osztály meghívásával jön létre és kap egy nevet, egy adat darabot a data listából és egy pointert az előbb említett thread listákra. Egy szál elindulása a szál.start fv-el indul, ami meghívja a run metódust. Ezt mindenképp meg kell írni, mert abban az interface-ben amit beimplementál az osztály(Runnable), már meg van írva mint abstract metódus. A Runnable interface teszi lehetővé azt, hogy a program képes legyen szálakat kezelni.

A run metódusban meghívódik a Mymap-map metódusa. Itt egy regex segítségével kiszedi a Stringből azokat a karaktereket amik nem kellenek. Aztán megvizsgálja a String méretét. Ha pont kettő hosszú, akkor egyszerűen hozzáadja a Main-ben található, megfelelő thread listához. Ha kettőnél kisebb méretű, akkor egyszerűen figyelmen kívül hagyja. Ha viszont 2-nél nagyobb a mérete, akkor meghívja a ciklus nevű metódust, ami feldarabolja a Stringet egy String tömbbé, aztán pedig kettesével hozzáfűzi a megfelelő thread listához. Itt sem foglalkozik az egyedül hagyott karakterrel.

Ezek után jön a Main következő fv hívása, a startshuffle. Ez egy szálon fut,mert folyamatosan dolgozik a finaly listán, így egyszerre több szál nehézkesen tudna egyszerre keresni, beszúrni vagy hozzáadni a listához. Az összes thread listával meghívja a Shuffle osztály shuffle metódusát, ami összefésüli a négy listát egy listákat tároló listába(finaly). Az összefésülés, úgy történik, hogy megy egy számlálós ciklus a (thread) lista végéig. A ciklus elindulása előtt a listát rendezzük. A cikluson belül egy while ciklus indul, ami addig fűzi az elemeket egy temp listához, amíg ugyan olyan elemeket talál egymás után. Ha ez a while megszakad, megnézi, hogy van e már lista a finaly-ben és ha nincs, akkor hozzáfűzi, ha pedig már van, akkor megnézi, hogy van e olyan lista,ami olyan elemeket tartalmaz, mint a temp. Ha nincs akkor hozzáfűzi a finaly listához a temp-et, ha viszont van, akkor a temp tartalmát adja hozzá a már meglévő listához.

Az utolsó lényeges hívás a startreduce metódust hívja meg. Itt ugyan úgy négy szál jön létre, meghívja a ThreadsForReduce class-t. Ez a klassz majdnem ugyan olyan mint amit a map-nél használtunk, annyi különbséggel, hogy itt a run metódusban a Reduce osztálynak a reduce metódusát hívjuk meg. Ebben a metódusban annyi történik, hogy a megkapott lista első elemét belerakja egy SortedMap tárolóba, mint key érték és a value helyére pedig a lista méretét teszi bele.

A program végén van még egy kiir metódus, azért hogy lássuk, hogy a program helyesen futott e le.

A tárolók amiket a program használ, mind a Collections synchronizedList illetve ugyanilyen Map-et használ. Ha sima listát használnának a szálak, akkor lehet hogy belekerül egy null, esetleg nem kerül bele semmi, de az is lehet hogy nem lesz probléma. Az ilyen fajta létrehozás lekezeli ezeket a hibákat, automatikusan lezárja a listát, ha egy szál használja illetve felnyitja ha már nem.

