**TEHNIČKA ŠKOLA BJELOVAR**

**Dr. Ante Starčevića 24**

**IME I PREZIME**

**ZAVRŠNI RAD**

**PROGRAMIRANJE U JAVI**

**Područje rada: ELEKTROTEHNIKA**

**Program: TEHNIČAR ZA RAČUNALSTVO**

**Bjelovar, travanj 2016.**

**TEHNIČKA ŠKOLA BJELOVAR**

**Dr. Ante Starčevića 24**

**IME I PREZIME**

**ZAVRŠNI RAD**

**PROGRAMIRANJE U JAVI**

**Područje rada: ELEKTROTEHNIKA**

**Program: TEHNIČAR ZA RAČUNALSTVO**

**Mentor: ime prezime, prof.**

**Bjelovar, travanj 2016.**

Nadnevak predaje rada: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

( mentor je prihvatio pisani dio izradbu )

Potpis mentora: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ocjena izradbe završnog rada: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Nadnevak obrane završnog rada:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ocjena obrane završnog rada:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

KONAČNA OCJENA:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

( prijedlog Povjerenstva )

Povjerenstvo:

Mentor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profesor struke: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profesor struke: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profesor struke: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profesor struke: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Izdvojeno mišljenje ili eventualni komentar:

*Rad je izrađen u Tehničkoj školi Bjelovar (ili naziv poduzeća u kojem je rad izrađen).*

*Rad ima 27 stranica, 1 sliku, i 1 tablicu.*

Sadržaj

1. UVOD

Tema završnog rada je programiranje u programskom jeziku Java. Cilj ovog rada je pružanje dovoljno informacija i razrada teme o Javi. U radu se pobliže objašnjavaju osnove tog programskog jezika. Rad je podijeljen u više cjelina.

Prvo je opisan Java jezik kao program, potom integrirano razvojno okruženje Eclipse u kojem programiram rad, a nakon toga objašnjenja programiranja Jave kroz razne primjere. Ovu temu sam izabrao iz razloga što mi se ovo područje računalne struke tijekom dosadašnjeg školovanja najviše svidjelo. Java je postala sve popularnija u raznim područjima, te jedan od rijetkih programskih jezika u kojima je standardiziran rad s elementima grafičkog sučelja. Također, programiranje za mrežno okruženje je bitno olakšano te postoji niz biblioteka koje programerima omogućuju rješavanje kompleksnijih problema. Zbog njene velike iskoristivosti, java programeri su najbolje plaćeni u svojoj struci, te me je to potaknulo i dobio sam želju da trenutna znanja iz jave proširim u daljnjem školovanju, primjenom na konkretnim zadacima i problemima iz prakse. Mislim da bi Java mogla naslijediti C++ glede programiranja, i u školstvu, te da je Java u stvari ono što je trebao biti C++. Izrada završnog rada će se vršiti u aplikaciji Eclipse IDE for Java EE Developers. Ovim radom, kroz zadatak i teoriju objasnit ću temelj i osnove Jave, te ponešto o algoritmima koji se nalaze u zadatku.

2. RAZVOJ JEZIKA I NJEGOVA NAMJENA

Java je danas jedan od najpopularnijih programskih jezika u raznim područjima, od programiranja igara do vrlo važnih aplikacija, kao što su aplikacije za trgovanje na burzama ili za upravljanje svemirskim vozilima. Programski jezik Java dizajnirao je James Gosling 1995. godine dok je radio u tvrtci Sun Microsystems (trenutno je ona podružnica tvrtke Oracle Corporation). Cilj je bio pružiti jednostavniju alternativu jeziku C++ koja će biti neovisna o platformi. Java je programski jezik opće namjene koji se koristi u svim industrijama i za gotovo sve vrste aplikacija. Ako ga savladate, šanse da se zaposlite kao razvojni inženjer bit će veće nego ako se specijalizirate za samo jedan programski jezik posebne namjene. Ne samo da je Java programski jezik otvorenog izvornog koda, već postoje i milijuni projekata otvorenog izvornog koda koji se razvijaju u Javi. Pridruživanje nekom od njih najbolji je način da se upoznate s procesom razvoja aplikacija i osigurate si prvi posao a da prije toga niste stekli nikakvo iskustvo kao programer. Programski jezik Java je objektno orijentiran, što omogućava da lako povezujete programske konstrukcije s objektima iz stvarnog svijeta. Postoje različite vrste programskih jezika. U nekima od njih pišete tekst programa (tzv. Izvorni kod) koji možete odmah izvršiti. Takve jezike nazivamo interpretiranim jezicima (JavaScript). Ali, zahtijeva da se izvorni kod prije izvršavanja kompajlira. On se pretvara u kod specifičan za procesor koji mogu razumjeti neki izvršni ili virtualni strojevi. Java kompajler neće samo provjeriti postoje li u kodu sintaktičke pogreške, već se nakon kompajliranja programu mogu dodati (pozvati) neke druge biblioteke Java koda. Ako želimo koristiti određeno računalo za razvoj Java programa, moramo imati JDK verziju, a ako želimo koristiti samo za izvršavanje programa koji su kompajlirani u nekom drugom računalu potrebno nam je okruženje Java Runtime Enviroment (JRE).

3. IZRADA I OBJAŠNJENJA OSNOVNOG JAVA PROGRAMA

*Public class Helloworld {*

*Public static void main (String[] args) {*

*System.out.println(“Hello World“);*

*}}*

Naš prvi program sadrži klasu HelloWorld. Java klasa i njena datoteka moraju imati isto ime. (Mogu postojati iznimke od ovog pravila, ali ne u ovakvom kratkom programu.)

Klasa Helloworld sadrži metodu main(). Metode u Java klasama predstavljaju funkcije (akcije) koje klasa može izvršavati. Java klasa može sadržavati nekoliko metoda, ali ako se jedna od njih zove main() i deklarirana je (ako ima potpis metode), kao u ovoj klasi, to čini java klasu izvršnom. Ako klasa ne sadrži metodu main(), ona se može koristiti iz drugih klasa ali je ne možete pokrenuti kao program. Metoda main() poziva metodu println() za prikazivanje teksta “Hello World“ na zaslonu.

Evo potpisa metode main():

*Public static void main (String[] args)*

Ovaj potpis sadrži razinu pristupa (public), upute za upotrebu (static), tip povratne vrijednosti (void), ime metode(main) i popis argumenata (string [] args).

Ključna riječ public znači da se metodi main() može pristupiti iz bilo koje druge Java klase.

Ključna riječ static znači da ne moramo napraviti instancu klase da biste mogli koristiti njene metode.

Ključna riječ void znači da metoda main() ne vraća nikakvu vrijednost pozivajućem programu.

Ključna riječ string[] args nam govori da će metoda kao argument primiti bilo koji niz znakova. (Ovoj metodi možete proslijediti vanjske podatke kroz naredbeni red.)

Metoda main() je početna točka programa. Možete napisati program koji sadrži više klasa, ali bar jedna mora imati metodu main() jer se program neće moći pokrenuti. Java klasa može sadržavati više metoda. Tijelo metode main() sadrži sljedeći red:

*System.out.println(“Hello World“);*

Metoda println() koristi se za ispisivanje podataka na sistemskoj konzoli. Iza naziva java metode uvijek slijede zagrade.

U ovom redu System predstavlja još jednu Java klasu.

Zapis s točkom System.out nam govori da je varijabla out definirana unutar klase system.

Out.println() govori da postoji objekt predstavljen varijablom out i da sadrži metodu println(). Zapis s točkom koristit ćete za referenciranje metoda ili varijabli klase.

Sada bi morali kompajlirati program. Kompajler javac dio je paketa JDK, pa možemo otvoriti naredbeni prozor na računalu, tako da prvo prijeđemo u mapu gdje se to nalazi :

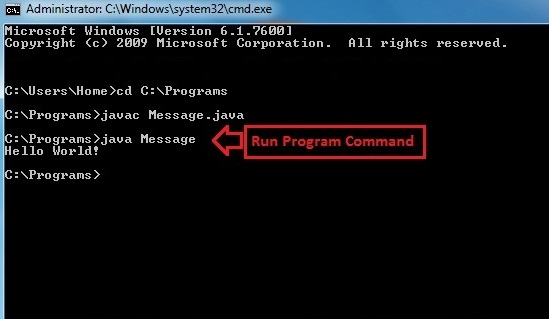
cd “ime mape“

javac HelloWorld.java

Kad je program uspješno kompajliran moramo pokrenuti program koristeći naredbu:

Java HelloWorld

Zamijetimo da smo ovaj put zadali naredbu java, što pokreće virtualni stroj za javu. U naredbenom prozoru ispisane su riječi “Hello World“.



Naredba za kompajliranje i rezultat

4. INTERGRIRANO RAZVOJNO OKRUŽENJE ECLIPSE

Iako smo prvi Java program napisali u običnom editoru teksta i kompajlirali ga u naredbenom redu, to nije produktivan način razvoja softvera. Profesionalni programeri koriste integrirano razvojno okruženje (IDE) koje se sastoji od editora, kompajlera sustava pomoći, programa za uklanjanje pogrešaka i drugih alata. Popularna razvojna okruženja su Eclipse, NetBeans , InteliJ IDEA i RAD. Neka su besplatna a neka se moraju kupiti. Eclipse je IDE okruženje koje se najčešće koristi. Razvojno okruženje Eclipse je proizvod otvorenog izvornog koda čiji je nastanak pomogla donacija velike količine izvornog koda tvrtke IBM zajednici koja je razvijala Javu. Od tog trenutka Eclipse je proizvod koji u potpunosti razvija zajednica. Započelo je kao integrirano okruženje za razvoj Java programa, ali danas je to razvojna platforma koja se koristi za gađenje tisuća alata i dodataka. Neki programeri koriste njen Rich Client Platform (RCP) za razvoj korisničkih sučelja za aplikacije. Eclipse podržava i druge Java jezike. Pored toga što je integrirano razvojno okruženje, Eclipse podržava i dodatke. Svaki programer može uključiti samo one dodatke koji ga zanimaju. Na primjer, postoji dodatak za prikaz UML dijagrama, drugi nudi sustav za izvješćivanje, a postoje i dodaci za razvoj programa u C-u, Adobe Flexu i drugim programskim jezicima.

4.1. JAVA PAKETI

Paketi se u javi koriste za bolje organiziranje projekata koji se sastoje od više datoteka i za zaštitu podataka. Nije neuobičajeno da se projekt sastoji od nekoliko stotina Java klasa i nikako nije dobra ideja sve ih čuvati u jednoj mapi. Zbog toga će datoteke biti smještene u različitim mapama i podmapama. Koja su pravila za imenovanje paketa? Java programeri koriste takozvano pravilo obrnute domene. Recimo da radimo na projektu Sales za tvrtku Acme, koja ima web stranicu na adresi acme.com. Tada će svako ime paketa počinjati sa obrnutom URL adresom web stranice tvrtke iz koje slijedi ime projekta: com.acme.sales. Sve Java klase koje pripadaju tom paketu čuvat će se u sljedećoj strukturi mapa: com/acme/sales.

Ako su neke Java klase specifične za prodaju u matičnoj zemlji a druge za prodaju u inozemstvu, izradit ćete još dvije podmape: com/acme/sales/domestic i com/acme/sales/international. Dok su imena mapa razdvojena kosom crtom ili obrnutom kosom crtom, odgovarajući Java paketi odvojeni su točkama. Java ima posebnu ključnu riječ package i njena deklaracija treba biti prvi red klase. Na primjer:

*Package com.acme.sales.domestic;*

Osim što se koristi za bolju organizaciju Java klasa, pomažu pri kontroliranju pristupa podacima.

5. OBJEKTNO ORIJENTIRANO PROGRAMIRANJE

Java je objektno orijentiran jezik, što znači da ima konstrukcije za predstavljanje objekata iz stvarnog svijeta. Svaki Java program ima najmanje jednu klasu koja zna kako obaviti određeni posao ili kako predstaviti neke tipove objekata. Na primjer, najjednostavnija klasa, HelloWorld, zna kako pozdraviti svijet. Klase u Javi mogu imati metode i polja. Izradit ćemo i analizirati klasu po imenu Car. Ova klasa će imati metode koje opisuju što taj tip vozila može učiniti primjerice, pokrenuti motor, ugasiti ga, ubrzati, i tako dalje. Klasa će imati i neka polja: boju vozila, broj vrata, cijenu, i tako dalje.

*Class Car{*

*String color;*

*Int numberofDoors;*

*Void startEngine{*

*// Ovdje dolazi nekakav kod*

*}*

*Void stopEngine{*

*Int tempCounter=0;*

*// Ovdje dolazi nekakav kod*

*} }*

Redovi koji počinju s dvije kose crte su komentari.

Car predstavlja zajedničke karakteristike brojnih modela automobila. Svi automobili imaju svojstva kao što su boja i broj vrata i svi obavljaju sličan posao. Izrada objekata, na temelju klasa jednaka je izradi pravih automobila na temelju nacrta. Izraditi instancu klase znači u memoriji računala izraditi objekt prema definiciji u klasi. Da biste instancirali klasu deklarirat ćemo varijablu tipa te klase i upotrijebiti operator new za svaku novu instancu automobila, npr:

*JamesBondCar car1=new JamesBondCar();*

Neke vrijednosti koje predstavljaju objekt mogu se mijenjati tokom vremena (varijable) dok druge ostaju iste(konstante).

5.1.VARIJABLE

Deklariranje varijabli – Java je statički tipiziran jezik: varijable programa moramo prvo deklarirati i nakon toga im možemo pridruživati vrijednosti bilo u vrijeme deklariranja bilo kasnije u nekoj od metoda klasa. Klasa Car definira varijablu color tipa String koja se koristi za predstavljanje tekstualnih vrijednosti.

5.2.KONSTANTE

Za čuvanje vrijednosti koje se nikada ne mijenjaju trebamo deklarirati konstantu, samo dodajte ključnu riječ final, npr:

*Final String MANUFACTURER;*

Java programeri nazive konstanti pišu velikim slovima. Vrijednost se konstanti može dodijeliti samo jednom, a pošto radimo instancu specifičnog automobila, njegov proizvođač je poznat i ne može se promijeniti tokom trajanja objekta:

*MANUFACTURER=“J.B.LIMITED“;*

Iako ne moramo inicijalizirati varijable final tokom njihove deklaracije, to je uvijek dobro učiniti. Prethodni primjer bi trebao biti napisan ovako:

*Final String MANUFACTURER=“J.B.Limited“;*

5.3.OSNOVNI TIPOVI PODATAKA

U Javi postoji osam osnovnih tipova podataka: četiri su za cjelobrojne vrijednosti, dva za vrijednosti s decimalnom točkom, jedan za čuvanje pojedinačnih znakova i jedan za logičke podatke koji mogu biti samo true ili false.

Tablica popisuje neke karakteristike tih JAVA tipova podataka.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OSNOVNI TIP | VELIČINA | MIN VRIJEDNOST | MAX VRIJEDNOST | KLASA OMOTAČA |
| Byte | 8 bita | -128 | 127 | Byte |
| Short | 16 bita | -32, 768 | 32,767 | Short |
| Int | 32 bita | -2 147 483 648 | 2 147 483 647 | Integer |
| Long | 64 bita | -9 223 372 036, 854 775 808 | 9 223 372 036 854 775 807 | Long |
| Float | 32 bita | Pokretni zarez jednostruke preciznosti | Pokretni zarez jednostruke preciznosti | Float |
| Double | 64 bita | Pokretni zarez dvostruke preciznosti | Pokretni zarez dvostruke preciznosti | Double |
| Char | 16 bita | Unicode 0 | Unicode | Character |
| Boolean | - | False(nije min) | True(nije max) | Boolean |

Tablica tipova podataka

5.4. DOSEG VARIJABLE

Doseg varijable definira koliko dugo će varijabla živjeti i ostati korisna. Ako varijablu deklariramo unutar metode, ona će imati lokalni doseg. To znači da je vidljiva samo kodu unutar te metode. Kada se izvršavanje metode završi, javin sakupljač otpada sve lokalne varijable automatski uklanja iz memorije. Zapravo, doseg je nešto uži od toga. Varijable su dostupne u metodi samo nakon što su deklarirane i samo unutar bloka u kojem su deklarirane. Na primjer, varijabla deklarirana unutar petlje for neće biti dostupna izvan petlje for čak ni unutar iste metode. Ako varijabla mora biti dostupna i izvan metode klase, deklarirajte je na razini klase.

5.5.OMOTAČI, AUTOMATSKO PAKIRANJE I RASPAKIRAVANJE

Svi osnovni tipovi podataka imaju odgovarajuću klasu omotača koja sadrži korisne metode za rad s njima. Klasa omotača ima dvije svrhe:

Sadrže određeni broj korisnih funkcija za rad s osnovnim tipovima. Na primjer klasa integer nudi korisne metode kao što su pretvaranje string u int, ili pretvaranje int u float i druge. Klasa integer također dopušta da postavite minimalnu i maksimalnu vrijednost za broj o kojem se radi.

Neke java kolekcije ne mogu čuvati osnovne tipove (primjerice ArrayList) pa oni moraju biti umotani u objekte. Na primjer:

*ArrayList myLotteryNumbers=new ArrayList();*

*myLotteryNumbers.add(new Integer(6));*

*myLotteryNumbers.add(new Integer(15));*

Ne moramo eksplicitno praviti novu instancu za svaki osnovni tip kao u prethodnom odlomku koda. Možemo jednostavno napisati:

myLotteryNumbers.add(6); i osnovna vrijednost 6 će automatski biti umotana u instancu klase integer. Ta značajka se naziva automatsko pakiranje.

5.6. KOMENTARI U PROGRAMIMA

Dok pišete kod Java programa trebate u njega dodavati komentare, tekst koji objašnjava što program radi. Programi se mnogo češće čitaju nego pišu. Sigurno će se dogoditi situacija kada će drugi programeri čitati i pokušati razumjeti vaš kod. Budite učtivi i olakšajte im posao. Najbolje je prvo pisati komentare, a zatim kod. Komentari se pišu tako da započnemo sa prvo dvije kose crte, npr: // ovdje ide komentar

5.7. UVJETNI ISKAZ IF

Java sadrži iskaz if koji utvrđuje da li je neki uvjet točan ili netočan. Na temelju odgovora na pitanje program će se i dalje odvijati na jedan ili drugi način.

Ako uvjetni izraz vrati true, izvršit će se kod između prvih vitičastih zagrada. U suprotnom će se izvršiti kod nakon iskaza else. Na primjer:

*If (totalOrderPrice>100) {*

*System.out.println(“You'll get a 20% discount“);*

*}*

*Else{*

*System.out.println(“Order books for more than a“ + “$100 to get a 20% discount“);*

*}*

Budući da ovaj uzorak koda ima za izvršiti samo jedan iskaz u klauzulama if i else, korištenje vitičastih zagrada nije obavezno, ali oni čine kod čitljivijim i sprječavaju nastajanje pogrešaka koje se teško otkrivaju ako kasnije budete dodavali još koda u iskaz if. To program čini razumljivijim drugim osobama koje moraju čitati što ste napisali. Nadalje, možemo izgraditi izraze koristeći i logičke operatore AND, OR i NOT. Operator AND se predstavlja s &&, OR sa ||, a NOT sa znakom !.

5.8. ISKAZ SWITCH

Iskaz switch je alternativa iskazu if. Oznaka case u uvjetu prebacivanja (taxCode) se evaluira i program se prebacuje na jednu od sljedećih case klauzula:

*Int taxCode=someObject.getTaxCode (grossIncome);*

*Switch(taxCode){*

*Case 0;*

*System.out.println(“Tax Exempt“);*

*Break;*

*Case 1;*

*System.out.println(“Low Tax Bracket“);*

*Break;*

*Case 2;*

*System.out.println(“High Tax Bracket“);*

*Break;*

*Default:*

*System.out.println(“Wrong Tax Bracket“);*

*}*

Prethodni kod će pozivati samo metodu println(). Ne zaboravimo staviti break na kraj svakog iskaza case kako bi program mogao napustiti iskaz switch nakon obrade slučaja. U suprotnom će kod „propasti“ i ispisati više redova čak i ako taxCode može imati samo jednu vrijednost.

5.9. NASLJEĐIVANJE

U objektno orijentiranim jezicima izraz inheritance znači mogućnost definiranja nove klase na temelju postojeće (ne ispočetka).

Specijalna ključna riječ extends koristi se kako bi se naznačilo da jedna klasa nasljeđuje od druge.

Primjer:

*Class Potomak extends Roditelj {}*

Klasa Potomak će imati sve značajke klase Roditelj ali u nju možete dodati nova svojstva i metode. U takvom slučaju klasa Roditelj se naziva nadklasa, a Potomak podklasa. Nova klasa moći će pristupiti metodama nadklase, osim ako nemaju posebno zadanu razinu pristupa.

5.10. PREMOŠĆIVANJE METODA

Još jedna važna tehnika u objektno orijentiranom programiranju je premošćivanje metode. Zamislimo klasu Tax sa dvadeset metoda. Većina ih radi bez problema u svim državama, ali jedna metoda se ne može koristiti za obračunavanje poreza u državi New Yersey. Umjesto da mijenjate tu metodu u nadklasi, možemo izraditi drugu metodu u podklasi s istim imenom i popisom argumenata. Ako podklasa ima tu metodu s istim imenom i popisom argumenata, ona će se premostiti (potisnuti) odgovarajuću metodu svog roditelja (pretka).

Premošćivanje metoda je korisno kada:

Izvorni kod nadklase nije dostupan a morate promijeniti njenu funkcionalnost.

Originalna verzija metode i dalje vrijedi za neke slučajeve i ne želite je mijenjati.

Koristite premošćivanje metoda kako biste koristili polimorfizam

6. METODE KLASE

6.1. ARGUMENTI METODE

Metode sadrže kod za akcije ili funkcije koja klasa može izvoditi.

Svaka metoda klase izvršava neku funkciju, kao što je obračun poreza, postavljanje narudžbe ili pokretanje automobila. Ako metoda zahtjeva vanjske podatke za obavljanje funkcije, oni se mogu pružiti u obliku argumenata ili parametara. U potpisu metode morate deklarirati tip i naziv svakog argumenta. Na primjer:

*Int calcLoanPayment (int amount, int numberOfMonths, string state) {*

*// Ovdje dolazi kod*

*}*

Kada pozovete funkciju, izvršno okruženje java pokušava pronaći funkciju deklariranu sa zadanim potpisom. Na primjer, ako prethodnu funkciju pozovemo na sljedeći način:

*calcLoanPayment (20000, 60);*

Java kompajler će prijaviti pogrešku da funkcija calcLoanPayment() koja prima dva argumenta nije pronađena.

6.2. PREOPTEREĆIVANJE METODE

Ako želimo dopustiti pozivanje funkcije s različitim brojem argumenata, moramo izraditi više verzija funkcije. Na primjer, možemo izraditi funkciju koja će koristiti državu New York kako programeri ne bi morali navoditi državu kao argument. Ako se većina kredita obračunava za stanovnika New Yorka, takva funkcija može biti dobra ideja.

*Int calcLoanPayment ( int amount, int numberOfMonths) {*

*// Kod za obračun kredita u New Yorku dolazi ovdje*

*}*

Preopterećivanje metode znači da u klasi postoji više metoda s istim imenom ali različitim popisom argumenata. Metoda može biti preopterećena ne samo u istoj klasi, već i u izvedenoj. Na primjer, klasa LoanCalculator može imati funkciju calcLoanPayment() definiranu s tri argumenta, dok njen potomak

MyLoanCalculator može sadržavati verziju calcLoanPayment() koja uzima dva argumenta. Zašto bismo preopterećivali metode u klasama? Zato da bi korisnicama klasa pružili fleksibilnije programsko sučelje.

6.3. KONSTRUKTORI

Kada program napravi instancu klase,Java poziva konstruktor klase – posebnu metodu koja e poziva samo jednom, kada se instanca gradi pomoću operatora new.

Primjer:

*Tax t = new Tax();*

Zagrade u prethodnom odlomku koda znače da ovaj kod poziva konstruktor bez argumenata na klasi Tax. Čak i ako nismo deklarirali konstruktor na klasi, java će za vas izraditi konstruktor bez argumenata.

Konstruktori imaju sljedeće karakteristike:

1. Pozivaju se kada se klasa instancira.
2. Moraju imati isto ime kao i klasa u kojoj se nalaze.
3. Ne mogu vratiti vrijednost i ne zadajete ključnu riječ void kao povratni tip.

Konstruktori se najčešće koriste za pridruživanje vrijednosti varijablama klase.

6.4. KLJUČNE RIJEČI SUPER,THIS I STATIC

Ključna riječ super omogućava da izravno pozovemo metodu ili konstruktor iz roditeljske klase.

Ključna riječ this je korisna kada se trebamo referencirati na instancu klase iz njene metode. Ona pomaže da se riješe sukobi imena. Kako bismo virtualnom stroju za Javu naložili da koristi varijablu instance grossIncome, koristimo sljedeću sintaksu:

*This.grossIncome=5000;*

Java ima specijalnu ključnu riječ, static, koja govori da će varijablu klase zajednički koristiti sve instance iste klase. Ako klasa Tax ima deklaraciju static double grossIncome; tada će vrijednosti te varijable moći zajednički koristiti sve instance klase Tax, što nema smisla. Ako u klasu Tax uvedemo varijablu klase za prebrojavanje njenih instanci, takva varijabla mora biti deklarirana kao statička kako bi njena jedina verzija mogla biti povećana nakon izrade svake instance.

7. NIZOVI

Niz (engl. Array) je spremište u kojem se čuva više podataka istog tipa. Oni nude najbrži pristup kolekciji podataka, ali moramo unaprijed znati broj elemenata koji će se u njima čuvati. Srećom, java ima klase koje nisu ograničene na taj način i možemo dodati elemente u kolekciju tokom izvršavanja, ako to bude bilo potrebno. Recimo da program mora čuvati imena dvadeset djevojaka. Umjesto deklariranja dvadeset varijabli string, možemo deklarirati jedan string niz s kapacitetom za smještaj dvadeset elemenata:

*String [] friends = new String [20]; // Deklarira i instancira niz*

*Friends [0] = “Ivana“; // Inicijalizira prvi element*

*Friends [1] = “Martina“; // Inicijalizira drugi element*

*// Nastavimo inicijalizirati elemente niza*

*Friends [19] = “Petra“; // Inicijalizira posljednji element*

Prvi element niza u javi ima indeks 0. Kada deklariramo niz uglate zagrade možemo smjestiti nakon tipa podataka ili nakon imena varijable. Obje ove deklaracije su točne:

*String friends[];*

*String[] friends;*

Isto tako, klase vector i arraylist pripadaju grupi niza. Obje te kolekcije interno koriste nizove za spremanje, ali ako nastavljamo dodavati elemente u kolekcije one interno povećavaju kapacitet niza, odnosno, ako obrišemo elemente, smanjuju ga. U te klase se spremaju samo objekti – dopušteni su samo osnovni. Pored toga, Java podržava automatsko pakiranje i ako pokušamo dodati primitivni objekt na kolekciju, on će automatski biti pretvoren u odgovarajući objekt omotač. Arraylist je sporiji od niza jer mora obaviti interno kopiranje niza kako bi se promijenila veličina kolekcije, a vector je još sporiji jer podržava sinkronizaciju niti, te je postala manje popularna.

8. ZADATAK I OBJAŠNJENJA

U nastavku ovog poglavlja opisan je tekst zadatka kojeg sam kodirao u Javi, zatim opis algoritma i kod.

Tekst zadatka:

U pekari se prodaje burek. Na početku dana ispečeno je n bureka, svaki kvalitete x. Kako vrijeme prolazi peku se novi bureci različite kvalitete, dolaze kupci i kupuju burek. Kada kupac dođe prodavačici: "Daj mi najbolji burek koji imaš!". Prodavačica odabere najbolji burek koji ima i proda mu ga. Pomogni pekari što bolje poslovati tako što će prodavačica uvijek dati kupcima najbolji burek. Zadan je n (1<=n<=100000) i nakon toga n brojeva x. Broj x opisuje kvalitetu bureka. Nakon toga slijedi broj m, zatim m (1<=m<=100000) brojeva y. y opisuje događaje tijekom dana. Ako je y nula, znači da je došao kupac i kupio najbolji burek pa treba ispisati kvalitetu prodanog bureka ili "Nema!" ako nema bureka. Ako y nije nula, znači da je ispečen novi burek kvalitete y koji se od tog trenutka može kupiti.

U praktičnom radu koriste se algoritmi stabla, u ovom slučaju binarno stablo. U zadatku se vektor sprema u binarno stablo koje je zapisano u nizu. Vektor možemo zamisliti kao malo bolje polje. U kodiranju rješenja koristi se i struktura podataka – heap - potpuno uređeno binarno stablo. Kod njega postoji pravilo da je na početku stabla uvijek veći broj.

Kod zadatka:

import java.util.\*;

public class Rad {

public static ArrayList<Integer> *heap* = new ArrayList<Integer>();

public static Scanner *cin* = new Scanner(System.*in*);

public static void main(String[] args)

{

*heap*.add(0);

// u heap ubacujemo nulti element, jer heap ne koristi indeks 0

int n,x;

n = *cin*.nextInt();

for (int i = 0;i < n;i++)

{

x = *cin*.nextInt();

*ubaci*(x);

// Jedan po jedan element ubacujemo u heap

}

int m,y;

m = *cin*.nextInt();

for (int i = 0;i < m;i++) {

y = *cin*.nextInt();

// unosimo y

if (y==0)

// Ako je y 0, provjerava se ima li što u heap-u (došao je kupac)

{

if (*heap*.size() > 1)

{

System.*out*.println(*heap*.get(1));

// Ako ima, ispisuje se najveći element heap-a tj. korijen stabla

*izvadi*();

// Izbacujemo ga

}

else

System.*out*.println("Nema!");}

// Ako je heap prazan ispisuje se "Nema!"

else

*ubaci*(y);

// Ako je ispečen novi burek, ubacujemo ga u heap

}

System.*exit*(0);

}

public static void ubaci(int x)

{

*heap*.add(x);

// Prvo se x dodaje na kraj heap-a

int t = *heap*.size()-1;

// U varijabli t pamti se indeks gdje se nalazi x

while (((t/2)!=0) && *heap*.get(t)>*heap*.get(t/2))

// petlja traje dokle god je potrebno podizati x u heap-u, provjerava je li roditelj elementa s indeksa t unutar heap-a

{

int temp = *heap*.get(t/2);

// Ako je zamijeni x s roditeljem, tj. podigne ga u heap-u

*heap*.set(t/2, *heap*.get(t));

*heap*.set(t, temp);

t/=2;

// Promijenimo t tako da opet pamti indeks varijable x u heap-u

}

}

public static void izvadi()

// služi za vađenje korijena heap-a (najviši čvor u stablu)

{

*heap*.set(1,*heap*.get(*heap*.size()-1));

// Umjesto korijena zapišemo zadnji element iz heap-a (zadnji element postavimo da bude prvi u stablu)

*heap*.remove(*heap*.size()-1);

// Izbacimo zadnji element iz heap-a

int t = 1, r;

// t pamti trenutni indeks varijable koju ispuštamo, a u r izračunavamo s kojim indeksom se heap(t) treba zamijeniti

while(true) {

if (t\*2+1 < *heap*.size())

// Ako heap(t) ima dvoje djece, onda r pamti indeks većeg djeteta (uspoređuje dva susjedna broja da se vidi koji je veći)

{

if(*heap*.get(t\*2) > *heap*.get(t\*2+1))

r = t\*2;

else

r = t\*2+1; }

else if(t\*2 < *heap*.size())

// Ako je samo lijevo dijete u heap-u, onda r pamti njegov indeks

r=t\*2;

else

// Ako heap(t) nema djece onda smo gotovi

break;

if(*heap*.get(r) > *heap*.get(t))

// Provjerava se je li dijete veće od roditelja

{

int temp = *heap*.get(t);

// Ako je, mora se napraviti zamjena

*heap*.set(t, *heap*.get(r));

*heap*.set(r, temp);

t = r;

// Obnavlja se vrijednost varijable t

}

else

// Ako je dijete manje od roditelja gotovo je ispuštanje

break;

}

}}

9. ZAKLJUČAK

Java je jako dobar i koristan program za razvoj web aplikacija koji programeri često prvi nauče koristiti. Kada bi uspoređivali C++ i Javu, C++ je dosta složeniji jezik, pa bi bilo bolje prvo učiti Javu. Prilikom učenja ta dva jezika jasnije se vidi razlika između samog jezika i biblioteka koje idu uz njega. Pa su tako npr. C++ templates puno bolja rješenja od Javinih generics-a. S druge strane, prije ćemo s Javom napraviti nešto korisno (desktop ili server aplikacije), nego u C++ za određeno vrijeme.

Kroz ovaj rad želim potaknuti druge, ponajprije kolege iz škole na uporabu ovog programskog jezika.

10. POPIS LITERATURE

1. Yakov, Fain. Programiranje JAVA. Zagreb:Dobar plan, 2011.
2. https://en.wikibooks.org/wiki/Java\_Programming (14.travnja 2016.)

**KONZULTACIJSKI LIST**

Ime i prezime učenika: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, razred: \_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Red.  br. | Nadnevak konzultacija | Bilješke o napredovanju | Potpis mentora |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |