Klasifikacija invarijanata u klasi

Dušan T. Malbaški, *Member IEEE* i Aleksandar D. Kupusinac

***Sadržaj* — Velika ekspanzija i nagli razvoj objektno orijentisanog programiranja doveli su do problema u značenju pojmova i termina, pa čak i onih fundamentalnih, kao što su objekat, klasa i invarijanta. Zbog toga, nameće se potreba da se uvede jasan i konzistentan konceptualno-terminološki sistem u objektno orijentisanom programiranju. U ovom radu ćemo izložiti konceptualne definicije klase, objekta i invarijante. Konceptualne definicije omogućavaju da se klasa i objekat definišu nezavisno jedan od drugog i na taj način se stvara ambijent u kojem se može napraviti jasna klasifikacija invarijanata u klasi, što je i glavni doprinos ovog rada.**

***Ključne reči* — invarijanta, klasa, konceptualne definicije, objekat, objektno orijentisano programiranje.**

I. Uvod

PRVI korak u proučavanju neke naučne i/ili stručne oblasti jeste upoznavanje sa osnovnim pojmovima i terminima. Objektno orijentisano programiranje je nastajalo evolutivnim putem, kao odgovor na sve veće zahteve softverskog tržišta, a sa druge strane zbog samog progresa u nauci, kao i načina razmišljanja. Zbog toga, već u fazi izgrađivanja teorije i prakse uspostavljeno je faktičko stanje u kojem postoje različiti pogledi na fundamentalne koncepte objektne metodologije – klasu, objekat i invarijantu. Terminologija koja se koristi u objektno orijentisanom programiranju takođe nije zadovoljavajuća, jer pored sinonimije, često se sreće neoprezna upotreba već ustaljenih termina iz drugih naučnih oblasti (naročito filozofije) [1]. U filozofiji objekat jeste stvar, predmet, korelat subjektu i sl. [2]. U objektno orijentisanom programiranju postoji više različitih definicija objekta [3], pa se tako objekat definiše kao individua koja se može identifikovati, jedinica ili entitet, realan ili apstraktan, sa dobro definisanom ulogom u domenu problema [4]; ili bilo šta sa oštro i jasno definisanim granicama [5]; ili stvar koja se može identifikovati [1]; ili instanca klase u vreme izvršavanja [6]; ili model entiteta koji ima identitet, stanje i ponašanje [1], [7] itd. Slična je situacija i sa terminom klasa [1].

Glavni problem sa definicijama u objektno orijentisanom programiranju jeste to što sve one imaju slobodnu formu i ne omogućavaju da se objekat i klasa definišu nezavisno jedan od drugog. U ovom radu ćemo izložiti konceptualne definicije klase, objekta i invarijante, koje se baziraju na već ustaljenom filozofskom terminu *konceptu* ili *pojmu*. Pojam predstavlja misao o bitnim karakteristikama jedinice posmatranja. Ovakvim pristupom možemo jasno objasniti šta je invarijanta klase, a šta invarijanta objekta, odnosno možemo uvesti klasifikaciju invarijanata u klasi, što je i glavna tema ovog rada. Analiza invarijanata zauzima centralno mesto kada se govori o analizi semantike klase, odnosno semantike objektno orijentisanog programa.

Dušan T. Malbaški, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (phone: 381-21-485-2421; e-mail: malbaski@uns.ac.rs).

Aleksandar D. Kupusinac (autor za kontakte), Fakultet tehničkih nauka, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (phone: 381-21-485-2441, e-mail: sasak@uns.ac.rs).

II. Objekat i klasa

Tokom projektovanja nekog informacionog sistema, objektno orijentisani programer će najviše vremena posvetiti modelovanju. Međutim, ovde treba napomenuti činjenicu da se svaki programer zapravo bavi mislima, a ne konkretnim učesnicima informacionog sistema. Zbog toga naše izlaganje će započeti definicijom koncepta:

- *Koncept* ili *pojam* je misao o bitnim karakteristikama predmeta [8],

pri čemu termin *predmet* treba shvatiti u najširem smislu, odnosno kao predmet posmatranja i/ili razmišljanja. Karakteristike predmeta možemo podeliti na bitne i nebitne. Misao o bilo kojoj karakteristici naziva se oznaka, a misao o bitnoj karakteristici naziva se **bitna oznaka** [8]. Nebitne karakteristike se mogu izvesti iz bitnih karakteristika. Npr. karakteristike pojma TROUGAO jesu *biti konveksni mnogougao* i *imati tri stranice*, dok jednakost visina kod jednakostraničnog trougla nije bitna karakteristika, jer sledi iz osobine jednakosti stranica.

Pojmove možemo klasifikovati na razne načine, ali za naše izlaganje važna je podela na **individualne** i **klasne pojmove**. Individualni pojmovi se odnose na individualne predmete, individualni predmeti koji imaju zajedničke oznake čine klasu, a misao o datoj klasi jeste klasni pojam. Npr. individualni pojmovi HAJDN, MOCART i BETOVEN jesu misli o poznatim kompozitorima kojima je zajedničko to da pripadaju periodu klasicizma i da je njihov stvaralački rad više ili manje vezan za grad Beč. Na osnovu zajedničkih oznaka, oni čine klasu, a misao o toj klasi jeste klasni pojam BEČKI KLASIČAR. Očigledno, individualni pojmovi mogu i ne moraju biti realni, dok klasni pojmovi nikada nisu realni. Npr. klasni pojam BEČKI KLASIČAR nije realan, dok individualni pojam MOCART jeste (kompozitor Mocart je postojao u periodu 1756-1791 god.). Međutim, i individualni i klasni pojam TROUGAO nisu realni. Metodom apstrakcije od klasnih pojmova se mogu dobiti još apstraktniji klasni pojmovi i tako se stvara hijerarhija klasa, koja po svojoj formi podseća na stablo. Npr. od klasnog pojma BEČKI KLASIČAR apstraktniji pojam je KOMPOZITOR, a od ovog apstraktniji pojam je UMETNIK itd.

Ustaljena je praksa da učenje objektno orijentisanog

programiranja počinje razmatranjem termina *entitet*. Po ISO definiciji, entitet je bilo koja konkretna ili apstraktna stvar koja postoji, koja je postojala ili je mogla postojati, uključujući i veze između ovih stvari. Autori ovog rada entitet tretiraju kao sinonim za jedinicu posmatranja i smatraju da nema potrebe stavljati ga u prvi plan, jer kao što je na početku ovog poglavlja rečeno, projektant informacionog sistema barata pojmovima, tj. mislima. Zbog toga u prvom planu našeg izlaganja nalazi se pojam.

Razmatranje svih bitnih oznaka pojma nije nimalo jednostavan posao i zbog toga se vidi opravdanost uvođenja domena problema. Za razliku od logičara, projektant softvera ne mora voditi računa o svim bitnim oznakama pojma, već samo o onima koje su relevantne, tj. o onima koje su od interesa za dati domen problema koji je predmet analize. Npr. projektant informacionog sistema poreske uprave će za pojam GRAĐANIN izabrati oznake *ime i prezime*, *matični broj*, *adresa*, *podaci o prihodima* i sl., ali sigurno neće izabrati oznake *visina* i *težina*, iako ih svaki građanin poseduje. S druge strane, projektant informacionog sistema zdravstvene ustanove će pored oznaka *ime i prezime*, *matični broj*, *adresa* i sl., sigurno izabrati i oznake *visina* i *težina*, jer ove osobine su neophodne lekaru prilikom određivanja terapije, ali neće izabrati oznaku *podaci o prihodima*. Uvodimo sledeću definiciju:

- Oznake pojma koje su od interesa u datom domenu problema zovu se **relevantne oznake pojma** [9], [10].

Sada možemo definisati model, pri čemu treba istaći da sâm termin model, kao homomorfna slika nečeg, ima upotrebu u raznim situacijama (npr. maketa zgrade predstavlja model zgrade koja će biti sagrađena). Međutim, softverski inženjer bavi se modelovanjem pojmova, tj. modelovanjem misli, pa ćemo zato sada definisati softversko modelovanje i softverski model pojma:

- Postupak izbora konačnog broja relevantnih oznaka pojma u odnosu na dati domen problema naziva se **softversko modelovanje**, a dobijeni konačni skup relevantnih oznaka naziva se **softverski model**.

Sada ćemo navesti konceptualne definicije objekta i klase, ali ćemo prvo skrenuti pažnju na to da ove definicije imaju dve velike prednosti u odnosu na razne definicije koje se mogu naći u literaturi iz objektno orijentisanog programiranja. Naime, konceptualne definicije su zasnovane na pojmovima (konceptima), tj. na dobro razrađenom i jasnom sistemu termina i njihovih značenja. Pored toga, konceptualne definicije klase i objekta su ravnopravne u semantičkom smislu, tj. klasa se ne definiše preko objekta niti obrnuto.

- Klasa objekata (kraće **klasa**) jeste softverski model klasnog pojma.

Oznake klasnog pojma ćemo podeliti u dve grupe: oznake u užem smislu i oznake tipa *sadrži klasni pojam*. Oznake u užem smislu ćemo ovde zvati **odlikama**. Klasni pojam može u svom sastavu da sadrži i druge klasne pojmove, a njih ćemo ovde nazvati **fragmentima**. Npr. klasni pojam DUŽ ima dva fragmenta (dva temena) koja jesu dva pojma TAČKA. Npr. klasni pojam AUTOMOBIL ima odliku *boja*. Odlike mogu biti deskriptivne, kao što su *boja*, *masa* i sl., ali i operacione, kao što su *mogućnost kretanja*, *mogućnost letenja* i sl. Sada ćemo izložiti i konceptualnu definiciju objekta:

- Objekat je softverski model individualnog pojma.

Očigledno, konceptualni pogled postavlja klasu i objekat u ravnopravni položaj. Razlika je samo u tome što klasa odgovara klasnom, a objekat individualnom pojmu. Klasa i objekat povezani su jednom pretpostavkom koja zapravo ima snagu postulata i glasi:

- Za svaki objekat postoji klasa koja poseduje sve njegove relevantne oznake i tada kažemo da objekat pripada datoj klasi.

Npr. svaki pojedinačni objekat *jednakokraki trougao* (sa konkretnim vrednostima dužina stranica) pripada klasi *Jednakokraki trougao*. Fragmenti i odlike klasnog pojma se pojavljuju u klasnoj varijanti, a fragmenti i odlike individualnog pojma u individualnoj varijanti. Npr. ako je odlika klase *boja*, tada kod individue se ona pojavljuje kao *bela* ili *zelena*.

Pojmovi mogu biti složeni što znači da njihovi fragmenti mogu imati svoje fragmente, a ovi opet svoje itd. Npr. udžbenik se sastoji od poglavlja, poglavlja sadrže pasuse, pasusi linije, a linije sadrže znake. Skup sastavljen od pojma, njegovih fragmenata, pa dalje njihovih fragmenata itd., uređen relacijom *biti fragment* čini **strukturu** takvog pojma. Ukoliko pojam ne sadrži fragmente, već samo odlike, tada kažemo da je takav pojam jednostavne strukture. Ukoliko pak pojam sadrži bar jedan fragment, tada kažemo da je takav pojam složene strukture. Na primer, pojam AUTOMOBIL sadrži fragment MOTOR, pa kažemo da ima složenu strukturu. Na primer, pojam TAČKA sadrži samo odlike, a to su vrednosti koordinata, pa kažemo da ima jednostavnu strukturu. S obzirom da je objekat model individualnog pojma, tada modelovanjem od strukture pojma dobijamo strukturu objekta. Slično, možemo govoriti o jednostavnoj i složenoj strukturi objekta.

Esencijalne osobine objekta su da ima identitet, stanje i ponašanje. Konceptualna definicija nije u koliziji sa navedenim esencijalnim osobinama. Naime, pošto svaki individualni pojam ima identitet koji ga jednoznačno određuje, ta osobina se preslikava i na objekat, kao njegov model, odnosno svaki objekat ima identitet koji ga jednoznačno određuje. Stanje objekta sa jednostavnom strukturom određuju njegove deskriptivne odlike. Stanje objekta sa složenom strukturom određuju njegove deskriptivne odlike, ali i stanja njegovih fragmenata. Ako sada ovaj zaključak razmotrimo sa druge tačke gledišta, tj. ako za objekat posmatramo skup stanja, tada zaključujemo da se deskriptivne odlike izvode iz stanja, tj. odlike su funkcije stanja. Najzad, došli smo do faze realizacije u nekom od objektno orijentisanih programskih jezika. U fazi realizacije deskriptivne odlike predstavljaće **podatke-članove** objekta, dok će fragmenti predstavljati **objekte-članove** objekta. Operacione odlike determinišu ponašanje objekta, koje će biti opisano u njegovim **metodama**.

Kada se iz klasnog pojma izuzmu oznake nivoa klase, dobija se skup oznaka (fragmenata i odlika) koje poseduje svaki individualni pojam vezan za tu klasu. Imajući u vidu definiciju objekta, logično sledi da objekti iste klase imaju istu strukturu i isto ponašanje. Klasa i objekat stoje u identičnom odnosu u kojem su tip podataka i promenljiva

u standardnim programskim jezicima. Npr. promenljive tipa *int* predstavljaju primerke (pojave) tog tipa, kao što objekti predstavljaju primerke (pojave, instance) svoje klase. Odavde je očigledno da konceptualna definicija klase i ovo izlaganje nisu u koliziji sa ključnim aspektima klase koji su navedeni u uvodnom delu ovog rada.

Najzad, u literaturi iz objektno orijentisanog programiranja često se koristi termin *atribut*, nažalost, ne na sasvim ispravan način. Atribut po definiciji označava bitnu oznaku. U ovom izlaganju pokazali smo da postoje deskriptivne i operacione oznake. Pojedini autori atributima nazivaju samo deskriptivne odlike, iako se suština objektne metodologije sastoji upravo u izjednačavanju svih odlika, kako deskriptivnih tako i operacionih. Npr. operaciona odlika *mogućnost letenja* je bitna oznaka pojma AVION. Naravno, ima autora koji eksplicitno navode da atributi obuhvataju i deskriptivne i operacione bitne oznake [11].

III. INVARIJANTA

Od svih oznaka pojma, i to kako deskriptivnih, tako i operacionih, uočavamo one **invarijantne**. Npr. za pojam TROUGAO, pri čemu je ovde svejedno da li se radi o individualnom ili klasnom pojmu, oznaka $a + b > c$ je invarijantna, gde su a, b i c deskriptivne oznake koje predstavljaju dužine stranica, kao i oznaka $\alpha + \beta + \gamma = 180^\circ$, gde su α, β i γ deskriptivne oznake koje predstavljaju unutrašnje uglove trougla. Očigledno, bez obzira na promene (transformacije) koje pojam pretrpi, invarijantna oznaka je uvek važeća. Očigledno, postoji beskonačno mnogo invarijantnih oznaka, a sve one zajedno grade **suštinu** ili **esenciju pojma**.

Modelovanjem se bira konačan skup relevantnih oznaka, i to kako deskriptivnih, tako i operacionih. Ujedno, to povlači i izbor svih invarijantnih oznaka koje su relevantne za dati model. Takve oznake ćemo zvati **invarijantnim relevantnim oznakama**, a očigledno može ih biti beskonačno mnogo. Sve invarijantne relevantne oznake opisuju suštinu pojma u datom domenu problema. Npr. rekli smo da su oznake $a + b > c$ i $\alpha + \beta + \gamma = 180^\circ$ invarijantne, međutim, ukoliko modelujemo trougao tako da model čine jedino dužine stranica a, b i c, tada oznaka $a + b > c$ je relevantna invarijantna oznaka, dok $\alpha + \beta + \gamma = 180^\circ$ nije. U fazi realizacije dobijamo objekat, odnosno klasu, sa konačnim apstraktnim skupom stanja. Međutim, invarijantna relevantna oznaka pojma i dalje ostaje važeća, samo ovaj put u obliku restrikcije nad usvojenim konačnim skupom apstraktnih stanja. Sada ćemo definisati invarijantu u objektu i invarijantu u klasi:

- **Invarijanta u objektu** predstavlja restrikciju invarijantne relevantne oznake individualnog pojma na usvojenom skupu apstraktnih stanja.
- **Invarijanta u klasi** predstavlja restrikciju invarijantne relevantne oznake klasnog pojma na usvojenom skupu apstraktnih stanja.

Na osnovu činjenice da invarijantna oznaka može biti kako deskriptivna, tako i operaciona, nakon modelovanja, dobijamo nekoliko vrsta invarijanata [12]:

- **Invarijante polja** – potiču od invarijantnih deskriptivnih relevantnih oznaka, a nakon modelovanja predstavljaju relaciju definisanu nad poljima. Npr. u prethodnom primeru $a + b > c$ je invarijanta polja, gde polja a, b i c predstavljaju dužine stranica trougla.
- **Funkcionalne invarijante** – potiču od invarijantnih operacionih relevantnih oznaka, a nakon modelovanja povezuju metode u smislu primene. Npr. uzastopna primena metoda *push* i *pop* ostavlja stek u stanju u kojem je bio pre primene.
- **Invarijante odnosa** – tipičan primer jeste kardinalitet veza između pojmova, pa tako npr. kardinalitet veze između pojmova DUŽ i TAČKA jeste 1:2.
- **Mešoviti oblici**.

Posmatrajmo sada objekat jednostavne strukture, koji sadrži samo podatke-članove nekog tipa. Npr. ako podatak-član a predstavlja dužinu stranice trougla tipa *realan broj*, postavlja se pitanje – da li taj tip odgovara odgovarajućoj relevantnoj oznaci pojma? Iako sve deluje logično, nažalost, odgovor je odričan, jer dužina stranice trougla može biti jedino tipa *dužina*. Takav tip, nažalost, ne postoji ni u jednom programskom jeziku, već moramo improvizovati i reći da je podatak-član a tipa *pozitivan realan broj*. Međutim, tu nije kraj problemu, jer postavlja se pitanje – da li bilo koja tri pozitivna realna broja predstavljaju dužine stranica trougla? Opet je odgovor odričan, jer ta tri broja moraju zadovoljavati teoremu o nejednakosti dužina stranica trougla. Drugim rečima, invarijantna relevantna oznaka pojma TROUGAO mora se očuvati. Sada je potpuno jasan problem, sa jedne strane imamo zahteve koji potiču od sâme prirode oznaka pojma, a sa druge strane imamo realnost koju nameće deskriptivna moć programskih jezika. Zbog toga, potpuno opravdano je podeliti konačni skup apstraktnih stanja na dva disjunktna podskupa, a to su **skupovi validnih** i **invalidnih stanja**. Na primer, kada je u pitanju trougao, stanje (3, 4, 5) je validno, dok stanja (−3,−4,−5) i (5, 1, 1) su invalidna.

Invarijanta jeste svaki predikat koji je tačan u svakom validnom stanju, dok u invalidnom stanju može i ne mora biti tačan. Posebno, predikat koji jednoznačno razdvaja skup validnih od skupa invalidnih stanja, zvaćemo **stroga invarijanta**. Drugim rečima, stroga invarijanta jeste svaki predikat koji je tačan u svakom validnom stanju, a netačan u svakom invalidnom stanju. Invarijanata i strogih invarijanata ima beskonačno mnogo. Međutim, pošto sve stroge invarijante jednoznačno razdvajaju skupove validnih i invalidnih stanja, zaključujemo da su sve stroge invarijante jedinstvene do nivoa ekvivalencije.

Do sada smo govorili o invarijantama koje potiču od invarijantnih relevantnih oznaka pojma, tj. potiču iz domena problema. Međutim, postoje i **invarijante koje nastaju u fazi realizacije**, iako ih ne mora biti u domenu problema. Npr. kod steka koji je realizovan sekvencijalno sa kapacitetom C postoji ograničenje da ne može imati više elemenata od C, dok kod steka koji je realizovan spregnuto takvo ograničenje ne postoji.

Centralno mesto u analizi semantike objektno orijentisanog programa zauzima analiza semantike klase, konkretnije analiza invarijanata koja se može kretati u dva smera:

- ***as prescribed*** – invarijanta je unapred zadata, a korektnost ponašanja objekta, odnosno klase se dokazuje u odnosu na tako zadatu invarijantu [6],
- ***as described*** – smatra se da se objekat, odnosno klasa korektno ponašaju, pa se iz semantičkog opisa metoda izvodi invarijanta [12].

Značaj invarijante u analizi semantike klase je najlepše opisao Bertrand Mejer [6]: „Pojam invarijante, po mom mišljenju, jedan je od najsvetlijih koncepata koji mogu da se nauče iz objektno orijentisane metodologije. Jedino kada izvedemo invarijantu (klase koju pišemo) ili kada pročitamo i razumemo invarijantu (klase koju je neko drugi napisao), tada zaista osećamo da znamo koja je namena klase."

Možemo uočiti da se invarijante u osnovi dele na *invarijante objekta* i na *invarijante klase*. Invarijanta objekta podrazumeva da se sve metode aktiviraju preko jednog i samo jednog objekta-predstavnika (tzv. *this*) i opisuju sva validna stanja u kojima se taj objekat može naći. Invarijanta objekta može biti *čista* kada su sva polja i metode nestatičke, odnosno *mešovita* kada među poljima i metodama ima i statičkih. Invarijanta klase opisuje sva validna stanja u kojima se može naći data klasa. Takođe, invarijanta klase može biti *čista* (promene stanja vrše se statičkim modifikatorima, konstruktorima i destruktorom) ili *mešovita* (promene stanja izazivaju svi modifikatori, konstruktori i destruktor). Konačno, invarijanta može biti *parcijalna* kada interpretirani prostor stanja obuhvata samo neka polja i *totalna* kada ih obuhvata sva.

IV. ZAKLJUČAK

U ovom radu prikazali smo konceptualne definicije klase, objekta i invarijante. Ove definicije su značajne zbog toga što omogućavaju da se klasa i objekat razmatraju nezavisno. Zahvaljujući ovim definicijama moguće je jasno definisati šta je softverski model i softversko modelovanje, što je, takođe, prikazano u ovom radu. Analiza invarijanata u klasi zauzima centralno mesto kada se razmatra semantika klase, a sâmim tim i semantika objektno orijentisanog programa. Analiza invarijanata može biti *as prescribed*, gde je invarijanta zadata i *as described*, gde se invarijanta izvodi iz klase. U ovom radu smo prikazali konceptualne definicije invarijante objekta i invarijante klase, kao i klasifikaciju invarijanata, što je temelj analize invarijanata.

LITERATURA

[1] D. Malbaški, *Objektno orijentisano programiranje kroz programski jezik C++*. Novi Sad: Fakultet tehničkih nauka, 2008.

[2] D. Marković, *Filozofski osnovi nauke.* Beograd: Prosveta, 1994.

[3] D. Malbaški and D. Obradović, "On Some Basic Concepts in Object Orientation", *6th Balcan Conference on Operational Research*, Thessaloniki, 2002.

[4] M. Smith and S. Tockey, *An Integrated Approach to Software Requirements Definition Using Objects*. Scattle WA: Boeing Commercial Airplane Suport Division, 1988.

[5] B. Cox, *Object Oriented Programming: An Evolutionary Approach.* Reading MA, Addison-Wesley, 1986.

[6] B. Meyer, *Object-Oriented Software Construction, (2nd edition).* Prentice Hall, 1997.

[7] D. Malbaški, *Objekti i objektno programiranje kroz programske jezike C++ i Pascal.* Novi Sad: Fakultet tehničkih nauka, 2006.

[8] G. Petrović, *Logika.* Zagreb: Školska knjiga, 1981.

[9] A. Kupusinac i D. Malbaški, „Invarijanta klase u objektno orijentisanom programiranju i njena primena", *15. TELFOR*, Beograd: Društvo za telekomunikacije, Beograd: 20-22. novembar, 2007, str. 589-592, ISBN 978-86-7466-301-1.

[10] A. Kupusinac, „Invarijanta klase u objektno orijentisanom programiranju", magistarska teza, Novi Sad: Fakultet tehničkih nauka, 2008.

[11] M. Amadi and L. Cardelli, *A Theory of Objects*, New York: Springer-Verlag, 1996.

[12] D. Malbaški, „O invarijantama u klasi", Tech. Rep. 021-21/24, Novi Sad: Fakultet tehničkih nauka, 2010.

ABSTRACT

The great expansion and fast development of the object oriented programming resulted in unclear meaning of the concept and terms. Even fundamental concepts, such as class, object and invariant exhibit the same problem. The collision frequently exists between the terminologies of object oriented programming and other sciences. In this paper we introduce conceptual definitions of class, object and invariant and describe classification of invariants in class. Key motivation for this research is a wish to introduce a clear and consistent system of concepts and terms in object oriented programming.

CLASSIFICATION OF INVARIANTS IN CLASS

Dušan T. Malbaški and Aleksandar D. Kupusinac