Programski jezik Decaf

Decaf.
Decaf je revizija jezika SOOP koji su razvili Maggie Johnson i Steve Freund. Decaf je strongly-typed, objektno-orijentisani jezik sa podrškom za nasleđivanje (inheritance) i učaurenje (encapsulation). Ime veoma mnogo sličnosti sa jezicima C, C++ i Java, ali se u potpunosti ne poklapa ni sa jednim od njih.

Leksičke karakteristike
Spisak ključnih riječi (keywords), koje se rezervisane (ne mogu se koristiti kao identifikatori ili se redefinisati).

void int double bool string class interface null this extends implements for while if else return break New NewArray Print ReadInteger ReadLine

Identifikator je niz slova, cifara i donjih crta (underscores) koji počinje slovom. Decaf je "case sensitive", tj. if je ključna riječ, ali je IF identifikator; binky i Binky su 2 različita identifikatora. Identifikatori mogu biti najviše 31 karakter dugi.
Bjeline ("whitespace", tj. space, tab i newline) razdvajaju tokene, ali se inače ignorišu.
Ključne riječi i tokeni moraju biti razdvojeni bjelinom ili tokenom koji nije ni ključna riječ ni identifikator. Na primjer, ifintthis je jedan identifikator, a ne 3 ključne riječi, dok if(23this predstavlja 4 tokena.
Boolean konstanta je ili true ili false, i ove riječi su rezervisane.
Cjelobrojna (integer) konstanta može biti decimalna (osnova 10) ili heksadecimalna (osnova 16). Decimalna konstanta je niz cifara (0-9). Heksadecimalna konstanta mora početi sa 0X ili 0x i slijedi niz heksadecimalnih cifara (decimalne cifre i slova a-f, velika ili mala). Primjeri ispravnih cjelobrojnih konstanti: 8, 012, 0x0, 0X12aE.
Double konstanta je niz cifara za kojim slijedi tačka, za kojom slijedi moguće prazan niz cifara. Otuda, .12 nije validna double konstanta, ali su validne 0.12 i 12. Double može imati opiconalni eksponent, na primjer, 12.2E+2. U ovoj notaciji, tačka je obavezna, znak je opcionalan (ako ga nema, podrazumijeva se +), i E može biti malo ili veliko slovo. 12E+2 nije validno, ali 12.E+2 jeste. Vodeće nule u mantisi i eksponentu su dozvoljene.
String konstanta je niz karaktera između dvostrukih navodnika, i može sadržati bilo koje karaktere osim newline i dvostrukih navodnika. String mora početi i završiti se u istoj liniji
Operatori i znaci interpunkcije su:
+ - * / % \ < <= > >= = == != && || ! ; , . [] [ ] ( ) { }

Komentar u jednoj liniji počinje sa //. Komentar u više linija počinje sa /* i završava prvim sledećim */. Svaki simbol je dozvoljen u komentarima, osim niza */ koji završava komentar. Nije moguće ugnježdavanje komentara.

Gramatika
Gramatika je data u varijanti proširene BNF. Meta-notacija je upotrebljena za:
x (courier font) znači da je x terminalni simbol tj. token. Imena terminalnih simbola su napisana malim slovima osim za ključne riječi. *y* (italic) znači y je neterminalni simbol. Sva imena netreminala počinju velikom slovom.

`x`	(courier font) znači da je x terminal tj. token . Imena terminala su napisana malim slovima osim u slučaju ključne riječi.
x	(italic) znači da je x neterminalni simbol. Imena terminala su napisana velikim slovima.
⟨ *x* ⟩	znači nula ili jedno pojavljivanje x, tj. x je opcionalno
x *	znači nula ili više pojavljivanja x
x +	znači jedno ili više pojavljivanja x
x +,	lista x-ova razdvojena zarezima, jedan ili više njih (zarez se pojavljuje samo između x-ova)
|	razdvajanje pravila gramatike
ε	označava epsilon (prazna riječ, odsustvo tokena)

Operatore predstavljamo leksimom koja ih označava, kao što je + ili !=, za razliku od tokena koji vraća skener (T_NotEqual, itd.).

Program ::= *Decl*+

Decl ::= *VariableDecl* | *FunctionDecl* | *ClassDecl* | *InterfaceDecl*

VariableDecl ::= *Variable* `;`

Variable ::= *Type* `ident`

Type ::= `int` | `double` | `bool` | `string` | `ident` | *Type* `[]`

FunctionDecl ::= *Type* `ident (` *Formals* `)` *StmtBlock* |
`void ident (` *Formals* `)` *StmtBlock*

Formals ::= *Variable*+`,` | ε

ClassDecl ::= `class ident` ⟨`extends ident`⟩ ⟨`implements ident`+`,`⟩ `{` *Field** `}`

Field ::= *VariableDecl* | *FunctionDecl*

InterfaceDecl ::= `interface ident {` *Prototype** `}`

Prototype ::= *Type* `ident (` *Formals* `) ;` | `void ident (` *Formals* `) ;`

StmtBlock ::= `{` *VariableDecl** *Stmt** `}`

Stmt ::= ⟨*Expr*⟩ `;` | *IfStmt* | *WhileStmt* | *ForStmt* |
BreakStmt | *ReturnStmt* | *PrintStmt* | *StmtBlock*

IfStmt ::= `if (` *Expr* `)` *Stmt* ⟨`else` *Stmt*⟩

WhileStmt ::= `while (` *Expr* `)` *Stmt*

ForStmt ::= `for (` ⟨*Expr*⟩ `;` *Expr* `;` ⟨*Expr*⟩ `)` *Stmt*

ReturnStmt ::= `return` ⟨*Expr*⟩ `;`

BreakStmt ::= `break ;`

PrintStmt ::= `Print (` *Expr*+`,` `) ;`

Expr ::= *LValue* `=` *Expr* | *Constant* | *LValue* | `this` | *Call* | `(` *Expr* `)` |
Expr `+` *Expr* | *Expr* `-` *Expr* | *Expr* `*` *Expr* | *Expr* `/` *Expr* |
Expr `%` *Expr* | `-` *Expr* | *Expr* `<` *Expr* | *Expr* `<=` *Expr* |
Expr `>` *Expr* | *Expr* `>=` *Expr* | *Expr* `==` *Expr* | *Expr* `!=` *Expr* |
Expr `&&` *Expr* | *Expr* `||` *Expr* | `!` *Expr* | `ReadInteger ( )` |
`ReadLine ( )` | `New ( ident )` | `NewArray (` *Expr* `,` *Type* `)`

LValue ::= `ident` | *Expr* `.` `ident` | *Expr* `[` *Expr* `]`
Call ::= `ident` `(` *Actuals* `)` | *Expr* `.` `ident` `(` *Actuals* `)`
Actuals ::= $Expr^{+,}$ | ε
Constant ::= `intConstant` | `doubleConstant` | `boolConstant` | `stringConstant` | `null`

Struktura programa
Decaf program je niz deklaracija, gdje svaka deklaracija uvodi promjenljivu, funkciju, klasu ili interfejs. Termin deklaracija (*declaration*) predstavlja naredbu kojom se daje identitet imenu, dok definicija označava potpun opis i tijelo f-ije. U našem slučaju, deklaracija i definicija su jedno te isto. Program mora imati globalnu f-iju main tipa void bez argumenata koja je startna tačka za izvršavanje programa.

Opsezi (scoping)
Decaf podržava nekoliko nivo opsega. Deklaracije na najvišem nivou su u globalnom opsegu (global scope). Svaka deklaracija klase ima vlastiti klasni opseg (class scope). Svaka funkcija ima lokalni opseg za listu parametara i još jedan opseg za tijelo f-ije. Par vitičastih zagrada unutar tijela f-ije postavlja novi ugnježdeni opseg za lokalni opseg. Unutrašnji opsezi imaju privilegiju u odnosu na spoljašnje. Na primjer, promjeljiva definisana u opsegu f-ije prekriva promjenljivu sa istim imenom definisanu na globalnom nivou. Isto važi i za funkcije.

- svi identifikatori moraju biti deklarisani
- po ulasku u opseg, sve deklaracije u opsegu su odmah vidljive
- identifikatori unutar opsega moraju biti jedinstveni (na primjer, ne mogu postojati dvije f-ije sa istim imenom, globalna promjenljiva i klasa sa istim imenom, itd.)
- identifikatori redeklarisani u unutrašnjem opsegu prekrivaju verziju iz spoljašnjeg opsega
- deklaracije iz globalnog opsega dostupne su svuda u programu
- deklaracije u zatvorenim opsezima nisu dostupne

Napomena o vidljivosti i vezi sa leksičkom strukturom: Kao u Javi, kompajer radi u više prolaza: prvi prolaz prikuplja informacije i kreira drvo parsiranja (parse tree). U drugom prolazu odrađujemo semantičku analizu. Prednost dvoprolaznog kompajlera je u tome što su sve deklaraciie dostupne u istom opsegu čak i prije leksičke tačke u kojoj su deklarisani. Na primjer, metodi klase mogu referencirati promjenljive koje su kasnije deklarisane, klase i podklase mogu biti pisane bilo kojim redom, itd. Ovo pravilo važi za sve deklaracije u svim opsezima.

Tipovi
Ugrađeni osnovni tipovi (built-in) su integer, double, bool, string i void. Decaf dopušta i imenovane tipove – imena klasa ili interfejsa. Nizovi mogu biti kreirani od bilo kojih ne-void tipova, uključujući nizove nizova.

Promjenljive
Promjenljiva može biti deklarisana da bude bilo kog ne-void osnovnog tipa, tipa niz ili imenovanog tipa. Promjenljive deklarisane van funkcija su globalne. Promjenljive deklarisane unutar klase, ali van metoda klase imaju klasni opseg. Promjenljive deklarisane u listi

parametara ili tijelu f-ije imaju lokalni opseg. Promjenljiva je vidljiva od ulaska u opseg pa sve do izlaska iz njega.

Nizovi

Decaf nizovi su homogene linearno indeksabilne kolekcije, koje su implementirane kao reference (pokazivači, pointeri). Deklarišu se bez informacije o broju članova i alociraju se sa heap-a pomoću built-in operatora NewArray.

- nizovi mogu biti bilo kog ne-void tipa, imenovanog tipa ili niza
- NewArray(N, type) alocira novi niz tipa type i sa N elemenata; N mora biti pozitivan integer inače se dobija runtime greška
- broj elemenata u nizu je alociran i ne može se mijenjati poslije alociranja
- nizovi imaju specijalnu sintaksu arr.length() (kao u Javi) za određivanje broja elemenata niza
- indeksiranje može biti primjenjeno na promjenljive tipa niza
- indeksi niza su u granicama od 0 do length-1
- indeks niza mora biti cio broj
- runtime error ako je indeks niza van granica
- nizovi mogu biti argumenti f-ija i vraćeni kao rezultat f-ije. Niz se predaje po vrijednosti, ali je on referenca, pa promjene u nizu su vidljive u pozivačkoj f-iji.
- dodjeljivanje nizu je "plitko" (shallow) (tj. dodjeljivanje jednog niza drugom nizu samo kopira referencu)
- poređenje jednakosti nizova (== i !=) vrši poređenje referenci

Stringovi

Programi mogu uključivati string konstante, učitavati stringove pomoću funkcije ReadLine, upoređivati stringove i štampati ih. Nema podrške za kreiranje i manpulisanje stringovima, konverziju u druge tipove i slično. Implementirani su kao reference.

- ReadLine() učitava niz karaktera do nailaska na oznaku newline
- dodjeljivanje stringova je plitko (kopira se samo referenca)
- stringovi mogu biti argumenti i rezultat f-ija
- poređenje jednakosti stringova (== i !=) upoređuje nizove karaktera u case-sensitive vidu

Funkcije

Deklaracija f-ije uključuje ime f-ije i pridruženu signaturu tipa (*type signature*) koja uključuje tip rezultata i broj i tipove formalnih argumenata.

- f-ije su ili globalne ili deklarisane u opsegu klase (metodi); ne mogu biti ugnježdene unutar drugih f-ija
- mogu imati nula ili više formalnih parametara
- formalni parametri mogu biti bilo koji ne-void tip, niz ili imenovani tip
- identifikatori u listi formalnih parametara moraju biti jedinstveni
- formalni paramateri su u posebnom opsegu različitom od lokalnih promjenljivih f-ije
- tip rezultata može biti bilo koji bazni tip, niz ili imenovani tip. void označava da f-ija ne vraća vrijednost
- nije dozvoljen overloading f-ija (upotreba istog imena f-ije sa različitim signaturama)
- ako f-ja ne vraća void, vrijednost koja se vraća mora biti tipa koji je kompatibilan sa datim tipom

- ako je tip f-ije void, može se koristiti samo prazna return naredba
- dozvoljene su rekurzivne f-ije

Pozivanje f-ija (function invocation)
Pozivanje f-ije uključuje predaju vrijednosti argumenata od pozivača ka pozvanoj f-iji, izvršavanje tijela pozvane procedure i povratak u pozivačku f-iju, sa eventualnim predavanjem rezultata. Po pozivu f-ije, izračunavaju se stvarni parametri i vezuju se za formalne parametre. Svi parametri i rezultat se predaju po vrijednosti.
- broj stvarnih argumenata mora se poklopiti sa brojem formalnih argumenata
- tip svakog od stvarnih argumenata mora biti kompatibilan sa tipom formalnog argumenata
- stvarni argumenti se izračunavaju slijeva udesno.
- povratak u pozivačku f-iju sa return ili dostizanjem kraja f-ije
- poziv f-ije izračunava se po deklarisanom povratnom tipu

Klase (Classes)
Deklarisanje klase kreira novo ime tipa i novi klasni opseg (class scope). Deklaracija klase je lista polja, gdje je polje ili promjenljiva ili f-ija. Promjenljive klase se ponekad zovu instance promjenljive ili članovi klase (member data), a f-ije se nazivaju metodama ili funkcijma članicama(methods ili member functions).
Decaf nas prisiljava na enkapsulaciju tako što su sve promjenljive privatne (vidljive i klasi i njenim podklasama, u Javi je to protected) i svi metodi su public (dostupni na globalnom nivou), pa je jedini način pristupa promjenljivim klase preko metoda.
- sve deklaracije klase su globalne
- sve klase moraju imati jedinstvena imena
- ime polja može biti upotebljeno najviše jednom u klasnom opsegu
- polja deklarišemo u proizvoljnom redosledu
- instance promjenljive mogu imati bilo koji ne-void tip, niz ili imenovani tip
- upotreba "this." je opcionalna ako pristupamo poljima u metodu

Objekti
Promjenljiva čiji je tip imenovani tip je objekat ili instanca imenovanog tipa. Implementirani su kao reference i dinamički se alociraju sa heap-a primjenom ugrađenog operatora New.
- ime tipa u deklaraciji objekta mora biti ime klase ili interfejsa
- argument za New mora biti ime klase
- operator "." se koristi za dostup poljima (promjenljivim i metodama) objekta
- za poziv metoda oblika expr.method(), tip expr mora biti klasa ili intefejs T i method mora biti metod u T
- za pristup promjenljivoj expr.var, tip expr mora biti klasa T, var mora biti ime jedne od promjenljivih u opsegu za T ili njenih podklasa
- napomena: unutar klase možete pristupati privatnim promjenljivim te klase, ali ne i drugim klasama koje nisu povezane sa datom
- dodjeljivanje objekata je "plitko" (kopira se samo referenca)
- objekti mogu biti argumenti i rezultat f-ija i predaju se po vrijednosti (kako su oni reference, svaka promjena promjenljivih objekta je vidljiva pozivačkoj f-iji).

Nasleđivanje (inheritance)

Decaf podržava jednostruko nasleđivanje (single inheritance) (kao Java, ne kao C++), tako što izvedena klasa (podklasa) nadgrađuje (extend) osnovnu klasu dodavanjem novih polja ili prepisivanjem postojećih metoda novom definicijom (overriding). Semantika izraza A extends B (A nasleđuje B) je da A ima sva polja kao i B i još neka dodatna polja.
Podklasa može dati novu definciju naslijeđenom metodu, ali naslijeđena verzija mora odgovarati originalnoj po tipu rezultata i tipovima parametara. Podržana je automatska konverzija nagore (upcast), pa se objekat podklase može koristiti u svim situacijama gdje se može očekivati objekat osnovne klase. Za sve metode se dinamički određuje klasa kojoj pripadaju (dynamically dispatched) (tj. kao virtual metodi u C++). Kompajer ne može odredti tačnu adresu metoda koji bi trebalo pozvati u trenutku kompajliranja (na primjer, pozivanje metoda koji je ponovo definisan (overridden) na objektu koji pripada nekoj nadklasi), pa se posao određuje u trenutku izvršavanja provjeravanjem tabele metoda pridružene svakom objektu.

- ako je specificirana, nadklasa mora biti ispravno deklarisana klasa
- sva polja nadklase se nasleđuju u podklasi
- podklasa ne može ponovo definisati naslijeđene promjenljive
- podklasa može ponovo definisati naslijeđene metode, ali se oni moraju poklapati sa originalom po tipu rezultata i tipovima parametara
- nema overloading-a: klasa ne može ponovo koristiti isto ime za metod sa različitom signaturom
- instanca podklase je kompatibilna sa nadklasom. Obrnuto ne važi. Ovo pravilo važi za više nivoa nasleđivanja
- provjera pripadnosti polja objektu se vrši u vrijeme kompajliranja (tj. ako smo objekat klase Cow dodijelili promjenljivoj tipa Animal, ne možemo pristupiti poljima koja postoje samo u klasi Cow)
- ne postoje podtipovi nizova: ako T2 extends T1, niz T2[ ] nije kompatibilan nizu T1[ ] (za razliku od Jave).

Intefejsi (Interfaces)

Decaf podržava podtipove dozvoljavajući klasi da implementira jedan ili više intefejsa. Dekalarcija interfejsa se sastoji od liste prototipova f-ija bez implementacije. Ako deklaracija klase naglasi implementaciju interfejsa, ona mora da obezbijedi implementaciju za svaki metod naveden u interfejsu (za razliku od Jave i C++ nema apstraktnih klasa). Svaki metod mora da odgovara originalnom metodu po tipu rezultata i tipovima parametara. Decaf podržava automatski upcasting za interface tipove.

- svaki interfejs nabrojan u deklaraciji klase mora biti ime pravilno deklarisanog interfejsa
- sve metodi interfejsa moraju biti implementirani u klasi koja implementira interfejs
- svaki metod mora da odgovara originalnom metodu po tipu rezultata i tipovima parametara
- deklaracija klase mora formalno da naglasi da želi implementirati interfejs (nije dovoljno samo implementirati metode interfejsa)
- instanca klase je kompatibilna sa svakim intefejsom koji implementira i može biti zamijenjena u izrazu koji zahtijeva interface tip. Obrnuto ne važi.
- podklasa nasleđuje interfejse nadklase
- provjera pripadnosti polja objektu se vrši u vrijeme kompajliranja

- ne postoje podtipovi nizova: ako T2 implementira T1, niz T2[ ] nije kompatibilan nizu T1[ ] (za razliku od Jave).

Ekvivalencija tipova i kompatibilnost (Type equivalence and compatibility)
Decaf je (uglavnom) strongly-typed jezik: određeni tip je pridružen svakoj promjenljivoj i promjenljiva sadrži vrijednosti koje pripadaju intervalu. Ako je tip A ekvivalentan tipu B, izraz bilo kog tipa može biti upotrebljen umjesto onog drugog u svakoj situaciji. Dva osnovna tipa su ekvivalentna akko su isti tip. Dva niza su ekvivalentna akko imaju isti tip elemenata (koji ponovo može biti niz, što implicira rekurzivnu definiciju strukturalne ekvivalencije). Dva imenovana tipa su ekvivalentna akko imaju isto ime (tj. ekvivalencija imena nije strukturalna). Kompatibilnost tipova (type compatibility) je više ograničavajuća. Ako je tip A kompatibilan tipu B, tada se izraz tipa A može upotrebiti svuda gdje se očekuje izraz tipa B. Ništa se ne govori za suprotan smjer. Dva ekvivalentna tipa su kompatibilna u oba smjera. Podklasa je kompatibilna sa nadkalsom, ali obrnuto ne važi. Klasa je kompatibilna sa svakim interfejsom kojeg impelementira. Null tip je kompatibilan sa svim imenovanim tipovima. Operacije kao dodjeljivanje i predaja parametara su dozvoljene ne samo za ekvivalentne tipove već i za kompatibilne tipove.

Dodjeljivanje (Assignment)
Za osnovne tipove, Decaf koristi value-copy semantiku; LValue = Expr kopria vrijednost koja se dobija izračunavanjem Expr u lokaciju označenu sa LValue. Za nizove, stringove i objekte, koristi se reference-copy semantika; LValue = Expr znači da LValue sadrži referencu na objekta koji je rezultat izračunavanja Expr (tj. kopira se pokazivač na objekat a ne sam objekat).
- LValue mora biti lokacija promjenljive koja može da primi vrijednost (assignable variable location)
- desna strana naredbe dodjeljivanja mora bitia kompatibilna sa tipom lijeve strane
- null se može dodijeliti samo imenovanom tipu
- ispravno je dodjeljivanje formalnom parametru u f-iji, takvo dodjeljivanje utiče samo na opseg f-ije

Kontrolne strukture
Decaf kontrolne strukture su bazirane na C/Java verzijama i ponašaju se veoma slično.
- else je upareno sa najbližim prethodnim if
- izrazi u testiranju u if, while i for naredbama moraju biti tipa bool
- break naredba može biti samo unutar while ili for petlje
- vrijednost u return naredbi mora biti kompatibilna sa povratnim tipom f-ije u kojoj je return

Izrazi (Expressions)
Decaf ne dozvoljava konverziju tipova unutra izraza (tj. nije moguće sabirati integer i double, koristiti integer kao bool, itd.).
- konstante se izračunavaju u svoju vrijednost (true, false, null, integers, doubles, string literali)
- this je vezano za objekat u klasnom opsegu, greška van klasnog opsega

- dva operanda za binarne aritmetičke operatore (+,-,*,/, and %) moraju biti ili oba int ili oba double. Rezultat je istog tipa kao i operandi.
- operand unarnog minusa mora biti int ili double. Rezultat je istog tipa kao i operand.
- dva operanda za binarne operatore poređenja (<, >, <=, >=) moraju biti ili oba int ili oba double. Rezultat je tipa bool.
- dva operanda za binarne operatore jednakosti (!=, ==) moraju biti ekvivalentnih tipova (dva int-a, dva niza elemenata tipa double, itd.) (ima izuzetaka od ovog pravila kod objekata). Rezultat je tipa bool.
- dva operanda za binarne operatore jednakosti (!=, ==) mogu biti dva objekta ili objekat i null. Tipovi objekata moraju biti kompatibilni. Rezultat je tipa bool.
- operandi svih binarnih i unarnih logičkih operatora (&&, || , and !) moraju biti tipa bool. Rezultat je tipa bool.
- logički && i || nemaju "short-circuit"; oba izraza se izračunavaju prije određivanja rezultata
- operandi svih izraza izračunavaju se slijeva udesno

Prioritet operatora (Operator precedence), od najvišeg ka najnižem:
[. (indeksiranje niza i selekcija polja
! - (unarni -, logičko not)
* / % (množenje, dijeljenje, mod)
+ - (sabiranje. oduzimanje)
< <= > >= (poređenja)
== != (jednakost)
&& (logičko and)
|| (logičko or)
= (dodjeljivanje)
Svi binarni aritmetički operatori i oba binarna logička operatora su lijevo asocijativni. Dodjeljivanje i operatori poređenja nisu asocijativni (tj. ne možemo ih povezivati kao u C/C++ a=b=c ili a < b >= c (greška u parsiranju), ali će a < b == c biti OK). Zagradama se može promijeniti prioritet i/ili asocijativnost.

Standardne biblioteke (Standard library functions)
Decaf ima veoma malo rutina u standardnoj biblioteci koje omogućavaju prosti I/O i alokaciju memorije. Funkcije su Print, ReadInteger, ReadLine, New i NewArray.
- argumenti f-ije Print mogu biti string, int ili bool
- argument za New mora biti ispravno deklarisano ime klase
- prvi argument za NewArray mora biti cio broj, drugi bilo koji ne-void tip
- return tip za NewArray je niz elemenata tipa T, gdje je T tip specificiran drugim argumentom
- ReadLine() čita niz karaktera koje unosi korisnik, do nailaska na newline (ne uključuje newline)
- ReadInteger() čita liniju teksta koju unosi korisnik i konvertuje je u cio broj koristeći atoi (vraća 0 ako korisnik ne unese ispravan broj)

Decaf linking

Kako Decaf ne dozvoljava odvojene module ili deklaracije odvojene od definicije, osim semantičke analize nema mnogo posla oko povezivanja (linking). Jedan zadatak koji "linker" treba da uradi je provjera da li postoji deklaracija za globalnu f-iju main, što se može odraditi kao dio generisanja koda.

Provjere u vrijeme izvršavanja (Run time checks)

Postoje samo dvije provjere (može se nadgraditi)

- indeks niza mora biti u odgovarajućim granicama (0 . . arr.length() –1)
- veličina predata f-iji NewArray mora biti pozitivna

Ako dođe do greške u toku izvršavanja, štampa se odgovarajuća poruka i program završava rad.

Koje osobine nema Decaf!

Postoje mnoge osobine koje Java ili C/C++ prevodioci rade a ne postoje od Decaf-u:

- ne vodi računa o vrijednostima neinicijalizovanih vrijednosti ili vrijednostima tek alociranih objekata ili nizova
- ne provjerava upotrebu neinicijalizovanih promjenljivih
- ne detektuje da li f-ija koja je deklarisana da vraća vrijednost zaista to radi
- ne provjerava objekat ili niz prije alokacije
- ne detektuje poziv metoda ili pristup promjenljivoj null objekta
- ne otkriva nedostupan kod
- nema f-ije za dealokaciju memorije niti postoji "garbage collection"
- nema podrške za konstruktore i destruktore
- itd.