



پروژه‌ی درس طراحی کامپایلر

توضیحات زبان Decaf

نسخه ۱.۱

۱۷ مهر ۱۳۹۹

شما در پروژه‌ی این درس قرار است کامپایلری برای یک زبان برنامه‌نویسی شیء‌گرای ساده به اسم Decaf بنویسید. این زبان از نظر طراحی شباهت زیادی به زبان‌های C/C++/Java دارد. پس با کمی تلاش می‌توانید به آن مسلط شوید. البته توجه داشته باشید که دقیقاً منطبق بر هیچ‌یک از زبان‌های گفته شده نیست. امکانات این زبان برای آن که پیاده‌سازی کامپایلرش به یک کاپوس تبدیل نشود، ساده شده‌اند. با این حال باز هم به اندازه‌ی کافی قدرت‌مند است که بتوان با استفاده از آن برنامه‌های جالبی نوشت. این مستند جهت آشنایی شما با سینتکس و قواعد زبان Decaf نوشته شده. در طول پروژه مرتباً به آن نیاز پیدا خواهید کرد.

۱ بررسی واژه‌ای زبان

در لیست زیر می‌توانید کلیدواژه^۱ زبان را مشاهده کنید. آنها رزرو شده‌اند. که به این معنی است که توسط برنامه‌نویس به عنوان شناسه^۲ قابل استفاده نیستند.

void, int, double, bool, string, class, interface, null, this, extends, implements, for, while, if, else, return, break, continue, new, NewArray, Print, ReadInteger, ReadLine, dto, itod, btoi, itob, private, protected, public

یک شناسه، دنباله‌ای از حروف، اعداد و زیرخط^۳ است که با یک حرف شروع می‌شود. این زبان حساس به بزرگی و کوچکی حروف است (Case-sensitive).

فاصله‌ی سفید^۴ به عنوان جداکننده‌ی نشانه‌ها^۵ استفاده می‌شوند. و در غیر این صورت نادیده گرفته می‌شوند. کلیدواژه‌ها و شناسه‌ها باید توسط فواصل سفید و یا نشانه‌ای که نه کلیدواژه است و نه شناسه از هم جدا شوند. پس ifintthis یک شناسه است. ولی if(23this در scanner به عنوان ۴ نشانه شناسایی می‌شود.

یک ثابت بولین می‌تواند true یا false باشد. مانند کلیدواژه‌ها، این کلمات نیز رزرو هستند. یک ثابت صحیح^۶ می‌تواند در مبنای ۱۰ یا ۱۶ نمایش داده شود. یک ثابت در مبنای ۱۰، دنباله‌ای از ارقام است. ولی یک ثابت در مبنای ۱۶ با 0x یا 0X شروع می‌شود و به دنبالش دنباله‌ای از ارقام مبنای ۱۶ می‌آیند. دقت کنید که ارقام مبنای ۱۶ شامل حروف a تا f نیز می‌شوند که می‌توانند uppercase و یا lowercase نوشته شوند. برای مثال نشانه‌های زیر به عنوان ثوابت صحیح قابل قبول هستند:

8, 012, 0x0, 0X12aE

ضمناً ثابت صحیح می‌تواند leading zero داشته باشد، مانند 00123.

یک ثابت اعشاری^۷ دنباله‌ای از ارقام، یک نقطه به عنوان ممیز و مجدداً دنباله‌ای از ارقام است (ممکن است بعد ممیز رقمی نداشته باشیم). به این معنی که عدد زیر غیر قابل قبول است:

.12

و دو عدد زیر قابل قبول‌اند:

0.12, 12.

Keyword^۱
 identifier^۲
 underscore^۳
 Whitespace^۴
 tokens^۵
 integer^۶
 double^۷

همچنین یک عدد اعشاری می‌تواند یک بخش برای توان داشته باشد. مثلاً عدد زیر:

$12.2E + 2$

برای اعداد اعشاری که به صورت نمایش علمی نشان داده می‌شوند، حتماً نیاز به ممیز داریم. علامت بخش توان اختیاری است (اگر گذاشته نشود، مثبت در نظر گرفته می‌شود). همچنین E می‌تواند lowercase و یا uppercase باشد. پس عدد زیر غیرقابل قبول است:

$.12E + 2$

ولی عدد زیر قابل قبول است:

$12.E + 2$

در قسمت صحیح و همچنین بخش توان، ممکن است قبل عدد تعدادی صفر بیایند که تاثیری در مقدار عدد ندارند. همچنین توجه کنید در نمایش علمی یک عدد هیچ فاصله‌ای وجود ندارد.

یک رشته‌ی ثابت، دنباله‌ای از کاراکترها است که بین دو double-quote قرار دارند. رشته‌ها می‌توانند هر کاراکتری به جز double-quote و new-line را شامل شوند. پس یک ثابت رشته‌ای در یک خط شروع شده و در همان خط تمام می‌شود.

"this string is missing its close quote
this is not a part of the string above

عملگرها و نقطه‌گذاری‌ها^۸ که در این زبان قرار دارند:

+ - * / % < <= > >= == != || ! ; , . [] ()

یک کامنت تک‌خطی با // شروع می‌شود و تا آخر خط ادامه پیدا می‌کند. همچنین کامنت‌های چندخطی به صورت زیر هستند:

/*this is a comment
also the comment
still the comment*/

۲ گرامر زبان

گرامر زبان را در قالبی شبیه به BNF در زیر آورده می‌شود. قواعدی که در این گرامر استفاده شده‌اند:

نماد	معنی
x	به این معنی است که x یک ترمینال است. ترمینال‌ها اغلب به صورت lowercase نوشته شده‌اند.
x	به این معنی است که x یک nonterminal است. تمامی nonterminal ها با حروف بزرگ شروع شده‌اند.
< x >	به معنی صفر یا یک تکرار x است. مثلاً می‌تواند به معنای اختیاری بودن x باشد.
x*	به معنای صفر یا بیشتر تکرار x است.
x ⁺	به معنای یک یا بیشتر تکرار x است.
x ⁺ ,	به معنای یک یا بیشتر x است که با ویرگول از هم جدا شده‌اند (ویرگول‌ها فقط بینشان قرار دارند).
	حالت‌های مختلف production را از هم جدا می‌کند.
ε	به معنای واژه‌ی خالی است.

^۸punctuations

$Program ::= Decl^+$
 $Decl ::= VariableDecl \mid FunctionDecl \mid ClassDecl \mid InterfaceDecl$
 $VariableDecl ::= Variable;$
 $Variable ::= Type \text{ ident}$
 $Type ::= int \mid double \mid bool \mid string \mid ident \mid Type []$
 $FunctionDecl ::= Type \text{ ident } (Formals) StmtBlock \mid$
 $\quad void \text{ ident } (Formals) StmtBlock$
 $Formals ::= Variable^+, \mid \epsilon$
 $ClassDecl ::= class \text{ ident } < extends \text{ ident } > < implements \text{ ident}^+, > \{Field^*\}$
 $Field ::= AccessMode \text{ VariableDecl} \mid AccessMode \text{ FunctionDecl}$
 $AccessMode ::= private \mid protected \mid public \mid \epsilon$
 $InterfaceDecl ::= interface \text{ ident } \{Prototype^*\}$
 $Prototype ::= Type \text{ ident } (Formals); \mid void \text{ ident } (Formals);$
 $StmtBlock ::= \{VariableDecl^* Stmt^*\}$
 $Stmt ::= < Expr >; \mid IfStmt \mid WhileStmt \mid ForStmt \mid$
 $\quad BreakStmt \mid ContinueStmt \mid ReturnStmt \mid PrintStmt \mid StmtBlock$
 $IfStmt ::= if (Expr) Stmt < else Stmt >$
 $WhileStmt ::= while (Expr) Stmt$
 $ForStmt ::= for (< Expr >; Expr; < Expr >) Stmt$
 $ReturnStmt ::= return < Expr >;$
 $BreakStmt ::= break;$
 $ContinueStmt ::= continue;$
 $PrintStmt ::= print (Expr^+,);$
 $Expr ::= LValue = Expr \mid Constant \mid LValue \mid this \mid Call \mid (Expr) \mid$
 $\quad Expr + Expr \mid Expr - Expr \mid Expr * Expr \mid Expr / Expr \mid$
 $\quad Expr \% Expr \mid -Expr \mid Expr < Expr \mid Expr <= Expr \mid$
 $\quad Expr > Expr \mid Expr >= Expr \mid Expr == Expr \mid Expr != Expr \mid$
 $\quad Expr \&\& Expr \mid Expr \parallel Expr \mid !Expr \mid ReadInteger() \mid$
 $\quad readLine() \mid new \text{ ident} \mid NewArray(Expr, Type) \mid$
 $\quad itod(Expr) \mid dtoid(Expr) \mid itob(Expr) \mid btoid(Expr)$
 $LValue ::= ident \mid Expr . \text{ ident} \mid Expr [Expr]$
 $Call ::= \text{ ident } (Actuals) \mid Expr . \text{ ident } (Actuals)$
 $Actuals ::= Expr^+, \mid \epsilon$
 $Constant ::= intConstant \mid doubleConstant \mid boolConstant \mid$
 $\quad stringConstant \mid null$

(1)

۳ ساختار برنامه

یک برنامه‌ی Decaf از دنباله‌ای از تعاریف^۹ تشکیل شده است. هر تعریف نشان‌دهنده‌ی یک متغیر، تابع، کلاس و یا interface است. یک برنامه باید حتماً شامل تابع global ای به نام main باشد که ورودی‌ای ندارد و یک int برمی‌گرداند. این تابع به عنوان نقطه‌ی شروع برنامه استفاده می‌شود.

۴ Scope

این زبان از سطوح مختلف Scope پشتیبانی می‌کند. تعاریفی که در سطح بالایی باشند، به عنوان global در نظر گرفته می‌شوند. هر تعریف کلاس، class-scope مختص به خودش را دارد. هر تابع یک local-scope برای لیست پارامترهایش دارد و یک local-scope برای بدنه‌اش. دقت کنید که مشابه زبان C در این زبان نیز scope های داخلی بر روی scope های خارجی سایه می‌اندازند. برای مثال یک متغیر که در یک تابع تعریف شده، متغیر global ای که هم‌نامش است را mask می‌کند.

به طور کلی قوانین زیر در این زبان در خصوص scope ها وجود دارند:

۱. وقتی وارد یک scope می‌شویم، تمامی تعاریف داخل آن scope بلافاصله قابل دیدن هستند.^{۱۰}
۲. شناسه‌های درون یک scope باید منحصر به فرد باشند.
۳. شناسه‌ای که در scope داخلی‌تر است، شناسه‌ی بیرونی را mask می‌کند.
۴. تعاریفی که به صورت global انجام شده‌اند در تمام برنامه قابل دسترسی‌اند مگر آن که توسط تعریف دیگری mask شوند.
۵. تعاریفی که درون scope های بسته شده‌اند، دیگر قابل دسترسی نیستند.
۶. تمام شناسه‌ها باید جایی تعریف شده باشند.

۵ انواع داده

انواع داده که به صورت builtin در زبان موجوداند عبارت‌اند از:

int, double, bool, string, void

انواع int, double و bool به میزان ۴ بایت فضا اشغال می‌کنند.

همچنین امکان تعریف انواع دیگر (شیء‌ها^{۱۱} و کلاس‌ها و interface ها) نیز وجود دارد. آرایه‌ها از تمام انواع به جز void قابل ساختن‌اند. همچنین Decaf از آرایه‌ای از آرایه نیز پشتیبانی می‌کند.

^۹declarations

^{۱۰}مانند Java کامپایلر ما هم باید بیش از یک دور برنامه را بخواند. در دور اول اطلاعات را جمع‌آوری می‌کند و درخت پارس را می‌سازد. پس از آن به بررسی معنایی (Semantic) می‌پردازیم. مزیت این کار در این است که می‌توان طوری عمل کرد که تعاریف قبل از این که به صورت واژه‌ای واقعا تعریف شده باشند در آن Scope قابل دسترسی باشند. مثلاً توابع درون یک کلاس می‌توانند به متغیرهایی که بعداً درون کلاس گرفته شده‌اند دسترسی داشته باشند.

^{۱۱}objects

۶ متغیرها

متغیرها می‌توانند از انواع غیر void و آرایه‌ها و شیء‌ها و کلاس‌ها و interface ها باشند. متغیرهایی که بیرون کلاسی تعریف شده باشند global اند. آنهایی که بیرون توابع ولی داخل کلاس قرار دارند، دارای class-scope اند. آنهایی که درون تابع و یا لیست پارامترهایش قرار دارند نیز دارای local-scope هستند.

۷ آرایه‌ها

تمامی آرایه‌ها در Heap و بر اساس سایز مورد نیاز قرار دارند.

۱. آرایه‌ها می‌توانند از تمامی انواع غیر void و شیء‌ها و کلاس‌ها و interface ها و همچنین خود آرایه‌ها باشند.

۲. با دستور زیر می‌توان آرایه‌ای از type و سایز N ساخت:

`NewArray(N, type)`

دقت کنید که N باید بزرگتر از ۰ باشد. وگرنه خطای زمان اجرا رخ می‌دهد.

۳. تعداد عناصر یک آرایه در زمان تعریف مشخص می‌شود و پس از آن قابل تغییر نیست.

۴. آرایه‌ها از تابع خاص `arr.length()` پشتیبانی می‌کنند که تعداد عناصر آرایه را بر می‌گرداند.

۵. اندیس‌گذاری آرایه‌ای فقط بر روی انواع آرایه قابل استفاده است.

۶. اندیس‌ها از ۰ شروع می‌شوند.

۷. اندیسی که برای آرایه‌ها استفاده می‌شود باید از جنس `int` باشد.

۸. خطای دسترسی به اندیس نامعتبر در زمان اجرا داده می‌شود.

۹. آرایه‌ها ممکن است به عنوان پارامتر به توابع پاس داده شوند یا توسط آنها `return` شوند. همچنین با این کار تغییرات مشابه `call-by-reference` اعمال می‌شوند.

۱۰. با انتساب یک آرایه به آرایه‌ی دیگر، صرفاً `reference` یکی در دیگری کپی می‌شود.

۱۱. مقایسه‌ی بین آرایه‌ها تنها بر روی `reference` ها انجام می‌شود.

۱۲. می‌توان دو آرایه را با عملیات جمع به یکدیگر الحاق^{۱۲} کرد. مثلاً:

$$[1, 4] + [4, 5] = [1, 4, 4, 5]$$

^{۱۲} Concatenation

۸ رشته‌ها

برنامه قادر است شامل ثوابت رشته‌ای باشد، رشته‌ای را از کاربر با تابع ReadLine ورودی بگیرد، رشته‌ها را مقایسه کند و چاپشان کند. نوع رشته به انواع دیگر قابل تبدیل نیست (و برعکس). رشته‌ها نیز به صورت reference پیاده‌سازی می‌شوند.

۱. تابع ReadLine یک خط از کاراکترها را از کاربر ورودی می‌گیرد.
۲. مانند انتساب آرایه‌ها، اینجا نیز اگر یک رشته به رشته‌ی دیگری انتساب یابد، تنها reference ها کپی می‌شوند.
۳. رشته‌ها ممکن است به توابع پاس داده شوند یا توسط آنها return شوند.
۴. رشته‌ها با == و != قابل مقایسه‌اند. این مقایسه به صورت case-sensitive است.
۵. می‌توان دو رشته را با عملیات جمع به یکدیگر الحاق کرد. مثلاً:
"he" + "llo" = "hello"

۹ توابع

تعریف یک تابع شامل نام آن تابع به همراه امضایش^{۱۳} است. امضا به معنی نوع خروجی در کنار تعداد و نوع پارامترهای یک تابع است.

۱. توابع یا global اند و یا درون یک کلاس تعریف شده‌اند. توابع نمی‌توانند به داخل یک تابع دیگر تعریف شوند (توابع تو در تو نداریم).
۲. توابع می‌توانند صفر یا بیشتر پارامتر داشته باشند.
۳. پارامترهای توابع می‌توانند از همه‌ی انواع غیر void و شیء‌ها و کلاس‌ها و interface ها و آرایه‌ها باشند. خروجی‌اش می‌تواند void نیز باشد.
۴. شناسه‌هایی که برای پارامترهای رسمی^{۱۴} استفاده می‌شوند باید منحصر به فرد باشند.
۵. پارامترها در یک scope مجزا از متغیرهای داخل بدنه‌ی تابع قرار دارند. پس متغیرهای داخل یک تابع می‌توانند پارامترهایش را mask^{۱۵} کنند.
۶. اضافه‌بار توابع^{۱۶} در این زبان مجاز نیست.
۷. اگر نوع خروجی تابع غیر void باشد، هر جا که return می‌کنیم، نوع داده شده باید با نوع خروجی هم‌خوان باشد.
۸. اگر نوع خروجی یک تابع void باشد، تنها می‌تواند از return خالی استفاده کند.

^{۱۳} type signature

^{۱۴} formal-parameters

^{۱۵} وقتی یک متغیر جدید، یک متغیر قبلی را mask می‌کند، به این معنی است که در آن scope دیگر به متغیر قبلی دسترسی نداریم و با ارجاع به آن، به متغیر جدید دسترسی پیدا می‌کنیم.

^{۱۶} function-overloading

۹. فراخوانی بازگشتی مجاز است.

۱۰. تابع abstract نداریم.

۱۰ فراخوانی توابع

فراخوانی توابع شامل پاس دادن آرگومان‌ها از فراخوان^{۱۷} به تابع، سپس اجرای بدنه‌ی تابع و در نهایت بازگشت به فراخوان (و در صورت نیاز برگرداندن مقدار خروجی) است. وقتی یک تابع فراخوانی می‌شود، آرگومان‌های واقعی^{۱۸} ارزیابی می‌شوند و در جای پارامتر رسمی قرار می‌گیرند. تمام پارامترها و مقادیر خروجی در Decaf به صورت call by value هستند. البته دقت کنید که در مورد شیء‌ها، مقادیر همان reference ها هستند.

۱. تعداد آرگومان‌هایی که به یک تابع پاس داده می‌شوند باید با تعداد پارامترهای رسمی آن تابع برابر باشد.

۲. نوع پارامترهایی که به تابع پاس داده می‌شوند باید با نوع پارامترهای رسمی متناظر یکسان باشد.

۳. آرگومان‌های واقعی یک تابع از چپ به راست ارزیابی می‌شوند.

۴. بازگشت به فراخوان در هنگام رسیدن به یک return و یا رسیدن به انتهای تابع صورت می‌گیرد.

۵. خود فراخوانی تابع به صورت یک مقدار از نوع خروجی تابع ارزیابی می‌شود.

۱۱ کلاس‌ها

تعریف یک کلاس جدید باعث ایجاد یک نوع جدید و همچنین scope مربوط به آن کلاس می‌شود. تعریف کلاس شامل لیستی از فیلدها می‌شود که هر فیلد می‌تواند متغیر یا تابع باشد. همچنین متغیرهای یک کلاس با نام‌های instance variable و member data نیز شناخته می‌شوند. و گاهی به توابع کلاس‌ها نیز method یا member functions گفته می‌شود.

متغیرها و توابع یک کلاس می‌توانند خصوصی^{۱۹}، محافظت‌شده^{۲۰} یا عمومی^{۲۱} باشند. فیلدهای خصوصی تنها در همان کلاس، فیلدهای محافظت‌شده درون کلاس و زیرکلاس‌هایش و نهایتاً فیلدهای عمومی در تمامی کلاس‌ها و global-scope قابل دسترسی هستند. همچنین در صورت ذکر نشدن نوع دسترسی، فیلد تعریف شده عمومی تلقی می‌شود.

۱. تعریف تمام کلاس‌ها به صورت global انجام می‌شود.

۲. نام کلاس‌ها منحصر به فرد است.

۳. هر نام در میان فیلدهای داخل یک کلاس حداکثر یک بار استفاده می‌شود (نمی‌شود یک متغیر با یک تابع داخل یک کلاس هم‌نام باشند).

^{۱۷} caller

^{۱۸} actual-parameter

^{۱۹} private

^{۲۰} protected

^{۲۱} public

۴. فیلدها به هر ترتیبی ممکن است تعریف شوند.
۵. متغیرها می‌توانند از تمامی انواع غیر void و کلاس‌ها و شیء‌ها و ... باشند.
۶. استفاده از this در هنگام دسترسی به فیلدها درون توابع اختیاری است.

۱۲ شیء‌ها

- یک متغیر از انواع جدید تعریف شده (کلاس‌ها و interface ها) به نام یک شیء و یا نمونه ^{۲۲} شناخته می‌شود. شیء‌ها به صورت reference پیاده‌سازی می‌شوند. همه‌ی آنها به صورت پویا درون heap گرفته می‌شوند و با عملگر new تعریف می‌شوند.
۱. نوعی که در هنگام استفاده از new استفاده می‌شود باید نام یک کلاس باشد (نام interface ها اینجا مجاز نیست). عملگر new نمی‌تواند برای ساخت متغیر از انواع پایه‌ی زبان استفاده شود.
 ۲. عملگر . برای دسترسی به فیلدهای شیء استفاده می‌شود.
 ۳. در هنگام فراخوانی توابعی به صورت expr.method() عبارت (expr) داده شده باید یک کلاس یا interface باشد. و method باید یکی از توابع داخل آن باشد.
 ۴. در هنگام دسترسی به متغیرها به صورت expr.var خود expr باید یک کلاس مانند T باشد و var باید متغیر داخلی آن باشد. ضمناً قواعد مربوط به دسترسی هم باید لحاظ شوند.
 ۵. در هنگام انتساب یک شیء به شیء دیگر تنها reference منتسب می‌شود.
 ۶. شیء‌ها می‌توانند به توابع پاس داده شوند یا از آنها return شوند. و از آنجایی که reference منتقل می‌شوند، تغییرات در تابع فراخوان قابل دیدن هستند.

۱۳ ارث‌بری

در این زبان ارث‌بری یگانه ^{۲۳} پشتیبانی می‌شود که اجازه می‌دهد یک کلاس، کلاس دیگری را extend کند و به آن تعدادی فیلد اضافه کند (می‌تواند برخی توابع کلاس پدر را override کند). معنای extends A این است که A تمام فیلدهایی که در B قرار دارند را دارا است و تعدادی فیلد نیز خودش تعریف کرده. یک زیرکلاس می‌تواند توابع کلاس پدر را override کند ولی باید امضای تابع جدید با تابع قبلی یکسان باشد. همچنین این زبان از upcast خودکار پشتیبانی می‌کند.

تمام method ها در Decaf به صورت پویا هستند و کامپایلر لزوماً نمی‌تواند بفهمد که آدرس دقیق یک تابع که در قرار است در زمان کامپایل صدا زده شود چیست. برای واضح‌تر شدن این موضوع فرض کنید که قرار است تابعی از یک شیء upcast شده فراخوانی شود (و آن تابع در کلاس فرزند نیز پیاده‌سازی شده). پس انتخاب تابع در زمان اجرا انجام می‌شود. برای این کار برای هر شیء یک جدول توابع نگه می‌داریم.

^{۲۲} instance
^{۲۳} single-ingeritance

۱. نمی‌توان از انواع پایه، آرایه‌ها و interface ها ارث‌بری کرد.
۲. تمام فیلدهای کلاس پدر به فرزند به ارث می‌رسند.
۳. زیرکلاس‌ها نمی‌توانند متغیرها را override کنند.
۴. زیرکلاس می‌تواند توابع پدر را override کند ولی باید این توابع امضای یکسانی داشته باشند.
۵. امکان overload وجود ندارد. دو تابع با نام یکسان و امضای متفاوت درون یک کلاس نداریم.
۶. فرض کنید کلاس Cow کلاس Animal را extend کند. در این صورت می‌توان عبارتی از نوع Cow را به شی‌ای از نوع Animal نسبت داد. یا مثلاً اگر یک تابع ورودی‌ای از نوع Animal دارد، می‌توان در عوض به آن شی‌ای از نوع Cow پاس داد. ولی برعکس این کار ممکن نیست.
۷. قانون بالا در سطوح دیگر ارث‌بری نیز صادق است.
۸. نوع یک شیء در زمان کامپایل مشخص می‌کند که چه فیلدهایی قابل دسترسی‌اند. مثلاً وقتی که Cow را به Animal که کلاس پدرش است upcast می‌کنیم، دیگر نمی‌توان به فیلدهای مختص Cow دسترسی داشت.
۹. امکان upcast کردن برای آرایه‌ها وجود ندارد. مثلاً آرایه‌ای از Cow با آرایه‌ای از Animal ناسازگار است.

۱۴ interface ها

یک کلاس می‌تواند یک یا چند interface را نیز پیاده‌سازی^{۲۴} کند. تعریف یک interface شامل پروتوتایپ تعدادی تابع (بدون پیاده‌سازی بدنه‌ی آنها) است. وقتی یک کلاس یک interface را پیاده‌سازی می‌کند، موظف است که تمام توابع آن interface را داشته باشد (برخلاف Java در اینجا کلاس abstract نداریم). همچنین upcast خودکار به interface های پیاده‌سازی شده ممکن است.

۱. اگر کلاسی یک interface را پیاده‌سازی کند موظف است تمام توابعش را داشته باشد.
۲. فرض کنید کلاس Shape به گونه‌ای تعریف شده باشد که Colorable را که یک interface است، پیاده‌سازی کند. حال نمونه‌ای از Shape را می‌توان از سمت راست به متغیری از نوع Colorable منتسب کرد. همچنین می‌توان در هنگام پاس دادن آرگومان به توابع از این upcast استفاده کرد.
۳. یک زیرکلاس تمام interface های پدرش را به ارث می‌برد.
۴. در زمان کامپایل مشخص می‌شود که به چه فیلدهایی دسترسی داریم. مثلاً بعد از این که Shape را به Colorable که توسط Shape پیاده‌سازی شده است upcast کردیم، دیگر به فیلدهای مختص Shape دسترسی نداریم.
۵. مجدداً در اینجا نیز امکان upcat کردن برای آرایه‌ها وجود ندارد.

^{۲۴}implement

۱۵ همخوانی و یکسان بودن انواع

انواع در Decaf قویاً رعایت می‌شوند. اگر نوع A با نوع B یکسان باشد، هر عبارتی از نوع A می‌تواند آزادانه به یک متغیر از نوع B منتسب شود و برعکس. دو نوع آرایه با هم یکسان‌اند اگر و تنها اگر نوع درایه‌هایشان یکسان باشد (که خود درایه‌ها ممکن است آرایه باشند). انواع دیگر در صورتی یکسان‌اند که نام یکسانی داشته باشند.

همخوانی انواع به صورت محدودتر و بدون جهت است. اگر نوع A با نوع B همخوان باشد، آنگاه عبارتی از نوع A هر جا که نوع B مورد انتظار باشد قابل استفاده است. ولی در مورد عکس این قضیه چیزی نمی‌توان گفت. اگر هم A با B همخوان باشد و هم B با A آنگاه A و B طبق تعریف بالا یکسان‌اند. یک زیرکلاس با پدرش همخوان است. ولی عکس این موضوع درست نیست. یک کلاس با interface هایی که پیاده‌سازی‌شان کرده همخوان است. ثابت null با هر کلاس یا interface ای همخوان است. عملیات‌هایی مانند انتساب و پاس‌دادن پارامتر برای انواع همخوان قابل انجام‌اند.

۱۶ انتساب

برای انواع پایه، مقدار کپی می‌شود. یعنی با انتساب زیر:

$LValue = Expr$

مقدار حاصل از ارزیابی Expr درون LValue ریخته می‌شود. برای آرایه‌ها، شیء‌ها و رشته‌ها -ref- erence کپی می‌شود. یعنی در نهایت LValue دارای reference همان شیء‌ای می‌شود که از ارزیابی Expr بدست می‌آید.

۱. مقدار سمت راست باید با نوع متغیر سمت چپ همخوان باشد.
۲. ثابت null فقط به کلاس‌ها و interface ها قابل انتساب است.
۳. درون یک تابع می‌توان به پارامترهای رسمی‌اش مقدار منتسب کرد. ولی این تغییرات تنها در scope تابع هستند.

۱۷ ساختار کنترلی

اینها در Decaf مشابه C/Java اند و به همان صورت رفتار می‌کنند.

۱. یک بند else همیشه به نزدیک‌ترین if قبل خودش که تمام نشده متصل می‌شود.
۲. یک break یا continue فقط داخل حلقه‌های while و for قابل استفاده است.
۳. مقدار خروجی در عبارت return باید با نوع خروجی تابع همخوان باشد.

۱۸ عبارات

برای ساده‌سازی امکان co-mingling و تبدیل نوع وجود ندارد. مثلاً نمی‌توان یک integer را با یک double جمع کرد.

۱. عملوندهای عملیات محاسباتی (+, -, *, /, %) یا هر دو int اند و یا هر دو double هستند. همچنین عملوند منفی یگانی^{۲۵} . نوع حاصل نیز از همان نوع است. البته عملیات محاسباتی جمع، در مورد انواع رشته و آرایه نیز همانطور که قبلا اشاره شد، تعریف شده است.
۲. عملوندهای عملیات مقایسه‌ای (<, >, <=, >=) یا هر دو int اند و یا هر دو double هستند. نوع حاصل نیز bool خواهد بود.
۳. عملوندهای عملیات‌های =, !=, باید از نوع یکسان باشند. نوع حاصل نیز bool است. دقت کنید این دو عملگر روی رشته‌ها نیز تعریف شده‌اند. همچنین می‌توانند دو شیء باشند و یا یک شیء و null که ثابت است. نوع این شیء‌ها باید حداقل در یک طرف همخوان باشد. نوع حاصل مجدداً bool است.
۴. عملوندهای عملیات منطقی !, ||, && باید از نوع bool باشند و نوع حاصل نیز bool خواهد بود.
۵. برای تمام عبارات، عملوندها از چپ به راست ارزیابی می‌شوند.
۶. این زبان از اتصال کوتاه بولین^{۲۶} استفاده نمی‌کند. یعنی اگر در یک دنباله از AND ها مشخص شد عبارت اول مقدار false است، محاسبه را متوقف نکرده و تا انتهای عبارت را محاسبه می‌کند.
۷. عملگر % تنها در مورد اعداد صحیح تعریف شده است. عملوندهای این عملیات می‌توانند منفی نیز باشند.

اولویت عملیات‌ها از زیاد به کم:

نماد	توضیح
[.	اندیس‌گذاری آرایه و انتخاب فیلد
! -	منفی و not یگانی
* / %	ضرب و تقسیم و باقیمانده
+ -	جمع و تفریق
< <= > >=	مقایسه
== !=	برابری
&&	and منطقی
	or منطقی
=	انتساب

برای override کردن اولویت‌ها از پرانتز استفاده می‌شود.

۱۹ توابع کتابخانه‌ای

تعداد محدودی از این توابع در زبان وجود دارند که برای I/O و گرفتن حافظه استفاده می‌شوند. این توابع از این قرار اند:

^{۲۵} unary-minus
^{۲۶} boolean-short-circuit

Print, ReadInteger, ReadLine, NewArray, btoi, itob, dtoi, itod

۱. آرگومانی که به Print پاس داده می‌شود تنها می‌تواند از نوع string و int و یا bool باشد. دقت کنید تابع Print با دقتاً ۸ رقم اعشاری خروجی چاپ می‌کند.
۲. در تابع Print در پایان هر خط، کاراکتر new-line چاپ می‌شود.
۳. تابع Print می‌تواند چند آرگومان داشته باشد و آن‌ها را بدون فاصله از هم چاپ می‌کند، مثلاً:
 $\text{Print}(2, 3) \rightarrow 23$
۴. در تابع Print هنگام چاپ کردن متغیرهای از نوع بولین آن‌ها را به صورت true یا false نمایش می‌دهیم.
۵. اولین آرگومان NewArray باید integer باشد. دومی از هر نوع غیر void می‌تواند باشد.
۶. خروجی NewArray یک آرایه از نوع خواسته شده است.
۷. تابع ReadLine یک خط از کاراکترها را از کاربر می‌گیرد.
۸. تابع ReadInteger یک خط از کاربر می‌گیرد و آن‌را به integer تبدیل می‌کند (اگر کاربر یک عدد قابل قبول وارد نکرده بود * برمی‌گرداند).
۹. تابع ReadInteger تنها اعداد صحیح در مبنای ۱۰ را به عنوان ورودی می‌پذیرد.
۱۰. تابع itod برای تبدیل عدد صحیح به عدد حقیقی^{۲۷}، مثلاً:
 $\text{itod}(4) \rightarrow 4.0$
۱۱. تابع dtoi برای تبدیل عدد حقیقی به نزدیک‌ترین عدد صحیح. مثلاً:
 $\text{dto}(4.33) \rightarrow 4, \text{dto}(0.5) \rightarrow 1, \text{dto}(-0.1) \rightarrow 0$
 $\text{dto}(-0.5) \rightarrow 0, \text{dto}(-0.7) \rightarrow -1$
۱۲. تابع itob برای تبدیل اعداد غیر صفر به true و صفر به false.
۱۳. تابع btoi برای تبدیل false به صفر و true به یک.

۲۰ بررسی‌های زمان اجرا

در زمان اجرا تنها دو چیز بررسی می‌شوند:

۱. اندیس آرایه‌ها باید در محدوده‌ی مجاز باشد که برابر است با:
 $[0, \text{arr.length}() - 1]$
 ۲. سایزی که به NewArray داده می‌شود باید مثبت باشد.
 ۳. اگر از فیلد شی‌ای استفاده شود که این شی هنوز ساخته نشده است.
 ۴. در صورتی که تقسیم بر صفر صورت بگیرد.
- در زمان رخ دادن این مشکلات، پیغام مناسبی در terminal چاپ می‌شود و برنامه متوقف می‌شود.

^{۲۷}توجه کنید که این اعداد ۴ بایتی هستند

۲۱ کارهایی که Decaf انجام نمی‌دهد!

کارهای زیادی هستند که Java و یا C++ انجام می‌دهند ولی Decaf انجام نمی‌دهد. تعدادی از آنها را در اینجا می‌آوریم:

۱. این زبان حکمی در مورد مقادیر اولیه‌ی متغیرهایی که مقداردهی نشده‌اند نمی‌دهد.
۲. این زبان چک نمی‌کند که متغیری که استفاده می‌شود حتماً مقداردهی اولیه شده باشد.
۳. این زبان تشخیص نمی‌دهد که یک تابع که قرار بوده مقداری برگرداند، پس از تمام شدن بدنه‌اش مقداری return کرده یا نه.
۴. این زبان کدهای غیرقابل دسترسی را تشخیص نمی‌دهد.
۵. این زبان برای اشیاء از constructor و destructor پشتیبانی نمی‌کند.