پروژه‌ی درس طراحی کامپایلر

توضیحات زبان Decaf

آقای بهرامی

نسخه ۱

۱۳ اردیبهشت ۱۳۹۹

شما در پروژه‌ی این درس قرار است کامپایلری برای یک زبان برنامه‌نویسی شیءگرای ساده به اسم Decaf بنویسید. این زبان از ارث‌بری [۱] و encapsulation پشتیبانی می‌کند. 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 دقیقا منطبق بر هیچ‌یک از زبان‌های گفته شده نیست. امکانات این زبان برای آن که پیاده‌سازی کامپایلرش به یک کابوس 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 مرتبا به آن نیاز پیدا خواهید کرد.

۱ بررسی واژه‌ای زبان

در لیست زیر می‌توانید کلیدواژه [۲] زبان را مشاهده کنید. آنها رزرو شده‌اند. که به این معنی است که توسط برنامه‌نویس به عنوان شناسه [۳] قابل استفاده نیستند.

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

یک شناسه، دنباله‌ای از حروف، اعداد و زیرخط [۴] است که با یک حرف شروع می‌شود. این زبان حساس به بزرگی و کوچکی حروف است (Case-sensitive) . همچنین شناسه‌ها حداکثر می‌توانند ۳۱ کاراکتر باشند.

فاصله‌ی سفید [۵] به عنوان جداکننده‌ی نشانه‌ها [۶] 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(۲۳this 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 [۷] 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 ۰x و یا ۰X 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 lowecase نوشته شوند. برای مثال نشانه‌های زیر به عنوان ثوابت صحیح قابل قبول هستند:

$$8, 012, 0x0, 0X12aE$$

یک ثابت اعشاری [۸] دنباله‌ای از ارقام، یک نقطه به عنوان ممیز و مجددا 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:

---

[۱] Inheritance
[۲] Keyword
[۳] identifier
[۴] underscore
[۵] Whitespace
[۶] tokens
[۷] integer
[۸] double

$0.12, 12.$

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

$12.2E + 2$

برای اعداد اعشاری که به صورت نمایش علمی نشان داده می‌شوند، حتما نیاز به ممیز داریم. علامت بخش توان اختیاری است (اگر گذاشته نشود، مثبت در نظر گرفته می‌شود). همچنین E می‌تواند lowecase 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.

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 [۹] 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:

---

[۹] punctuations

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. مثلا 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.

$$
\begin{aligned}
Program &::= Decl^+ \\
Decl &::= VariableDecl \mid FunctionDecl \mid ClassDecl \mid InterfaceDecl \\
VariableDecl &::= Variable; \\
Variable &::= Type \text{ ident} \\
Type &::= \text{int} \mid \text{double} \mid \text{bool} \mid \text{string} \mid \text{ident} \mid Type\,[\,] \\
FunctionDecl &::= Type \text{ ident } (Formals)\ StmtBlock \mid \\
&\qquad \text{void ident } (Formals)\ StmtBlock \\
Formals &::= Variable^+, \mid \epsilon \\
ClassDecl &::= \text{class ident } < \text{extends ident} > < \text{implements ident}^+, > \ \{Field^*\} \\
Field &::= VariableDecl \mid FunctionDecl \\
InterfaceDecl &::= \text{interface ident } \{Prototype^*\} \\
Prototype &::= Type \text{ ident}(Formals); \mid \text{void ident}(Formals); \\
StmtBlock &::= \{VariableDecl^*\ Stmt^*\} \\
Stmt &::= < Expr >; \mid IfStmt \mid WhileStmt \mid ForStmt \mid \\
&\qquad BreakStmt \mid ReturnStmt \mid PrintStmt \mid StmtBlock \\
IfStmt &::= \text{if } (Expr)\ Stmt\ < \text{else } Stmt > \\
WhileStmt &::= \text{while } (Expr)\ Stmt \\
ForStmt &::= \text{for } (< Expr >; Expr; < Expr >)\ Stmt \\
ReturnStmt &::= \text{return } < Expr >; \\
BreakStmt &::= \text{break;} \\
PrintStmt &::= \text{print } (Expr^+,); \\
Expr &::= LValue = Expr \mid Constant \mid LValue \mid \text{this} \mid Call \mid (Expr) \mid \\
&\qquad Expr + Expr \mid Expr - Expr \mid Expr * Expr \mid Expr \,/\, Expr \mid \\
&\qquad Expr \,\%\, Expr \mid -Expr \mid Expr < Expr \mid Expr <= Expr \mid \\
&\qquad Expr > Expr \mid Expr >= Expr \mid Expr == Expr \mid Expr != Expr \mid \\
&\qquad Expr \,\&\&\, Expr \mid Expr \,||\, Expr \mid !Expr \mid \text{ReadInteger()} \mid \\
&\qquad \text{readLine()} \mid \text{new ident} \mid \text{NewArray}(Expr, Type) \\
LValue &::= \text{ident} \mid Expr\ .\ \text{ident} \mid Expr[Expr] \\
Call &::= \text{ident } (Actuals) \mid Expr\ .\ \text{ident } (Actuals) \\
Actuals &::= Expr^+, \mid \epsilon \\
Constant &::= \text{intConstant} \mid \text{doubleConstant} \mid \text{boolConstant} \mid \\
&\qquad \text{stringConstant} \mid \text{null}
\end{aligned}
\tag{1}
$$

۳ ساختار برنامه

یک برنامه‌ی Decaf از دنباله‌ای از تعاریف [۱۰] تشکیل شده. که هر تعریف نشان‌دهنده‌ی یک متغیر، تابع، کلاس و یا interface است. یک برنامه باید حتما 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. [۱۱]

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

همچنین امکان تعریف انواع دیگر (شیءها [۱۲] 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.

---

[۱۰] declarations

[۱۱] 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 دسترس باشند. مثلا توابع درون یک کلاس می‌توانند به متغیرهایی که بعدا درون کلاس گرفته شده‌اند دسترسی داشته باشند.

[۱۲] 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. آریه‌ها 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، صرفا reference یکی در دیگری کپی می‌شود.

۱۱. مقایسه‌ی بین آرایه‌ها تنها برروی reference ها انجام می‌شود.

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.

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 می‌توانند 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 یکسان باشد.

۸. اگر نوع خروجی یک تابع void باشد، تنها می‌تواند از return خالی استفاده کند.

۹. فراخوانی بازگشتی مجاز است.

۱۰. تابع abstract نداریم.

۱۰ فراخوانی توابع

فراخوانی توابع شامل پاس‌دادن 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

[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
[۱۷] caller
[۱۸] actual-parameter

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.

زبان Decaf کپسوله‌سازی [۱۹] را در شیءها با مکانیزم ساده‌ای اعمال می‌کند: تمام متغیرهای یک کلاس، محافظت‌شده‌اند و تنها توسط کلاس و زیرکلاس‌هایش قابل دسترسی‌اند و تمام تابع‌ها عمومی (public) می‌باشند و در global-scope قابل دسترسی‌اند. پس به state یک شیء تنها از طریق توابعش می‌توان دسترسی داشت.

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).

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 [۲۰] شناخته می‌شود. شیءها به صورت reference پیاده‌سازی می‌شوند. همه‌ی آنها به صورت پویا درون heap گرفته می‌شوند و با عملگر new تعریف می‌شوند.

---

[۱۹] encapsulation

[۲۰] instance

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. همچنین این دسترسی تنها درون خود T و یا زیرکلاس‌هایش قابل انجام است.

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 [۲۱] 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 A extends B 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 که به این معنی است که شیء از کلاس مشتق‌شده قابل ارائه است.

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.

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 کند.

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 شیءای از نوع Animal نسبت داد. و یا مثلا اگه یک تابع ورودی‌ای از نوع Animal دارد، می‌توان در عوض به آن شیءای از نوع Cow پاس داد. ولی برعکس این کار ممکن نیست.

[۲۱] single-ingeritance

۷. قانون بالا در سطوح دیگر ارث‌بری نیز صادق است.

۸. نوع یک شیء در زمان کامپایل مشخص می‌کند که چه فیلدهایی قابل دسترسی‌اند. مثلا 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. مثلا آرایه‌ای از Cow با آرایه‌ای از Animal ناسازگار است.

۱۴ interface ها

یک کلاس می‌تواند یک یا چند interface را نیز پیاده‌سازی [۲۲] 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 پاس‌دادن 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. مثلا بعد از این که Shape را به Colorable که توسط Shape پیاده‌سازی شده‌است upcast کردیم، دیگر به فیلدهای مختص Shape دسترسی نداریم.

۵. مجدداً در اینجا نیز امکان upcat کردن برای آرایه‌ها وجود ندارد.

۱۵ همخوانی و یکسان‌بودن انواع

انواع در Decaf قویاً رعایت می‌شوند. و یک متغیر از یک نوع، فقط مقادیر آن نوع را می‌تواند نگه دارد. 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 پایه با هم یکسان‌اند اگر و تنها اگر نوعشان دقیقا یکسان باشد. 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 عباراتی از نوع A هرجا 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.

[۲۲] implement

یک زیرکلاس با پردش 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 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. مثلا 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 [۲۳] . نوع حاصل نیز از همان نوع 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 (یک استثنا برای رشته‌ها وجود دارد). نوع حاصل نیز bool است.

همچنین می توانند دو شیء باشند و یا یک شیء و null که ثابت است. نوع این شیءها باید حداقل در یک طرف همخوان باشد. نوع حاصل مجددا bool است.

[۲۳] unary-minus

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 مدار کوتاه بولین [۲۴] 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.

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:

Print, ReadInteger, ReadLine, NewArray

۱. آرگومانی که به Print پاس داده می‌شود تنها می‌تواند از نوع string و int و double و یا bool باشد. دقت کنید که در هنگام چاپ داده از نوع double عدد را تا ۴ رقم اعشار گرد کنید.

۲. اولین آرگومان NewArray باید integer باشد. دومی از هر نوع غیر void می‌تواند باشد.

۳. خروجی NewArray یک آرایه از نوع خواسته‌شده است.

۴. تابع ReadLine یک خط از کاراکترها را از کاربر می‌گیرد.

۵. تابع ReadInteger یک خط از کاربر می‌گیرد و آن‌را به integer تبدیل می‌کند (اگر کاربر یک عدد قابل قبول وارد نکرده بود ۰ برمی‌گرداند).

[۲۴] boolean-short-circuit

۲۰ بررسی‌های زمان اجرا

در زمان اجرا تنها دو چیز بررسی می‌شوند:

۱. اندیس آرایه‌ها باید در محدوده‌ی مجاز باشد که برابر است با:

$$[۰, arr.length() - ۱]$$

۲. سایزی که به NewArray داده می‌شود باید مثبت باشد.

در زمان رخ‌دادن این مشکلات، پیغام مناسبی در terminal چاپ می‌شود و برنامه متوقف می‌شود.

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 بررسی نمی‌کند که شیء‌ای که در حال استفاده است حتماً قبلاً در حافظه قرار گرفته باشد (یعنی allocate شده باشد).

۵. این زبان کدهای غیرقابل دسترسی را تشخیص نمی‌دهد.

۶. این زبان مکانیزمی برای deallocation و یا garbage-collection ندارد.

۷. این زبان برای اشیا از constructor و destructor پشتیبانی نمی‌کند.