

/technologie

Java; what's next?

>>de wondere wereld van programmeertalen

Een programmeertaal is het gereedschap van de programmeur. In programmeertalen is een eeuwige evolutie gaande. Het dient continu te verbeteren zodat een programma steeds sneller en efficiënter geschreven kan worden en fouten vooraf gedetecteerd worden. Maar is elke nieuwe taal ook een verbetering? En wanneer stap je over op een nieuwe taal?

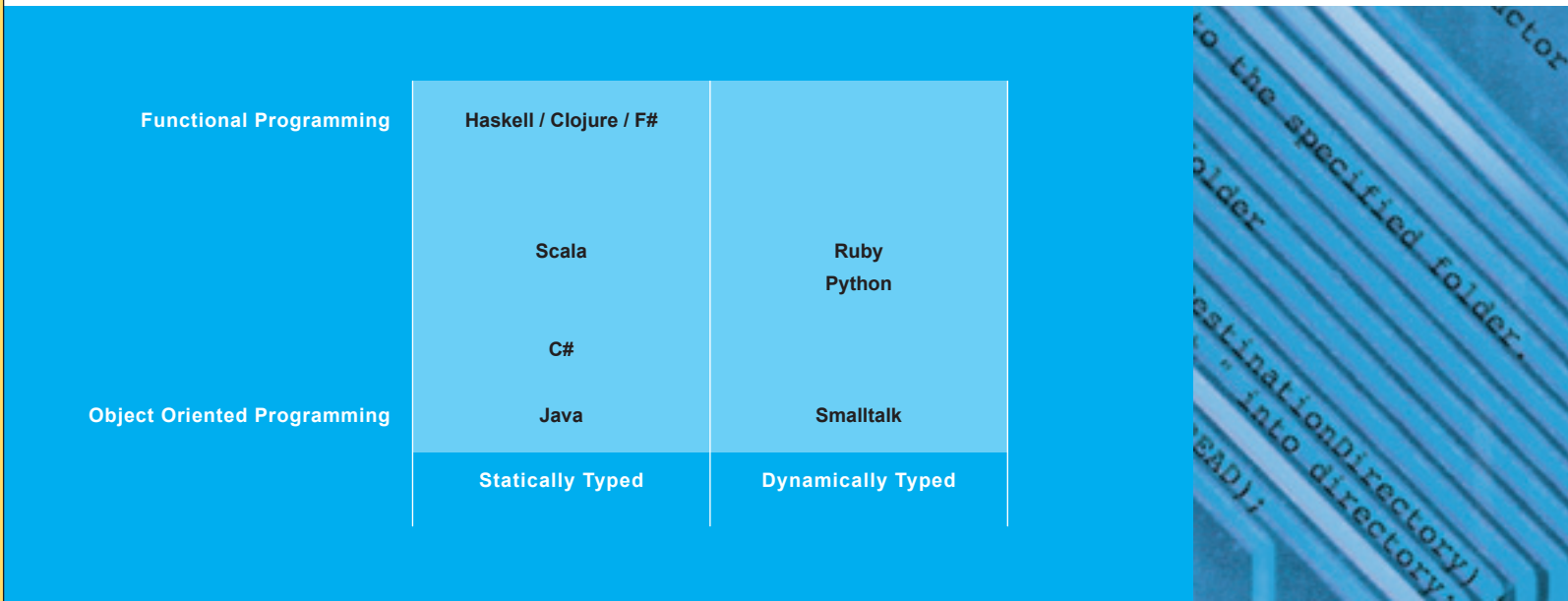
Er is een ijzeren wet in computerland: ieder jaar worden processors aanzienlijk sneller. Snellere computers maken het mogelijk dat software steeds meer voor ons kan doen. Programmeertalen vormen het gereedschap om software te bouwen. En ook daar vindt continu evolutie plaats. Er komen regelmatig nieuwe talen bij, waardoor de programmeur efficiënter kan werken. Nieuwe talen komen niet 'uit het niets'; ze bevatten vaak nieuwe elementen, maar ook altijd beproefde elementen uit het verleden.

Als talen een beetje op elkaar lijken, is het leerproces eenvoudiger. Maar als ze heel veel op elkaar lijken, maak je geen vooruitgang. Voor de gebruiker van deze talen noopt dat tot strategische keuzes: blijven we de huidige taal gebruiken of stappen we over op een nieuwe taal? De nieuwe taal heeft de belofte van efficiënter werken, maar vergt wel de inspanning om de nieuwe taal te leren en de hoop dat de rest van de wereld ook volgt.

Evolutie

Verandering in programmeertalen is van alle tijden. De eerste talen waren lineair. In de eerste 'Basic-dialecten' programmeerde je regel voor regel, met loops en sprongen naar een ander stuk van je code. Vervolgens ontstonden vriendelijkere talen zoals Pascal en C. Dit soort talen noemt men imperatieve of procedurele talen. Binnen de academische wereld is er ook een andere stroming: de functionele talen zoals Lisp en Haskell. In dit soort talen zijn functies het centrale concept. In de industrie zijn echter de functionele talen nooit echt populair geweest.

Een grote innovatie was de stap van imperatieve talen naar object-georiënteerde (OO) talen. De taal Smalltalk is de moeder van de OO-talen, maar met C++ werd deze stroming pas gemeengoed. Java is een OO-taal die nu veel gebruikt wordt, maar eigenlijk



```

switch(Label)
{
    case eNodeLabel.Stage:
        g.FillRectangle(BrushGray, new Rectangle(RectPercent.Width * Bau.
        string strperc = " " + 100.0f;
        g.DrawString(strperc, MainFont, BrushBlack, RectPercent.Width * Bau.
        break;
    case eNodeLabel.Amount:
        g.FillRectangle(BrushGray, RectPercent);
        string stram = Bau.Kombiniert.ToString();

        foreach(int hyb in Bau.HybridAnzahl)
            stram += (" "+hyb.ToString());

        g.DrawString(stram, MainFont, BrushBlack, RectPercent.Width * Bau.
        break;
    case eNodeLabel.None:
    default:
        g.FillRectangle(BrushGray, RectPercent);
        break;
}

```

moeten we het hebben over het Java-platform. Dit is een drie-eenheid van de taal Java, de Java Virtuele Machine en de Java-bibliotheken.

De Java-taal

Java stamt uit de jaren negentig en toen was hardware performance een issue. Om toch het onderste uit de kan te halen qua performance en tegelijk de overstap naar die (destijds nieuwe) taal niet te groot te maken, zijn soms concessies gedaan aan de taal. Java is daarom geen zuivere objectgeoriënteerde taal zoals Smalltalk: niet alles is een object. Voor een programmeertaal betekent dit tweeslachtigheid en omgaan met uitzonderingen.

De Java Virtuele Machine (JVM)

Het toepassen van een virtuele machine is één van de belangrijkste innovaties van het Java-platform. Bij het compileren van C wordt programmacode gegenereerd in machinetaal voor een specifieke processor. Uitvoering door een andere processor vereist dat het C-programma opnieuw gecompileerd wordt voor die andere processor. Bij compileren van Java wordt het programma in bytecode gegenereerd. Bytecode wordt niet direct door een processor uitgevoerd maar door een softwarelaag: de JVM. De JVM is typisch in C geschreven en wel voor een specifieke processor gecompileerd. Dit betekent dat, mits een JVM beschikbaar is, een Java-programma overal kan worden uitgevoerd (write once, run anywhere). Daarnaast doet de JVM zich voor als een ideale processor en vereenvoudigt het programmeerwerk. Belangrijk daarbij is het eenvoudige geheugenbeheer en de eliminatie van vele soorten fouten door strenge controles bij compileren en uitvoeren.



Scala:

```
case class Clock (var hour: Int, var min: Int);
```

Java:

```
public class Clock {
    private int hour;
    private int min;

    public Clock(int hour, int min) {
        this.hour = hour;
        this.min = min;
    }

    public int getHour() {
        return hour;
    }

    public void setHour(int hour) {
        this.hour = hour;
    }

    public int getMin() {
        return min;
    }

    public void setMin(int min) {
        this.min = min;
    }

    public String toString() {
        ...
    }

    public boolean equals(Object arg1) {
        ...
    }
}
```



De Java-bibliotheken

Voor de productiviteit is het ook van belang dat er genoeg bibliotheken beschikbaar zijn. In een bibliotheek staat code die al door anderen is geschreven. Een populaire taal kan megabytes aan reeds geschreven code in bibliotheken hebben, waar je maar een paar regels aan toevoegt om bijvoorbeeld een website te maken. De bibliotheek is dus ook een overweging bij de keuze voor een taal. Java bestaat al ruim tien jaar en in die tijd zijn er heel veel bibliotheken ontwikkeld voor Java. Veelal is van deze bibliotheken ook de Java-code vrij (open source) beschikbaar.

>>iedere taal heeft zijn eigen levenscyclus

De ontwikkelomgeving

Naast het Java-platform is voor de productiviteit ook nog de ontwikkelomgeving van groot belang. Deze ontwikkelomgevingen worden ook wel 'Integrated Development Environment' (IDE) genoemd. Voor Java zijn er een aantal beschikbaar en Eclipse is hiervan het meest bekend. De IDE geeft de programmeur veel ondersteuning bij het programmeren; zo controleert het al tijdens het invoeren of de code correct is. Het bevat helpfuncties, ingebouwde debuggers en links naar bibliotheken.

Wet van de remmende voorsprong

De innovatie in een taal is omgekeerd evenredig met de leeftijd van de taal en het aantal gebruikers. Softwareproducten van een taal die al in gebruik zijn, noodzaken tot 'backwards compatibility'. Een aspect dat voormalig Java-eigenaar Sun altijd heel goed in de gaten heeft gehouden. Maar dat remt de innovatie tot het punt dat de taal stilstaat. Dan is het tijd om uit te kijken naar een nieuwe taal. Iedere taal heeft dus zijn levenscyclus.

What's next?

De laatste jaren duiken er steeds nieuwe talen op. Een aantal talen bouwt voort op wat er al was en maakt gebruik van de JVM met alle voordelen van dien. Voorbeelden zijn Ruby, Python en Groovy, die over het algemeen veel compactere code geven dan Java. Ook zijn innovaties in deze talen doorgevoerd zoals 'closures' waar een functie eenvoudig wordt uitgeschreven op de plaats waar die nodig is. Voor kleine functies bespaart dit veel typewerk. De Java-gemeenschap is al jaren aan het vergaderen over hoe ze dit alsnog in Java kunnen inbouwen.

Een belangrijk nadeel van de hierboven genoemde nieuwe talen is dat ze allemaal gebaseerd zijn op 'dynamic typing'. Dit betekent dat je fouten pas vindt bij het uitvoeren van het programma. Voor het bouwen van complexe programma's met grote teams zijn deze talen daarom niet geschikt. Je wilt juist dat de compiler je maximaal helpt met het vinden van mogelijke fouten in het programma, nog voordat het wordt uitgevoerd.

Scala

Een positieve uitzondering is de nieuwe taal Scala. De persoon achter deze taal is Martin Odersky, tegenwoordig professor aan de universiteit van Lausanne (EPFL) en een belangrijk persoon achter één van de laatste innovaties in Java: Generics. Een Scala-programma wordt uitgevoerd op de JVM en kan gebruik maken van de bestaande Java-bibliotheken. Scala is een taal met relatief veel innovatie. Het is een hybride taal waarin OO-concepten en functionele concepten samenkomen. Deze gedachtegang staat niet op zichzelf, maar is een trend. Ook Microsoft maakt hiervan gebruik in het .Net-platform door middel van LINQ. Zonder hier verder op in te gaan kan van LINQ gezegd worden dat het zwaar leunt op het gedachtegoed van de functionele talen.

Verder is belangrijk dat Scala gebruik maakt van 'static typing'. Dit betekent dat de compiler streng is en helpt in het voorkomen van vele soorten fouten. Een opvallende innovatie is 'type inference'. Dit betekent dat een programmeur niet altijd expliciet de types van variabelen en parameters hoeft te herhalen, met compactere code als resultaat. Eigenlijk bepaalt de compiler, indien mogelijk, zelf de types. Het wetenschappelijke idee achter het beredeneren van de types door de compiler komt ook weer uit de functionele wereld. Een andere innovatie zijn 'traits'. Dit zijn Java-interfaces waarin ook implementatie (velden en code) is toegestaan. Met traits kunnen lastige beperkingen van enkelvoudige overerving overwonnen worden.

De Scala-gemeenschap groeit snel en heeft een IDE ontwikkeld voor Eclipse die al redelijk werkt. Scala heeft hiermee alle ingrediënten in zich als potentieel opvolger van Java... toch?

Drempel

Het speelveld overziend heeft Scala inderdaad goede papieren, maar zo hard gaat het natuurlijk ook weer niet. De taaldrempel is voor de meeste gebruikers nu nog te hoog, omdat door de vele innovaties de taal Scala toch behoorlijk verschilt van de taal Java. Dat zien we vaker bij de overgang naar een nieuwe taal. Eerst zul je moeten investeren, voor je de vruchten van de verbeterde productiviteit kunt plukken. Ook werkt de IDE wel, maar is nog niet zo handig als dat men bij Java gewend is.

Daarnaast lijkt de Java-gemeenschap een beetje zijn kop in het zand te steken en liever nog een jaartje te vergaderen over een nieuwe verbetering aan de taal. Hier spelen wellicht ook politieke zaken die niets te maken hebben met de innovaties in de taal.

Zolang grote bedrijven een nieuwe taal niet oppikken, zal Scala een nicheproduct blijven. Een bedrijf als Google zou Scala het benodigde gewicht kunnen geven om over het kantelpunt heen te komen. De zoekgigant heeft ook vaak het lef getoond om iets nieuws uit te proberen. Gelukkig zie je al wel dat Scala kan meedraaien in het 'Summer of code'-programma van Google, waar het sponsor is van universitair vakantiewerk voor veelbelovende projecten.

Voor Technolution is belangrijk dat we het speelveld van de programmeertalen blijven volgen, dat we daarbij het kaf van het koren scheiden en dat we op tijd in de juiste trein stappen. Eén ding is echter zeker: er komt een dag dat Java weer gewoon een eiland in de Indische Oceaan is.

