Lezione 1

Introduzione alla programmazione con Java

Questa lezione illustra le motivazioni che sono alla base della nascita di Java, insieme alle prerogative che lo hanno reso un valido e celebre strumento per lo sviluppo di software multipiattaforma. La lettura dei seguenti paragrafi è caldamente consigliata a chiunque non si intenda dei meccanismi propri del linguaggio e del suo ambiente di esecuzione.

Vi presento Duke, la mascotte di Java. Non è ben chiaro cosa rappresenti, si dice sia stato concepito per essere semplice ma efficace. Sin dalle origini, ha accompagnato con la sua immagine ogni prodotto relativo al linguaggio. E' davvero un testimonial d'eccezione!

1.1 - Computer che interagiscono

L'irrefrenabile diffondersi dei personal computer ed il continuo espandersi di Internet hanno portato in primo piano il problema dell'interazione tra macchine distinte, già sentito da diverso tempo nelle università e negli ambienti di lavoro più tecnologicamente all'avanguardia. Benché i sistemi operativi di tipo Microsoft Windows, oggi, occupino la stragrande maggioranza delle postazioni individuali, basta guardare un po' più in là del proprio naso per rendersi conto di come l'universo dell'informatica sia frazionato in una miriade di costellazioni distinte, fatte di approcci differenti, di scelte non sempre omogenee e di piattaforme di ogni tipo.
La diversità, sia ben chiaro, è un fattore importante, e lo è tanto nell'hardware quanto nel software. Ogni soluzione si adatta a delle particolari esigenze meglio di altre. Altrettanto importante, comunque, è fare in modo che due o più piattaforme tra loro dissimili possano dialogare e cooperare senza incomprensioni dovute alla loro natura disunitaria. In assenza di un prerequisito del genere, qualsiasi rete sarebbe impensabile.
Il problema si accentua con l'entrata in scena di alcuni dispositivi di nuova generazione, che portano i computer a manifestarsi in forme assai diverse dal classico "tastiera-scatola-schermo" che tutti sanno facilmente figurare. Palmari, laptop, telefonini, postazioni bancomat e piccoli elettrodomestici intelligenti sono solo alcuni tra gli esempi che potrei riportare. Una cosa è certa: il mondo si sta informatizzando, e lo sta facendo piuttosto in fretta. Il nostro stile di vita, negli anni più recenti, è stato radicalmente mutato da questo progressivo imporsi dell'elettronica e dell'informatica, e molto deve ancora venire.
La cooperazione e lo scambio dei dati tra macchine distinte sono problemi che sono stati gradualmente ridotti dalla nascita di standard aperti. Non importa come funzioni internamente un sistema operativo, è sufficiente che sappia raccogliere ed emettere dati rispettando degli standard. Questo concetto si applica tanto ai file quanto alle reti. Nel primo caso si parla di *formati*. Si pensi ad un'immagine JPEG. Non importa su quale macchina o con quale software la si è realizzata la prima volta: giacché il formato JPEG è libero e conosciuto, si troverà sempre uno strumento software in grado di manipolare l'immagine sulla piattaforma desiderata. Nel caso delle comunicazioni di rete si parla invece di *protocolli*. Rispettando un medesimo protocollo, due o più macchine possono venire in contatto tra loro, comprendendosi e lavorando insieme. L'esempio, questa volta, lo offre il Web. Con un browser qualsiasi, su piattaforma Windows, ci si può tranquillamente collegare ad un sito Web ospitato da un sistema UNIX. Benché le due piattaforme citate siano profondamente diverse, il reciproco rispetto del protocollo HTTP permette il dialogo e lo scambio dei dati.

1.2 - Un linguaggio multipiattaforma

Cosa c'entra Java con l'interazione tra macchine distinte? Apparentemente nulla, perché

Java è un linguaggio di programmazione, e non un formato aperto o un protocollo pubblico. In realtà, Java è intimamente legato al discorso portato avanti sinora, perché il suo principale scopo è estendere al software la portabilità che nel tempo hanno acquisito i dati. Per quanto le informazioni possano essere universalmente comprese e trasferite, infatti, i programmi rimangono sempre intimamente legati alla macchina per la quale sono stati realizzati. La dimostrazione di ciò è sotto gli occhi di tutti: nessun programma concepito per un particolare sistema può essere avviato su una macchina di tipo differente. Un EXE di Windows, per riportare un semplice esempio, è un file privo di significato su Linux. Esistono dei linguaggi, come l'ottimo C++, che possono essere impiegati su più sistemi operativi. Questo perché, attraverso uno strumento chiamato compilatore, il codice scritto dal programmatore viene tradotto nel linguaggio macchina specifico della piattaforma utilizzata. Spostandosi da un sistema all'altro, e adoperando il compilatore corrispondente, si possono ottenere più versioni dello stesso programma. Tutto questo, però, avviene solo in linea teorica. Per compiere delle operazioni anche basilari, come ad esempio mostrare una finestra, ciascun software deve agganciarsi alle funzionalità offerte dal sistema operativo in uso. Ogni piattaforma, in base alla propria architettura hardware e software, fornisce questi servizi in maniera diversa. Pertanto, la conversione di un programma prevede sempre due fasi: la riscrittura di tutte le parti che si occupano dell'interazione con il sistema operativo e la ricompilazione del sorgente sulla macchina puntata. La prima operazione, in particolar modo, può risultare ardua e sconveniente. Per questo molti programmi sono disponibili solo per alcuni sistemi: evidentemente il produttore ha reputato sconveniente o fuori dalla sua portata la conversione verso altre piattaforme.

Porting

L'operazione di conversione di un programma in codice nativo da una piattaforma ad un'altra è chiamata porting. Il porting si suddivide in due fasi: l'adattamento del sorgente e la sua conseguente ricompilazione sulla macchina puntata. I programmi Java non necessitano di alcun porting: la portabilità del bytecode è la principale caratteristica dell'ambiente di Sun Microsystems.

Java nasce da un'intuizione che permette di superare queste oggettive difficoltà nello sviluppo e nella distribuzione del software. I programmi realizzati con Java sono multipiattaforma: per essere eseguiti su sistemi operativi differenti non necessitano né di una ricompilazione, né tanto meno della riscrittura di alcune loro parti. Il prodotto di una compilazione su Windows, ad esempio, può essere trasferito, così com'è, su Linux, per essere immediatamente eseguito.
I vantaggi di un approccio multipiattaforma sono immediatamente evidenti: non c'è bisogno di programmatori esperti conoscitori di ciascun sistema operativo per ottenere le conversioni richieste, non bisogna distribuire più pacchetti differenti di un medesimo programma e non si deve mai escludere una specifica piattaforma dal proprio target semplicemente perché si è valutata sconveniente un'ipotetica conversione.
Sun Microsystems, la casa che ha ideato Java e che controlla l'evoluzione del linguaggio, riassume tutto questo con un solo motto: "*write once, run everywhere*", ossia "*scrivi una volta, esegui ovunque*".

1.3 - Il bytecode e la Java Virtual Machine

Quando un programma Java viene compilato, il risultato dell'operazione non è un codice eseguibile. Se così fosse, si tornerebbe alla situazione di altri linguaggi come C++, ed i programmi non potrebbero essere trasportati liberamente da una piattaforma ad un'altra. Quando si compila un programma Java si ottiene un *bytecode*, ossia una codifica intermedia. Il bytecode, a differenza del codice eseguibile, non è indipendente. Preso così com'è, è del tutto inutile. Ad eseguire il bytecode ci pensa un apposito ambiente di

runtime, chiamato *Java Virtual Machine* (*Macchina Virtuale di Java*, abbreviato *JVM*). Questo componente deve essere presente nel sistema operativo sul quale si vuole eseguire del software Java.

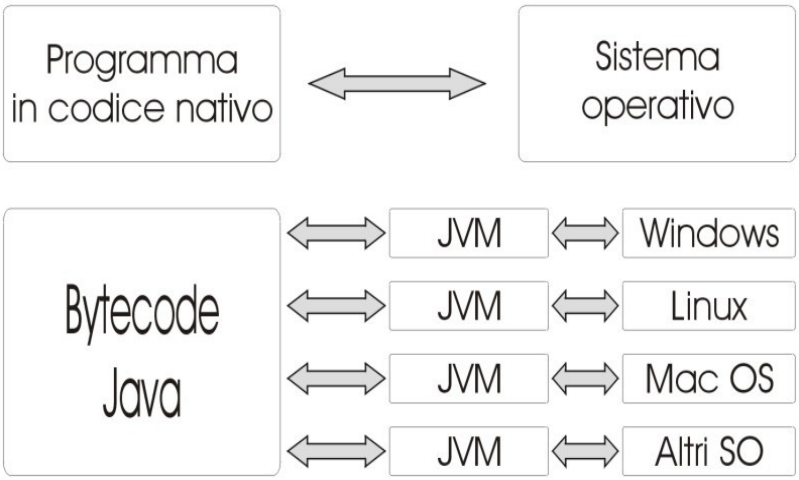

Figura 1.1

I programmi in codice nativo si possono rapportare esclusivamente con il sistema operativo per il quale sono stati concepiti, giacché fanno uso dei meccanismi interni di quest'ultimo. I programmi in bytecode Java, al contrario, utilizzano il ponte offerto dalla Java Virtual Machine (JVM), l'unico componente dal quale dipendono. Qualsiasi sistema dotato di una JVM adatta, pertanto, può eseguire un bytecode Java, senza che questo debba essere rivisto, adattato o ricompilato.

La macchina virtuale, oltre ad eseguire e supervisionare il bytecode, offre anche tutti i servizi di cui un programma può normalmente avere bisogno. Un codice eseguibile deve necessariamente interfacciarsi con il sistema operativo, richiedendo ad esso tutte le peculiari funzionalità di cui necessita. Un programma Java, invece, richiede i servizi direttamente alla macchina virtuale. Mentre ciascuna piattaforma dispone di funzionamenti interni differenti, tutte le Java Virtual Machine offrono al programmatore gli stessi identici strumenti: il codice necessario per aprire una finestra passando attraverso la Java Virtual Machine di Windows, ad esempio, è esattamente lo stesso richiesto dalla macchina virtuale di Linux.

Compilato o interpretato?

Un linguaggio si dice *compilato* quando il suo sorgente, attraverso un *compilatore*, viene tramutato in codice nativo, immediatamente eseguibile sulla macchina per la quale è stato concepito. Non è necessario trasportare il sorgente insieme agli eseguibili, che sono autosufficienti e garantiscono alte prestazioni. Alcuni esempi di linguaggi compilati sono Pascal e C/C++.

Un linguaggio si dice *interpretato* quando il suo sorgente viene eseguito, istruzione dopo istruzione, da un apposito strumento chiamato *interprete*. In questo caso, codice sorgente e software sono un tutt'uno. I linguaggi interpretati garantiscono usualmente prestazioni inferiori rispetto ai linguaggi compilati. In compenso, però, possono essere trasferiti da una macchina all'altra. La loro natura aperta favorisce lo studio, l'adattamento e la revisione dei programmi. Esempi di linguaggi interpretati sono Perl, Lisp ed i Basic del passato.

1.4 - I vantaggi

Oltre alle caratteristiche di portabilità appena elencate, che facilitano lo sviluppo di software compatibile con più piattaforme, Java gode di numerose altre buone caratteristiche. Un nuovo linguaggio di programmazione viene usualmente concepito per rendere semplice e lineare ciò che prima non lo era. Quando il linguaggio C++ venne ideato, i programmatori rimasero stupefatti dalla sua linearità e dalla sua semplicità di impiego. Inoltre C++ non rinnegava affatto quanto conquistato con il linguaggio C. Le esigenze, ad ogni modo, cambiano con il trascorrere del tempo. Benché C++ difficilmente potrà cadere in totale disuso durante questo ed il prossimo decennio, in pochi si permettono oggi di ritenerlo un linguaggio semplice. Alle esigenze degli anni passati,

infatti, se ne sono aggiunte di nuove, alle quali C++ non può rispondere ottimamente. Esistono strumenti buoni per certi scopi, ma non ideali per altri. Quando nasce una nuova esigenza, nasce con essa un nuovo strumento che possa soddisfarla in maniera semplice e lineare. Java è un linguaggio fresco. Ha raggiunto la maturità da poco ed ora è nel pieno delle sue forze. Risponde in maniera semplice a tutte le principali esigenze attuali. Non rinnega quanto proviene dal passato, ma lo arricchisce con diverse novità, volte a soddisfare le richieste degli sviluppatori. Java è:

- **Completamente orientato agli oggetti**, più di quanto non lo sia C++. La possibilità di riutilizzare il codice e di gestire facilmente i progetti di vaste dimensioni sono i principali pregi di questo approccio.
- **Sicuro**. L'esistenza di un ambiente di runtime coeso come la Java Virtual Machine garantisce una sicurezza del codice mai avuta prima. La macchina virtuale, frapponendosi tra i programmi ed il sistema operativo, controlla ogni operazione effettuata. Con Java è impossibile utilizzare la memoria in maniera illecita. Ad ogni programma possono essere associati differenti profili di sicurezza, in maniera esplicita o implicita. Un software *untrusted* (*non di fiducia*), ad esempio, non può né leggere né scrivere dati sul disco fisso dell'utente, così come non può accedere in piena libertà alle risorse di rete. Usando la dovuta accortezza, qualsiasi software Java può essere impiegato senza timori.
- **Robusto**. I software robusti sono stabili ed affidabili. Java fornisce dei modelli che aiutano lo sviluppatore nel gestire le situazioni inattese, fronteggiandole per tempo ed evitando ogni spiacevole inconveniente. La memoria, inoltre, è amministrata dalla macchina virtuale, che rimuove automaticamente le risorse non più utilizzate. In C++, al contrario, questo compito è completamente demandato al programmatore. Benché la gestione manuale della memoria costituisca un grado di libertà in più, nelle applicazioni complesse può divenire davvero difficile scrivere del codice che non vada a generare errori ed incoerenze, inficiando la robustezza del software prodotto.
- **Pensato per la rete**. Java è nato per rispondere alle esigenze della rete. Scrivere software orientati al networking è semplice e divertente. Esiste un connubio molto forte, inoltre, tra Java ed il Web.
- **Dinamico**. Il bytecode prodotto dal compilatore non è un blocco monolitico, ma è suddiviso in tante parti dinamiche che possono essere trasferite, scambiate e recuperate con grande semplicità, anche durante l'esecuzione. La riusabilità del codice già compilato è massima. Non c'è bisogno di registrare nel sistema operativo i differenti componenti condivisi di cui fa uso un software. Le installazioni e gli aggiornamenti sono semplici e veloci, poiché si basano spesso sulla semplice copia dei file.
- **Multithreaded**, ossia supporta nativamente la *programmazione concorrente*. Un programma Java può fare più cose contemporaneamente. Il modello offerto al programmatore consente di scrivere software multithreaded senza dover badare agli aspetti più strettamente tecnici della concorrenza, dovuti al dialogo con il sistema operativo. La sincronizzazione, inoltre, è gestita in maniera stabile ed elegante.
- **Concepito dai programmatori per i programmatori**. Si tratta di un aspetto molto importante, lo stesso che ha decretato il successo di C/C++ ed il declino di altri linguaggi pensati per ambiti troppo specifici e realizzati con mentalità e strutture logiche non proprie del programmatore.

1.5 - Gli svantaggi

Naturalmente, Java non è privo di controindicazioni. Se fosse diversamente, non si spiegherebbe come mai tutto il software di questo pianeta non venga distribuito sotto forma di bytecode. Ci sono delle limitazioni di cui bisogna sempre tener conto:

1. Java non è adatto per scrivere componenti che debbano interfacciarsi troppo direttamente con il sistema operativo o con l'hardware. Sarebbe impossibile, ad esempio, scrivere in Java un driver per un modem. Per questo genere di operazioni, infatti, devono essere impiegati dei linguaggi di più basso livello: solo questi, a fronte di una maggiore difficoltà di impiego, consentono un uso diretto di tutte le caratteristiche hardware e software necessarie. Java, invece, resta dietro il muro della sua macchina virtuale, oltre il quale non può davvero spingersi.
2. La Java Virtual Machine, interpretando ed amministrando il bytecode, si frappone tra il software ed il sistema operativo. Benché questa caratteristica favorisca la portabilità, la sicurezza, la robustezza, la dinamicità e la semplicità, bisogna considerare il suo più grave difetto: il calo delle prestazioni. Il bytecode, prima di essere eseguito, deve attraversare dei processi che impiegano tempo e spazio per essere portati a compimento. Un programma compilato in codice realmente eseguibile, al contrario, non è soggetto a manovre intermedie e può essere elaborato all'istante dal sistema operativo. Quando l'efficienza è una caratteristica prioritaria, quindi, conviene accantonare Java in favore di altri linguaggi di più basso livello, come C/C++. In casi estremi, addirittura, è meglio ricorrere a routine ottimizzate mediante un assemblatore.
3. Affinché il bytecode possa essere eseguito, è necessario che una Java Virtual Machine sia presente nel sistema. Se non si dispone di una macchina virtuale e non si ha la possibilità di installarla, Java è inutile.

I tre problemi elencati, in definitiva, si manifestano quando si cerca di impiegare Java in un ambito che non gli compete. Nonostante questo, Sun Microsystems ed altre aziende implicate nell'evolversi di Java si sono date parecchio da fare per ridurre al minimo le controindicazioni del linguaggio:

1. Le più recenti versioni di Java dispongono di funzionalità utili per interagire a basso livello con l'hardware. Ad esempio, le *Java Communications API* sono un insieme di strumenti che consentono un controllo diretto sulle porte seriali e parallele del PC, mentre le *Java Sound API* si interfacciano con i driver delle schede sonore in maniera molto più potente ed elastica di quanto si potesse fare prima della loro emissione. Inoltre, è possibile utilizzare parti di codice nativo all'interno di un software Java. In questa maniera, le routine che necessitano di accessi di basso livello possono essere efficacemente realizzate con linguaggi adatti allo scopo, che compilino in linguaggio macchina, per essere poi richiamate dall'interno di una più vasta e portabile sovrastruttura Java. Gli sforzi di conversione del software, grazie a questo espediente, sono minimi (anche se non nulli, visto che il codice nativo è differente di macchina in macchina), e le limitazioni di Java possono essere abilmente scavalcate persino in casi estremi.
2. Gli ultimi anni hanno portato degli stupefacenti progressi dal punto di vista delle prestazioni. In principio, la macchina virtuale interpretava il bytecode comando dopo comando, a tutto svantaggio dell'efficienza. Tanto Sun quanto altri marchi (un nome rappresentativo è IBM) hanno incrementato le prestazioni includendo la possibilità di una compilazione *Just in Time* (*JIT*) del bytecode. La tecnica consiste nel dotare

la macchina virtuale di un secondo compilatore integrato, capace di trasformare le parti più pesanti del bytecode in codice nativo, prima della loro esecuzione. In questa maniera, si raggiungono prestazioni che si distaccano di poco da quelle ottenibili con C/C++, senza sacrificare la portabilità. Inoltre, bisogna anche considerare la crescita tecnologia che permea costantemente il mondo dell'informatica. Un complesso software Java avrà sicuramente dei problemi di efficienza se eseguito da un obsoleto 200 MHz con poca memoria a disposizione. Il calo delle prestazioni, però, non è altrettanto evidente sulle configurazioni messe in commercio più recentemente. Insomma, l'efficienza di Java è un problema che tende a risolversi tanto con l'emissione di nuove macchine virtuali quanto con l'incedere della crescita tecnologica. Già si mormora di complessi videogiochi 3D completamente sviluppati in Java.

3. Attualmente, è praticamente impossibile imbattersi in un sistema che non disponga di una Java Virtual Machine. Non c'è piattaforma sulla quale non sia teoricamente possibile eseguire il software Java. Alcuni sistemi operativi, come MacOS X, includono la macchina virtuale direttamente nella loro installazione base, considerandola un tutt'uno con gli altri componenti. Diversi software di uso comune, come i browser, installano automaticamente una Java Virtual Machine su quei sistemi che ne sono sprovvisti nella loro configurazione originaria. Male che vada, il runtime di Java pesa pochi megabyte ed è distribuito gratuitamente. Insomma, la capillare diffusione e la grande notorietà di Java, al giorno d'oggi, sono dati di fatto.

1.6 - Gli ambiti

Preso atto dei vantaggi e degli svantaggi di Java, è finalmente possibile stilare una lista degli ambiti ideali per una corretta e coerente applicazione del linguaggio:

- **Grandi progetti**. Java, per propria natura, è uno strumento ideale per la gestione di grandi progetti, ai quali possono lavorare più persone simultaneamente. La programmazione orientata agli oggetti favorisce l'organizzazione, la manutenzione ed il riutilizzo del codice.
- **Applicazioni per la rete**. Java ama la rete, su questo non c'è ombra di dubbio. Tutto il software che punta sulla rete e sull'interoperabilità tra macchine distinte si presta benissimo ad essere sviluppato con Java, anche per via della portabilità del bytecode. Il linguaggio di Sun è adatto tanto alle applicazioni server quanto alle loro controparti client. La sicurezza, inoltre, è un elemento preponderante nello sviluppo di software per la rete, e Java fa della sicurezza uno dei suoi maggiori punti di forza.
- **Applicazioni per il Web**. Tra i software che lavorano in rete, hanno oramai assunto particolare risalto le applicazioni per il Web. Java è eccellente per il loro sviluppo, soprattutto nella programmazione lato-server. Le tecnologie *Servlet* e *JavaServer Pages* (*JSP*) costituiscono la punta di diamante di un vasto insieme di strumenti adatti o adattabili al Web. Dal punto di vista della programmazione lato-client, Java getta in tavola due prestigiose carte: le *Applet* e la tecnologia *Java Web Start* (*JWS*), che trovano significative applicazioni soprattutto nelle reti aziendali.
- **Applicazioni enterprise**. Le caratteristiche precedentemente elencate, insieme con altre, confluiscono tutte in una naturale predisposizione di Java per lo sviluppo di software per le imprese. In Java sono incorporati diversi meccanismi all'avanguardia, che consentono un facile accesso alle basi di dati e la possibilità di distribuire il calcolo in maniera efficiente e sicura.
- **Software per i dispositivi portatili**. Java è la tecnologia più all'avanguardia in

questo settore. Grazie alle sue doti di portabilità, ben si presta all'impiego su apparecchi mobili di ogni tipo e natura. La maggior parte dei produttori hanno adottato Java per lo sviluppo di software destinato ai dispositivi wireless. Sun Microsystems, d'altra parte, incoraggia il settore rilasciando continuamente nuove soluzioni disegnate per facilitare l'integrazione di Java in ogni genere di dispositivo portatile.

All'infuori di quanto già detto, è difficile tracciare dei confini netti sulla validità di Java. Molte tra le categorie elencate, inoltre, si compenetrano. L'importante è saper valutare le esigenze, per poter poi scegliere, di volta in volta, la soluzione migliore.
Dagli ambiti di applicazione di Java, sono invece esclusi tutti quei software che necessitino di prestazioni elevatissime e di una forte integrazione con la macchina sfruttata. In casi come questi, non è possibile risolvere efficacemente tutti i problemi solo con Java.

1.7 - Altri vantaggi

Ci sono ulteriori vantaggi nell'adozione di Java, che elencherò velocemente. Per prima cosa, Java è una tecnologia dalle specifiche aperte. Benché solo Sun Microsystems abbia diritto di vita e di morte sulla propria proposta, diversi produttori contribuiscono alla causa di Java nei modi più disparati. Le specifiche aperte consentono la libera implementazione del linguaggio e della sua macchina virtuale a chiunque se la senta di imbarcarsi in un progetto di tale portata. Quello di Java è uno dei pochi casi in cui il controllo esclusivo sull'evolversi di una tecnologia da parte di un'unica azienda è stato più un bene che non un male. In poche parole, nel tempo sono sempre state impedite le estensioni proprietarie sviluppate da terzi, mentre sono state favorite le implementazioni rispettose della formulazione originaria. Per la prima volta da tanti anni a questa parte, con Java arriva una tecnologia che sia allo stesso tempo aperta ed omogenea. Non possono sussistere differenze eclatanti tra le implementazioni di un produttore e quelle di un suo concorrente. Solo con una politica di questo genere è stato possibile mantenere inalterato il motto "*write once, run everywhere*".
I principali strumenti per lo sviluppo di software Java, come seconda cosa, sono gratuiti e liberamente adoperabili senza alcuna limitazione tecnica o legale. Chi sceglie il linguaggio per uso personale, didattico o no-profit, non si ritroverà inutili bastoni tra le ruote. Anche chi vuole applicare Java ad un settore business non è soggetto a limitazioni.
Per finire Java e l'Open Source sono due mondi che si toccano, e spesso l'uno favorisce la causa dell'altro. Numerosi strumenti di sviluppo e tantissimi software Java vengono distribuiti insieme al loro codice sorgente. I vantaggi di questa caratteristica sono una maggiore sicurezza, una più veloce risoluzione dei problemi, la presenza di una comunità attiva che collabora alla crescita e all'arricchimento delle caratteristiche disponibili, la possibilità di consultare i sorgenti per scopi didattici e la facoltà di modificarli per meglio renderli consoni alle proprie esigenze. Grazie a queste vaste possibilità, Java ha preso piede anche in ambito accademico.

1.8 - I concorrenti di Java

Anzitutto, bisogna sfatare il mito secondo il quale Java e C++ sono due linguaggi contrapposti. Più che antagonisti, sono alleati ed entrambi preziosi allo stesso modo. La leggenda ha preso piede subito dopo l'emissione delle prime versioni di Java, quando ancora non era chiara la differente validità del nuovo strumento. Probabilmente, tale credenza popolare è alimentata anche dai numerosi confronti che, in sede didattica, si impiegano per illustrare le somiglianze e le differenze tra i due approcci esaminati. L'importante è leggerli in chiave giusta. I concorrenti di Java, per farla breve, vanno

ricercati in quello che è il suo stesso settore di applicazione, e non altrove.
Fino all'avvento della piattaforma .NET di Microsoft, Java si è mosso ed è cresciuto senza importanti rivali. Microsoft .NET è un ambiente assai simile a quello proposto e portato avanti da Sun Microsystems, e fa proprie molte delle conquiste di Java. Entrambi si basano sulla programmazione orientata agli oggetti, sull'esistenza di un codice intermedio, su un ambiente di runtime che lo gestisca e sulla naturale predisposizione all'uso in rete. Il terreno dello scontro è costituito dalla sicurezza, dalla robustezza, dal networking e dalle applicazioni Web. La soluzione di Microsoft è però più giovane, con tutti i pro ed i contro che ne derivano. Che .NET sia abbastanza maturo da rivaleggiare ad armi pari con Java, è ancora tutto da dimostrare. Il principale vantaggio di .NET su Java risiede nella possibilità di far cooperare diversi linguaggi nella medesima applicazione, in maniera del tutto naturale. Il linguaggio principe di .NET è C#, assai simile a Java nella sintassi e nel modo di funzionare, anche se reintegra alcuni concetti più caratteristici di C/C++. Il runtime di .NET, inoltre, è davvero ben ottimizzato. Di contro, la piattaforma di Microsoft manca quasi totalmente di portabilità, di diffusione e di una comunità tanto vasta quanto quella dei programmatori Java. Microsoft, inoltre, deve ancora scrollarsi di dosso la cattiva reputazione che si è guadagnata relativamente al delicato tema della sicurezza.

Microsoft e Java

Microsoft, in passato, ha supportato Java con dei compilatori e delle macchine virtuali non completamente conformi alle specifiche di Sun Microsystems. In particolare, la casa di Redmond estese il linguaggio con delle caratteristiche che vincolavano il bytecode prodotto ai soli sistemi di tipo Windows. Queste iniziative sono state causa di una pesante sconfitta di Microsoft in sede legale e della conseguente inimicizia con Sun. Microsoft ha interrotto lo sviluppo della propria macchina virtuale e, per promuovere .NET, si è prodigata in alcune mosse di marketing pensate per ledere la popolarità di Java, come l'esclusione del runtime del linguaggio dall'installazione base di Windows XP. Di conseguenza, sui sistemi Windows è sempre necessario installare una recente macchina virtuale, giacché questa potrebbe essere assente o non aggiornata.

Java e JavaScript

Bisogna prestare particolare attenzione nel non confondere Java e JavaScript, due tecnologie che condividono il nome per ragioni di puro e semplice marketing. Java, come si spiega in questo corso, è un linguaggio di programmazione, e come tale ha tanti scopi e tanti risvolti. JavaScript, invece, è un linguaggio di scripting. Non è affatto la stessa cosa.