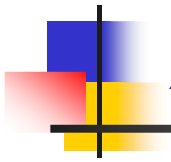
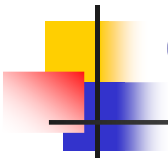


Linguaggi

Corso di Laurea Ingegneria Informatica (M-Z)
A.A. 2006-2007

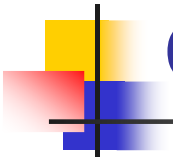
Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

Introduzione a Java



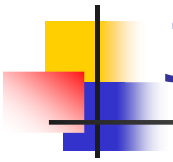
Code-Name Green

- Nome del progetto Sun con l'obiettivo di fornire "intelligent consumer-electronic devices".
- Il risultato fu Oak
  - Un linguaggio basato su C++
  - Creato da James Gosling
  - Il nome del nuovo linguaggio fu cambiato in Java
  - E' fortemente ispirato al C++ ma riprogettato senza il requisito della piena compatibilità con il C (a cui però assomiglia)



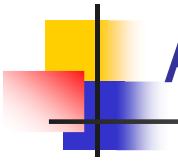
Green in the red?

- Il progetto Green non andò lontano
  - Il mercato degli "intelligent consumer-electronic devices" crebbe lentamente
  - Sun non si impose in tale settore di mercato
  - Fu sul punto di essere cancellato
- L'esplosione del World Wide Web nel 1993 salvo il progetto Green
- Java fu ripensato come linguaggio per fornire contenuto dinamico alle pagine web
- Java fu formalmente annunciato nel 1995



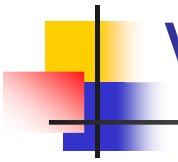
Java

- Java nasce per applicazioni "embedded"
- Si diffonde attraverso il concetto di applet come piccola applicazione da eseguirsi dentro un browser Internet
  - grafica portabile ed eseguibile ovunque
  - modello di sicurezza "sandbox"
- Può benissimo essere usato come linguaggio per costruire applicazioni anche non per Internet anche non grafiche (rimane un linguaggio general purpose)



Alcune date dello sviluppo Java

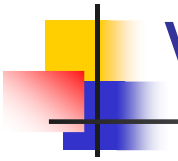
- Java 1.0 – rilasciato nel 1995 dalla Sun
- Java 1.1 – rilasciato nel 1997
- dalla v. 1.2 del 1998, riscrittura significativa
- attualmente 1.5 (5.0)



Versioni di Java

Le versioni di Java2

- 1.2, la prima
- 1.3, sostanzialmente equivalente alla 1.2
- 1.4, introduce alcune limitate funzionalità nuove incompatibili con il passato
- 1.5, ulteriore riscrittura significativa, con molte funzionalità nuove ed incompatibili



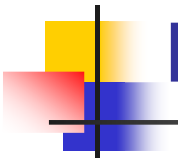
Versioni di Java

Java 2 v. 1.5 = Java 2 5.0

- è stato alla fine chiamato Java 2 v. 5.0 per sottolineare le novità rispetto alla versione precedente

Il problema di Java 2 5.0

- il codice compilato è incompatibile con le vecchie macchine virtuali
- le funzionalità vanno utilizzate in modo controllato

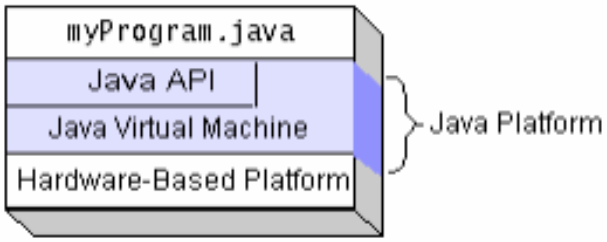


La Piattaforma Java

- Piattaforma: ambiente hardware o software dove sono eseguiti i programmi (Windows 2000, Linux, Solaris, MacOS)
- Una piattaforma in genere può essere descritta come una combinazione di sistema operativo e hardware
- la **Java platform** è solamente software e viene eseguita al di sopra di altre piattaforme basate sull'hardware

La Piattaforma Java

- La piattaforma consiste di due elementi:
  - Java Virtual Machine (**JVM**)
  - Java Application Programming Interface (**Java API**), ovvero una collezione di software pronti per l'uso, ad esempio per gestire Graphical User Interface (GUI), organizzati in librerie di classi e interfacce correlate (**packages**)

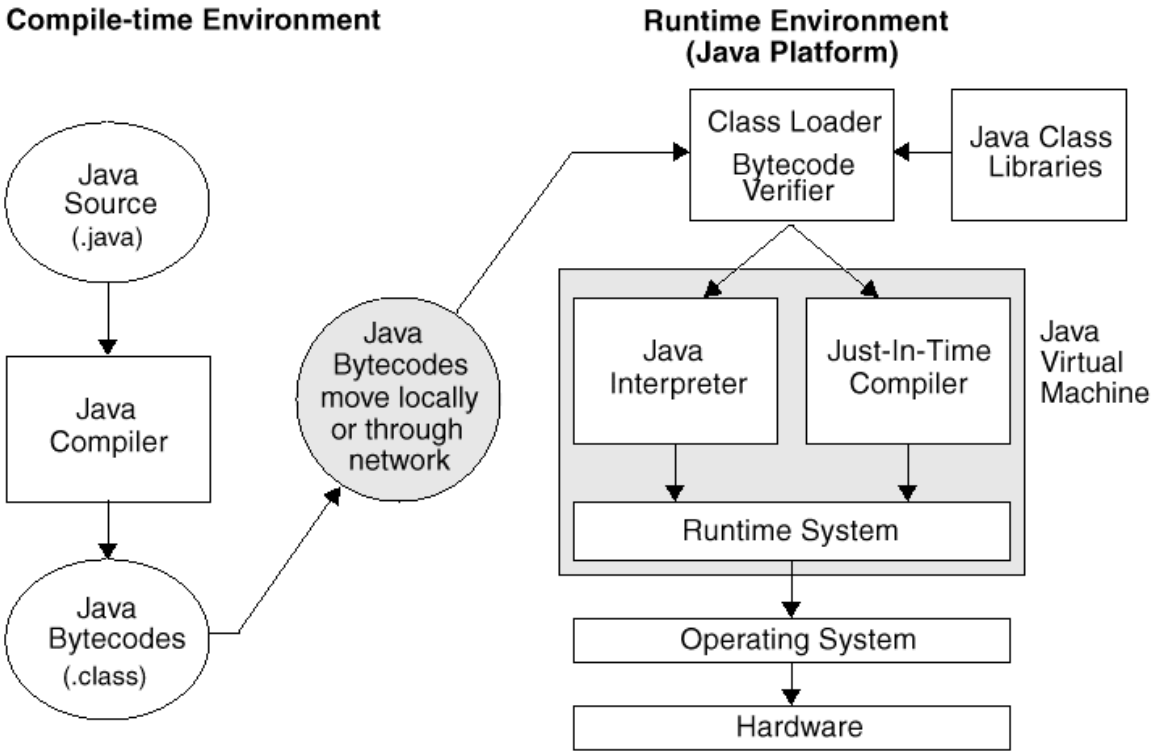


Java API e Java VM isolano il programma dall'hardware

La Piattaforma Java

- Il codice Java viene **compilato** in un linguaggio intermedio chiamato **bytecode**
- Il bytecode è **interpretato** a run-time dalla JVM e convertito nel linguaggio macchina del calcolatore su cui è richiesta l'esecuzione;
- la JVM è in quindi un computer virtuale sviluppato per rendere indipendente dalla macchina il codice Java: paradigma "*write once, run anywhere*", talvolta mutato dai detrattori in "*write once, debug anywhere*"
- La **portabilità** del codice è assicurata dalla garanzia progettuale che i tipi di data abbiano comportamento standard al variare della piattaforma (ad esempio, i reali sono *IEEE-compliant*);
- in alternativa, il bytecode può essere compilato tramite un ***just in time*** (***JIT***) ***compiler***, qualora occorrano prestazioni maggiori.

La Piattaforma Java



La Piattaforma Java

Esistono **edizioni** differenti della piattaforma Java

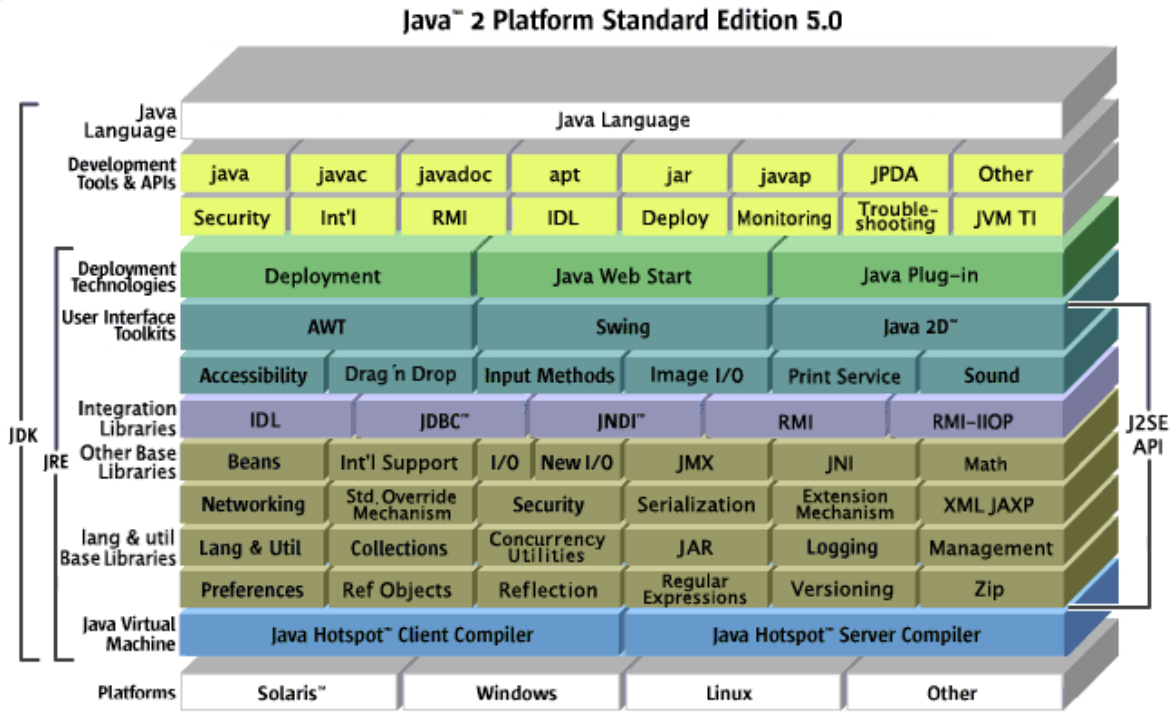
- **Standard Edition (J2SE)**
  - Librerie di base per lo sviluppo di applicazioni desktop (client applications) incluso AWT e Swing
  - Consente di eseguire applicazioni e applet
- **Enterprise Edition (J2EE)**
  - per lo sviluppo di applicazioni lato server
    - Per sviluppatori Web (EJB's, Servlets e JSPs)
    - richiede J2SE
- **Micro Edition (J2ME)**
  - per lo sviluppo di applicazioni mobili
    - librerie ridotte e più piccole
    - Implementa un subset delle funzionalita di Java

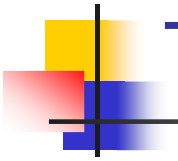
La Piattaforma Java

Ogni edizione si compone di:

- una serie di specifiche (documenti)
- una serie di strumenti (es: compilatore, debugger ecc.)
- una serie di librerie o API (Application Program Interfaces)
- schematizzando (eccessivamente) è possibile dire che i livelli più complessi includono i più semplici

Java2 – Platform Edition 5.0

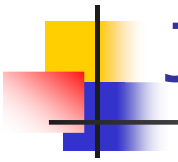




Tipi di Programmi

Java permette la realizzazone di 4 tipi di programmi:

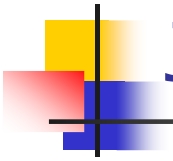
- Application
- Applet
- Servlet
- Beans



Java Application

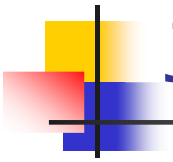
- Standalone program nel senso che richiede solo la JVM per essere eseguito
- Non richiede un programma host (come un browser) per l'esecuzione.
- Il metodo "main" è utilizzato come entry point e deve quindi essere presente
- Il metodo main deve avere la signature

  public static void main(String args[ ])



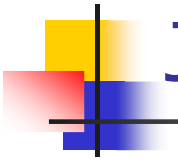
Java Applet

- Piccoli programmi tipicamente scaricati da un server su una macchina client
- La JVM è costruita all'interno del browser o di un apposito programma (appletviewer), che agiscono come programma host per l'*applet*.
- Un applet è tipicamente lanciato dentro un file HTML
- Un applet può essere inserito in un documento HTML con l'apposito tag <applet>
- Un *applet* generalmente opera sotto una gestione sicura che impone un *sandbox security*.
- Ciò previene che un applet possa eseguire operazioni potenzialmente pericolose come leggere e scrivere su un disco.
- Un applet è derivato dalle classi **`Applet`** o **`JApplet`**.



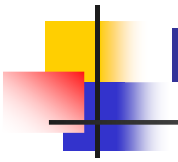
Java Servlet

- Una *servlet* è un programma che viene eseguito su una macchina server, tipicamente con l'obiettivo di processare una richiesta di un client
- L'host per una *servlet* è di norma un Web server, che fornisce una JVM.
- Una *servlet*, come un applet, è normalmente lanciato da un browser ma, a differenza degli applet, viene eseguito sul server.
- Una *servlet* comunemente esegue delle operazioni su un database e genera pagine Web dinamiche che vengono mostrate al client.
- Una classe *servlet* implementa l'interfaccia standard **`Servlet`** o è derivata da una classe che implementa tale interfaccia.



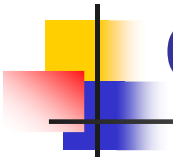
Java Bean

- Un *bean* è un componente software, cioè, una parte di software precostruito che può essere integrato con altri per costruire una applicazione.
- Un *bean* generalemte ha uno scopo preciso. Esempi sono calendar beans, login beans, email beans…
- Una classe *bean* tipicamente implementa l'interfaccia standard `Serializable`.



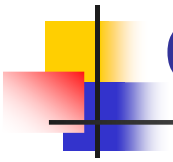
Il Java Development Kit (JDK)

- Il JDK della Sun Microsystems è l'insieme di strumenti di sviluppo che funge da "riferimento ufficiale" del linguaggio Java:
  - non è un ambiente grafico integrato: è solo un insieme di strumenti da usare dalla linea di comando
  - non è particolarmente veloce ed efficiente (non sostituisce strumenti commerciali)
  - però funziona, è gratuito ed esiste per tutte le piattaforme (Win32, Linux, Solaris, Mac..)
  - riferimento: http://java.sun.com sito ufficiale della Javasoft per scaricare tutto il software e la documentazione



Cosa serve per programmare

- J2SE: Installare il "Java Development Kit" (JDK)
- Java 2 Platform Standard Edition 1.4.2 o Java 2 Platform Standard Edition 1.5 detta **JAVA 5**, è l'ultima release ufficiale della piattaforma java.
- Java5 è quella che si definisce come major poichè apporta diverse novità nel modo di programmare in java e di gestire la JVM.
- Miglioramenti in termini di
  - Scalabilità
  - Performance
  - Monitoring
  - Gestibilità

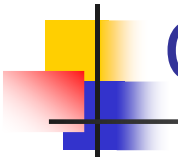


Cosa serve per programmare

Il kit di sviluppo jdk è:

- distribuito gratuitamente dalla Sun Microsystem
- comprende alcuni tools, come il compilatore, il runtime e il debugger, numerosi programmi di esempio e l'insieme completo dei sorgenti della ricchissima libreria che accompagna il linguaggio
- Il pacchetto di installazione denominato **'Jdk-1.5'** è scaricabile alla pagina

  http://java.sun.com/javase/downloads/index.jsp
- Terminato il download (il file zippato occupa 49,7 Mb) basta eseguire ed installare tutti i tools proposti, compresa la **JRE 1.5 (Java Runtime Enviroment)**, necessaria all'esecuzione di file 'jar' e delle applets di Internet.

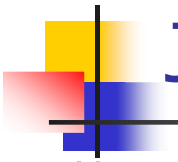


Configurare il Sistema

- Il compilatore della Sun è
  - un programma senza interfaccia grafica
  - deve essere invocato dalla console di comandi MS-Dos
  - Necessario settare un'apposita variabile di ambiente del computer, detta PATH. Il settaggio della path permette anche di eseguire i comandi java, jar, javac da qualsiasi directory del sistema
- Per sistema operativo Windows XP bisogna modificare il path nelle "VARIABILI DI AMBIENTE" del computer, aggiungendo al PATH già esistente la riga

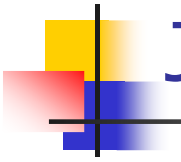
  C:\<percorso_di_installazione_scelto>\ JDK1.5.0_01\BIN
- Sempre nella stessa finestra delle Variabili di Ambiente è opportuno creare una variabile di ambiente di nome CLASSPATH, il cui valore indichi al compilatore java dove cercare le classi di eventuali package importati all'interno di un file java. Il valore consigliabile è

  .; C:\<percorso_di_installazione_scelto>\ JDK1.5.0_01\lib



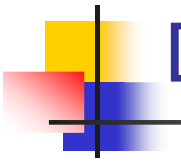
Java API

- Un programmatore Java deve:
  - costruire oggetti isolati e robusti
  - non influenzati da altri oggetti presenti nel programma
  - derivare i nuovi oggetti dai vecchi in modo veloce e additivo
  - documentare le caratteristiche di un oggetto, ossia corredatare il codice con una documentazione facilmente comprensibile anche da persone diverse rispetto agli autori.
- Sun mette a disposizione delle librerie dette Package, a cui l'utente può accedere e che offrono varie funzionalità. L'insieme dei package Java è chiamato Java **API (Application Programming Interface)**: non si può programmare in Java senza consultare le API.



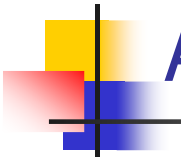
Java API

- Java include classi per costruire interfacce grafiche (GUI), per gestire I/O, networking, web applications, eccetera, denominate nel complesso Java **Application Programming Interface** (API), codice già scritto organizzato in packages relativi ad argomenti comuni
- Un **package** è una collezione di classi e interfacce (correlate) che fornisce una gestione dei nomi (namespace management) e dell'accesso (access protection)
- per usare una classe o un'interfaccia in un package:
  - usare il nome qualificato completo *java.awt.BorderLayout*
  - importare la classe o l'interfaccia import *java.awt.BorderLayout*
  - importare tutto il package *import java.awt.**
- **interfacce**: una collezione di definizioni di metodi e costanti che potranno essere implementate da classi (diremo che queste classi implementano l'interfaccia)



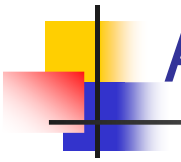
Documentazione

- La documentazione completa di tutorial è racchiusa nel file zippato "jdk-1_5_0-doc" (di circa 45 Mb) e si può liberamente scaricare dal sito http://java.sun.com/docs
- Terminato il download, si decomprime il file in una cartella (che occuperà circa 245 Mb) del computer; per consultare direttamente le Api basta eseguire file "index.htm" presente nella cartella "...\docs\api\".



Ambiente di sviluppo

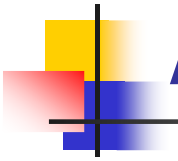
- un semplice text editor (esempio Notepad) è sufficiente, purchè si salvi con estensione ".java" (attenzione al mascheramento delle estensioni operato dal SO)
- Per esempio, volendo realizzare un programma che visualizzi su schermo la frase "Ciao a tutti", bisogna creare il file denominato ad esempio "Ciao.java".
- Per compilarlo è necessario aprire la finestra del prompt di MS-DOS ed invocare il comando "javac Ciao.java"
- se non vi sono errori il compilatore crea il file "Ciao.class" nella stessa directory.
- Per eseguire il file, si invoca il comando "java Ciao" (nome del file senza estensione)
- I comandi "javac" e "java" sono riconosciuti dal DOS solo se si è configurato correttamente il path!



Ambiente di sviluppo

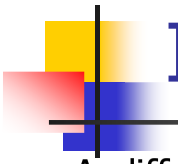
Esistono molti strumenti tesi a migliorare il JDK, e/o a renderne più semplice l'uso

- editor con "syntax highlightling", ad esempio Scite http://www.scintilla.org/SciTE.html
- ambienti integrati freeware o shareware che, pur sfruttando il JDK, ne consentono l'uso in modo interattivo e in ambiente grafico
  - JCreator LE, EditPlus, Forte, JaSupremo, etc...
  - Eclipse http://www.eclipse.org
- ambienti integrati commerciali, dotati di compilatori propri e debugger



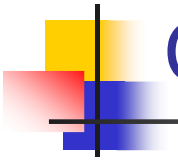
Ambiente di sviluppo

- Eclipse è un IDE (ambiente di sviluppo integrato)
- progetto open source legato alla creazione e allo sviluppo di una piattaforma di sviluppo ideata da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software
- Usato per la produzione di software di vario genere
- Fornisce
  - Un completo IDE per il linguaggio Java (JDT, "Java Development Tools")
  - un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tools")
  - plug-in che permettono di gestire XML, PHP
  - Plug-in per progettare graficamente una GUI per un applicazione JAVA (Eclipse VE, "Visual Editor"),



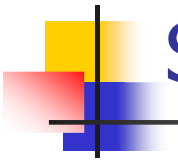
Internazionalizzazione

- A differenza di gran parte dei linguaggi Java usa un codice a 16 bit chiamato **Unicode**
- il primo codice ASCII è a 7 bit (ISO 646)
- con il bit 8, si hanno altri 128 caratteri, scelti in base alla lingua a cui si vuole offrire il supporto, dando luogo agli standard ISO 8859
  - 8859-1 (ISO Latin-1) è usato per la maggior parte delle lingue europee
  - 8859-15 cirillico, 8859-6 arabo, 8859-7 greco, 8859-8 ebraico
- utilizzando 16 bit si possono rappresentare insiemi di caratteri fonetici e ideogrammi che rappresentano intere parole (indispensabile per cinese e giapponese); si ottiene il codice UNICODE a 16 bit; i primi 128 caratteri sono identici all'ISO 646 e i primi 256 sono gli stessi dell'ISO 8859-1
- utilizzando 32 bit, si ottiene lo standard ISO 10646, che ha l'obiettivo di raccogliere tutti i simboli utilizzati da tutte le lingue del mondo inclusi quelli matematici, valutari ecc.
- C++ usa 8 bits ~ 256 caratteri differenti
- Java usa 16 bit ~ 65,535 caratteri differenti



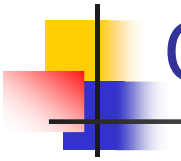
Caratteristiche di Java

- Semplice e familiare
- Object-oriented
- Robusto e sicuro
- Portabile
- Ad alte prestazioni
- Interpretato
- Dinamico
- Supporta i thread
- Supporto per networking



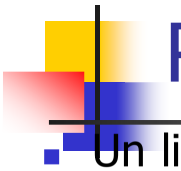
Semplice e familiare

- Java è **semplice** perché intuitivo, utilizza nomi estesi e notazioni puntate, ad esempio *System.out.println("prova")*
- Java è **familiare** perché la sintassi è molto simile al C e C++



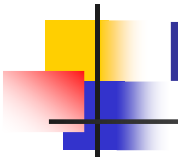
Object Oriented

- Java è un linguaggio Object Oriented
- Ogni elemento risiede all'interno di una classe
  - Il metodo Main e tutti gli altri metodi devono essere definiti all'interno di una classe
- Supporto per ereditarietà, polimorfismo, message passing
- L'uso di Java come linguaggio di programmazione non implica che che il programma sia object-oriented
- Le librerie standard di Java non sempre sono dei buoni esempi di programmazione OO. Esempi:
  - l'attributo length negli array
  - L'attributo out in System
- Compromesso fra considerazioni di pratica e efficienza e reale buon progetto OO



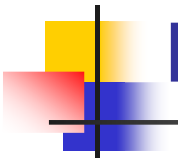
Robustezza e sicurezza

- Un linguaggio si dice ***robusto*** quando anche in situazioni di potenziale errore (*error-prone situations*) si riesce a mantenere l'operatività.
- ***Java è robusto*** in quanto vengono effettuati sia dei controlli durante la compilazione (*compile-time checking*) che durante l'esecuzione (*run-time checking*), ad esempio effettua l'array bounds checking ed il null pointer checking;
- Una delle caratteristiche che rendono Java robusto e che merita attenzione è la ***gestione delle eccezioni***, ovvero delle possibili situazioni di errore. Gestirle consente di evitare il *crash* del programma, e di personalizzare il comportamento del medesimo a fronte di situazioni inaspettate (appunto, *eccezioni*).



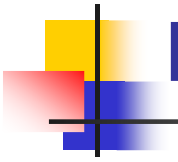
Robustezza e sicurezza

- Un linguaggio si dice sicuro quando fornisce meccanismi per evitare l'esecuzione di codice potenzialmente non corretto o intenzionalmente pericoloso (malicious). Nel caso di Java:
- la gestione della memoria non è delegata al compilatore (come in C o C++), ma rinviata al run-time;
- i puntatori non esistono (accesso diretto impossibile);
- tutto deve essere esplicito (ad esempio il casting), niente avviene "nell'ombra";
- tutte le classi locali sono poste in un name space distinto da quello per le classi scaricate dalla rete; le classi importate non possono "spiare" quelle locali;
- Java non si fida del codice proveniente dalla rete, per il quale viene eseguito la bytecode verification



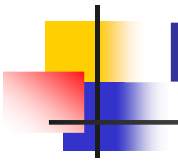
Robustezza e sicurezza

- La ***bytecode verification*** verifica che :
  - i puntatori siano usati legalmente;
  - siano rispettate le regole di accesso ai membri di una classe;
  - gli oggetti non siano usati in maniera maliziosa;
  - non ci siano stack overflow o underflow;
  - i tipi di tutti i parametri delle istruzioni bytecode siano conosciuti e corretti;
  - gli accessi ai campi degli oggetti siano legali



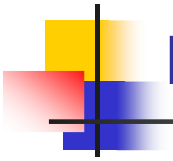
Robustezza e sicurezza

- Inizialmente Java distingueva il codice locale (trusted) e quello remoto (untrusted), modello sicuro ma limitativo
- successivamente, si è concesso al codice importato di poter essere trusted, ciò purchè esso fosse firmato (signed), modello migliore ma ancora troppo binario
- un modello *fine grained* è stato poi introdotto; in esso, per ogni codice remoto si può specificare una politica di comportamento in un apposito file



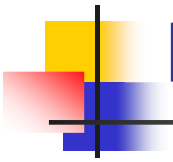
Robustezza e sicurezza

- Java fornisce un insieme completo di funzionalità per la security:
  - ***cryptography*** management (*JCE*), simmetrico (DES) e non (RSA);
  - ***key*** management (key database);
  - ***digital sign*** (firma con chiave privata, decodifica con chiave pubblica (esportata come certificato - ***certificate*** management);
  - ***message digest*** (***hashing***), per il controllo dell'integrità;
  - ***authentication*** e ***access control*** (*JAAS*);
  - secure network communication con ***SSL*** (*JSSE*);
- Java consente di governare tutti gli accessi alle risorse locali ed alla rete che il codice tenta di effettuare; questo è possibile definendo un opportuno ***security manager*** (ad esempio, esso viene automaticamente creato per il codice scaricato dalla rete, ritenendo quest'ultimo *untrusted*).



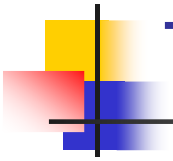
Portabilità, prestazioni, interpretato

- La portabilità è garantita con la compilazione del sorgente in bytecode, interpretato dovunque tramite la JVM
- quasi tutti i controlli sono fatti in compilazione, così la JVM a run time può offrire prestazioni adeguate nonostante l'ambiente interpretato



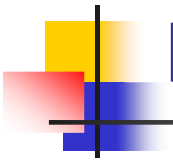
Dinamicità

- le classi sono caricate in maniera trasparente dal class loader quando sono richieste, e possono anche essere scaricate dalla rete.
- Ogni volta che ad una certa classe vengono aggiunti metodi o variabili, tutte le classi che ad essa fanno riferimento (ad esempio, le sue sottoclassi), richiedono una ricompilazione (**fragile superclass problem**).
- Per evitare questo problema, il compilatore Java lascia i riferimenti sotto forma simbolica sino al momento dell'esecuzione, quando avverrà la sostituzione (nel codice che ne fa uso) della forma simbolica con quella reale. Il codice che ne fa uso non richiede quindi la ricompilazione.



Threading

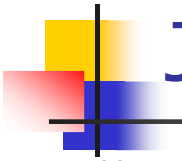
- Java gestisce il multithreading
- un thread è un processo "leggero"
- Java tratta i threads come Abstract Data Type



Networking

Java offre un meccanismo per l'ambiente distribuito, ossia il ***Remote method invocation*** (***RMI***), tramite cui si possono creare oggetti i cui metodi possono essere invocati da altre JVM, anche situate su macchine remote (architettura ***client-server***, gestione simile all'***RPC***)

Il supporto al networking viene fornito tramite un insieme di librerie che implementano i concetti di ***socket***, ***URL***, ***stream connection***, e soprattutto Java offre il supporto per gli ***applet*** e ***servlet***



Java Programming

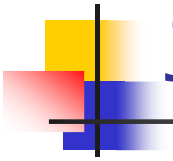
- Un programma Java è un insieme di classi e oggetti. Le classi sono componenti statici, che esistono già all'inizio del programma. Gli oggetti sono invece componenti dinamici, che vengono creati al momento del bisogno, durante l'esecuzione.
- Il più semplice Programma Java è costituito da una singola classe operante come singolo componente software. Essa avrà quindi la sola parte statica. Come minimo, tale parte dovrà definire una singola funzione (statica): il main
- Il main in Java è una funzione pubblica con la seguente interfaccia obbligatoria:

```
public static void main(String args[]){
...
}
```

- Deve essere dichiarato public, static, void
- Non può avere valore di ritorno (è void)
- Deve sempre prevedere gli argomenti dalla linea di comando, anche se non vengono usati, sotto forma di array di String

Java Programming

- Prima differenza rispetto al C:
  - il main deve sempre dichiarare l'array di stringhe args, anche se non lo usa (ovviamente può anche non chiamarlo args...)
  - il main non è più una funzione a sé stante: è definito dentro una classe pubblica, ed è a sua volta pubblico; In effetti, in Java non esiste nulla che non sia definito dentro a una qualche classe.
- Convenzioni:
  - il nome di una classe ha sempre l'iniziale maiuscola (es. Esempio)
  - se il nome è composto di più parole concatenate, ognuna ha l'iniziale maiuscola (es. DispositivoCheConta)
  - non si usano trattini di sottolineatura
  - i nomi dei singoli campi (dati e funzioni) iniziano invece per minuscola

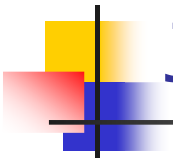


Java Programming

Un programma costituito da una singola classe EsempioBase che definisce il main:

```
public class EsempioBase {
  public static void main(String args[]) {
  int x = 3, y = 4;
  int z = x + y;
  }
}
```

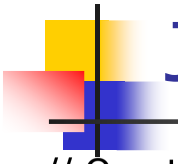
- Una classe pubblica deve risiedere in un file .java che ha lo stesso nome (case sensitive)
- All'interno di un file è ammessa al più una classe public



Java Programming

Per creare ed eseguire un programma Java:

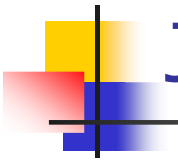
- Creare un file testo con estensione *.java* in cui inserire il codice sorgente. Per esempio il codice sorgente di un programma può risiedere nel file *Esempio.java*
- Scrivere il codice sorgente.
- Compilare il codice sorgente in bytecode.
  $# javac Esempio.java
- Se si verificano "fatal errors" durante la compilazione il processo termina e vengono generati messaggi d'errore
- Se si verificano "non fatal errors" durante la compilazione il processo continua e vengono generati messaggi di warnings.
- Viene generato il file *Esempio.class*
- A questo punto, per eseguire il programma, occorre invocare l'interprete (JVM) tramite il comando java, seguito dal nome del file .class in bytecode senza l'estensione .class
  $# java Esempio



Java Programming

```
// Questo programma stampa Ciao sullo standard output.
public class Esempio {
  public static void main( String[] a ){
   System.out.println( "Ciao" );
  }
}
```

- Questo programma e una "Java application" piuttosto che un applet, o una servlet, che sono altri tipi di programmi.
- Una "Java application" può essere considerato il tipo di programma più generale; una applicazione deve avere un metodo "public" e "static" chiamato "main", che accetta un array di stringhe:
- La sintassi *String[] a* indica che a è un array. In questo caso si tratta di un array di Strings, che sono gli argomenti passati al programma
- Nell'istruzione *System.out.println("Ciao!")* System è una classe Java standard che rappresenta il sistema su cui il programma viene eseguito. Out è un attributo field della classe System di tipo PrintStream e println è un metodo di PrintStream.



Java Programming

- Stile a " invio di messaggi":
  - non più chiamate di funzioni con parametri che rappresentano i dati su cui operare (ma che siano quelli lo sa solo l'utente…)…
  - ...ma componenti su cui vengono invocate operazioni a essi pertinenti
- Notazione puntata:

  System.out.println("Hello!");
- Il messaggio println("Hello!") è inviato all'oggetto out che è un membro (statico) della classe predefinita System