Java™

Dott. Antonio Giovanni Lezzi

JAVA

- Fondamenti
- Programmazione Object Oriented
- Java e OOP nel dettaglio
- Java Avanzato
- Network
- Database
- Web

Dott. Antonio Giovanni Lezzi

Afol Milano

FONDAMENTI

- Introduzione
- Il mondo Java: caratteristiche e vantaggi
- Ambiente di sviluppo JVM,JDK...
- Variabili
- Tipi di dati primitivi, operatori matematici, di confronto e booleani
- Istruzioni condizionali e cicli iterativi
- Array e Stringhe
- Input / Output

INTRODUZIONE

- **Programmazione non strutturata:** il programma è costituito da **un unico blocco di codice detto "main"** dentro il quale vengono manipolati i dati in maniera totalmente sequenziale. Tutti i dati sono **variabili di tipo globale**, ovvero visibili da ogni parte del programma ed allocate in memoria per tutto il tempo che il programma stesso rimane in esecuzione.

- **Programmazione Procedurale**: Il concetto base qui è di raggruppare i pezzi di programma ripetuti in porzioni di codice utilizzabili e richiamabili ogni volta che se ne presenti l'esigenza. Ogni procedura può essere vista come un sottoprogramma che svolge una ben determinata **funzione** visibile e richiamabile dal resto del codice.

- **Programmazione ad oggetti**: il paradigma è basato sul fatto che esiste una serie di oggetti che interagiscono vicendevolmente, scambiandosi messaggi ma mantenendo ognuno il proprio stato ed i propri dati.

STORIA

- Tra gli anni '70 e la fine degli anni '80 l'hardware aveva subito una delle sue più grandi rivoluzioni (prezzi caduti a picco e performance aumentate)
- In quegli anni il **C** (Dennis Ritchie, 1972) era nel pieno della sua diffusione ma la sua struttura ed il suo approccio "low level" (puntatori, allocazione statica della memoria, etc.) lo rendevano impegnativo e di difficile adozione per progetti di sviluppo complessi e team grandi.
- Alla fine degli anni '70 anche un nuovo player era entrato in campo: il **C++** (Bjarne Stroustrup, 1979) introduceva i principi della Programmazione Orientata agli Oggetti, alla fine degli anni ottanta era il punto di riferimento per progetti ambiziosi.
- Il piano del Green Team era dunque quello di realizzare un linguaggio che, come il C++, potesse trarre vantaggio dal nuovo paradigma di programmazione ad oggetti ma che non fosse low-level

STORIA

- Dopo 2 anni di lavoro del Green Team nacque un linguaggio chiamato *Oak* che rischiava di finire cancellato se Sun ed il Green Team stesso non avessero avuto la felice idea di catturare con il loro linguaggio la vera "next-wave" degli anni 90: Internet.

- Nel giro di pochi mesi riuscirono a sviluppare un web browser interamente in Java con il quale poterono mostrare l'idea che avrebbe legato il nuovo linguaggio alla evoluzione di Internet: le **applets**, piccole applicazioni che attraverso un browser potevano essere distribuite ed eseguite come mai prima.

- Il 23 Maggio 1995, John Gage (direttore Sun Microsystems) con a Marc Andreessen (co-fondatore di Netscape), annunciarono alla SunWorld che Netscape Navigator avrebbe integrato una nuova tecnologia: **Java**.

FONDAMENTI

- Introduzione
- Il mondo Java: caratteristiche e vantaggi
- Ambiente di sviluppo JVM,JDK...
- Variabili
- Tipi di dati primitivi, operatori matematici, di confronto e booleani
- Istruzioni condizionali e cicli iterativi
- Array e Stringhe
- Input / Output

COSA È

- Il linguaggio Java ha l'intento di occuparsi automaticamente della gran parte dei dettagli come la gestione della memoria e che sia facilmente portato su processori e dispositivi diversi.

JAVA

- Java è stato creato per soddisfare quattro scopi:
- essere orientato agli oggetti;
- essere indipendente dalla piattaforma;
- contenere strumenti e librerie per il networking;
- essere progettato per eseguire codice da sorgenti remote in modo sicuro.

JAVA

- L'indipendenza dalla piattaforma è ottenuta grazie all'uso di uno strato software chiamato Java Virtual Machine (JVM) che traduce le istruzioni dei codici binari indipendenti dalla piattaforma, generati dal compilatore Java, in istruzioni eseguibili dalla macchina locale.

- Java ha quindi unito i vantaggi di un linguaggio compilato a quelli di uno interpretato. Il sorgente Java viene infatti compilato in un codice intermedio tra il linguaggio macchina e il codice sorgente: il bytecode. Il file scritto in Bytecode sarà effettivamente il programma Java e ogni istruzione scritta in bytecode viene interpretata dalla JVM a runtime.

JAVA NEI SISTEMI CLIENT

- Vantaggi
  - Indipendenza dalla piattaforma di sviluppo e di utilizzo
  - Ciclo di sviluppo rapido
  - Distribuzione immediata, a richiesta, tramite web
- Svantaggi
  - Prestazioni limitate rispetto ad applicazioni native
  - Ambienti di esecuzione in forte evoluzione

JAVA NEI SISTEMI SERVER

- Varie tipologie di utilizzo:
  - Servlet / Java Server Pages / Java Server Faces
  - EnterpriseJavaBean
  - Application server
- È il contesto in cui Java sta ottenendo il massimo successo

JAVA NEI SISTEMI EMBEDDED

- Segmento di mercato in forte crescita:
  - Milioni di PC connessi ad Internet
  - Centinaia di milioni di telefoni cellulari
  - Miliardi di elettrodomestici
- Vengono usate librerie specifiche
  - JavaCard, Java Micro Edition, …

L'EVOLUZIONE NEI LINGUAGGI

- C, C++, e Java: Come Java è collegato a C e C++
- Vantaggi di Java: Scrivere buon codice è più semplice di C e C++
- Dove è la potenza di Java: Librerie (packages)
- Per cosa è buono: internet e ambienti in cui si vuole astrarsi dall'hardware

C/C++ E JAVA

- Linguaggio C disegnato negli anni '70 per programmare sistemi operativi come Unix
  - Alto Livello: comparato con C++ o Java (consente di implementare programmi con poche linee di codice)
  - Basso Livello: linguaggio comparato con linguaggio Assembly (facile manipolazione di HW)

- Punti di forza
  - Non è restrittivo, specialmente in relazione alla conversione di tipo di dati
  - Più efficiente di C++ e Java

- Debolezze
  - Non è Object oriented: non supporta l'astrazione e incapsulamenti, difficile gestire progetti di grande dimensioni
  - Lo sviluppo di un progetto deve necessariamente cominciare da Zero
  - È facile commettere errori (puntatori, a=b oppure a==b)

C/C++ E JAVA

- Linguaggio C++ progettato negli anni '80 per completare il sotto insieme di C
- Cambiamenti: Supporta la programmazione Object-Oriented, …

- Punti di forza:
  - Consente una astrazione dei dati e incapsulamenti
  - È più semplice gestire progetti di grandi dimensioni
  - È molto più esteso come linguaggio e contiene librerie standard integrate nel linguaggio

C/C++ E JAVA

- Il linguaggio Java è un linguaggio degli anni '90
- L'idea di Java consiste di avere una sintassi e semantica uguale simile al C
- Aggiungere alcune caratteristiche prese dal C++: Oggetti, Eccezioni
- Mette da parte la complessità, elementi non necessari e rende il codice più robusto
    - Gosling: "Java nella nostra esperienza porta più svantaggi che non benefici perché omette caaratteristiche, altre pensate male e confuse rispetto al linguaggio C."
- Aggiunge delle facilitazioni non presenti in in C or C++
    - Garbage collection, concorrenza, runtime error checking, object serialization, interface, inner classes, threads
- Punti di forza: portabilità e sicurezza

FONDAMENTI

- Introduzione
- Il mondo Java: caratteristiche e vantaggi
- Ambiente di sviluppo JVM,JDK...
- Variabili
- Tipi di dati primitivi, operatori matematici, di confronto e booleani
- Istruzioni condizionali e cicli iterativi
- Array e Stringhe
- Input / Output

MACCHINA VIRTUALE

Con la Macchina Virtuale (**JVM**) si fa in modo che i programmi non fossero compilati in codice macchina (nativo) ma in una sorta di codice "intermedio" (chiamato **bytecode**), quindi non è destinato ad essere eseguito direttamente dall'hardware ma che deve essere, a sua volta, interpretato da un secondo programma.

Quindi lo stesso codice può essere eseguito su più piattaforme semplicemente trasferendo il bytecode (non il sorgente) purché sia disponibile una JVM (concetto chiamato WORA, Scrivi una volta ed esegui ovunque).

JAVA DEVELOPMENT KIT (JDK)

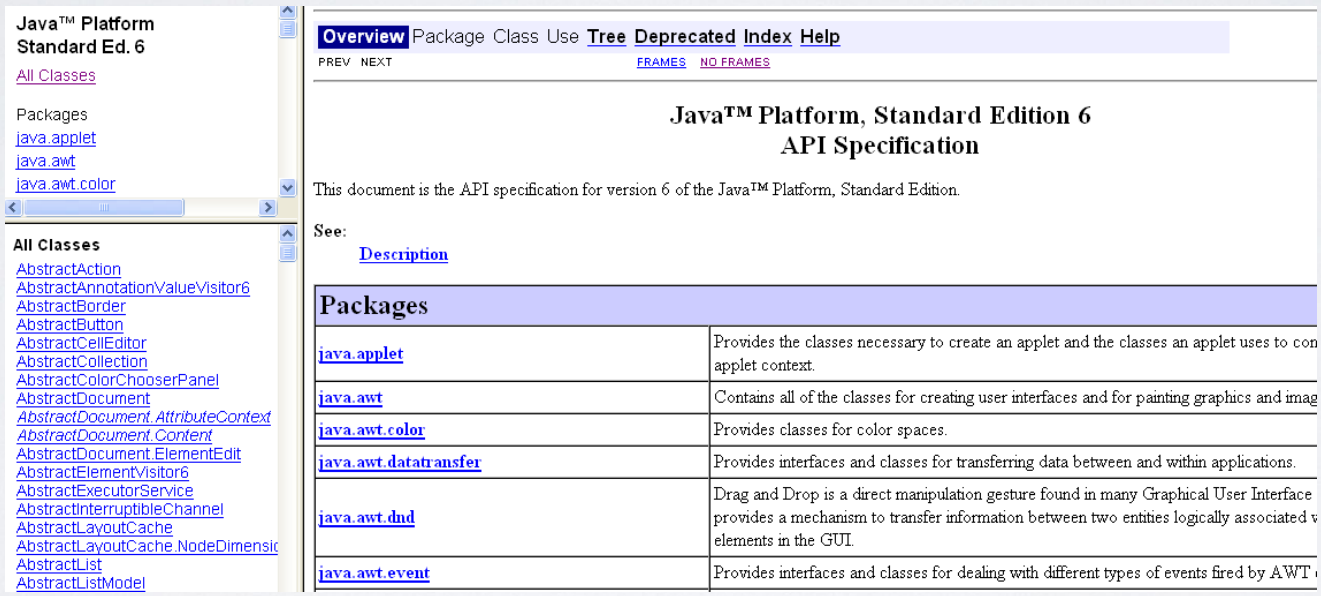
- Il codice sorgente di un programma Java può essere scritto con un qualsiasi editor di testi, e come quasi tutti i linguaggi di alto livello è indipendente dalla macchina e dal sistema operativo.
- Deve essere poi tradotto (con il compilatore javac) in un codice ottimizzato chiamato bytecode, che come il codice Java è indipendente dalla macchina e dal sistema operativo.
- javac e altri strumenti per lo sviluppatore (jar, javadoc, ecc.) fanno parte del JDK.

DOWNLOAD E INSTALLAZIONE

- Andare alla pagina: http://java.sun.com/javase/downloads/ e scaricare JDK 8 Update 45
- Alla fine dell'installazione si avranno Java SDK e JRE in C:\Programmi\Java\
- Potrebbe essere necessario definire la variabile d'ambiente JAVA_HOME e aggiornare la variabile d'ambiente Path.

DOCUMENTAZIONE

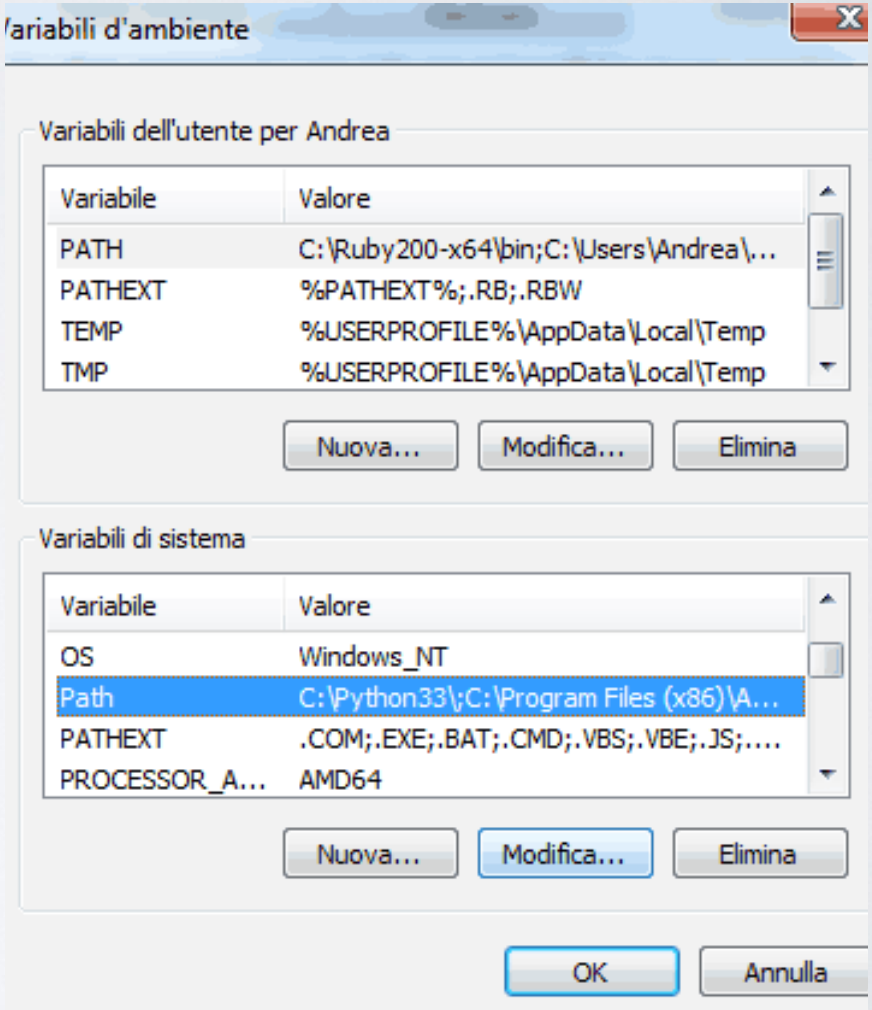
La documentazione online sulla Java Class Library si trova al seguente indirizzo: http://java.sun.com/javase/6/docs/api/



C'è anche una versione scaricabile per consultazione offline!
http://java.sun.com/javase/downloads/ → Java SE 6 Documentation

IMPOSTAZIONI

- Risorse del computer > Visualizza informazioni sul sistema > Avanzate > Variabili d'ambiente
- Aggiungere se non esistente:
- JAVA_HOME = C:\Programmi \Java\jdk1.8.0
- In Path aggiungere %JAVA_HOME%\bin; %JAVA_HOME%\lib;

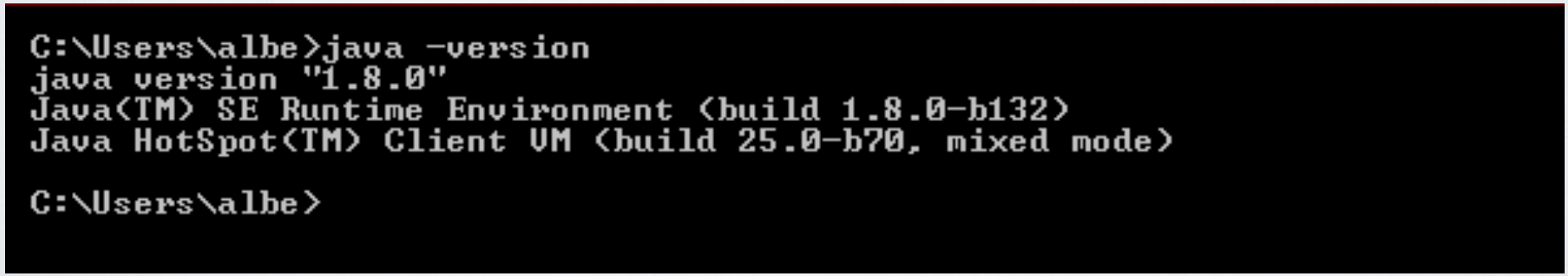


VERIFICA

Per verificare che l'istallazione abbia avuto successo potete aprire il terminale e scrivere il comando:

java -version

Il risultato dovrebbe essere simile a quello in figura:



AMBIENTE DI SVILUPPO

Ne esistono diversi tipi, quello maggiormente usato è Eclipse

Con una comoda interfaccia grafica, permette di scrivere codice Java, e poi compilarlo ed eseguirlo con un semplice click del mouse.

www.eclipse.org

L'installazione è semplice: basta scaricarlo e scompattarlo.

API JAVA

- Application Programming Interface
  - Insieme di meccanismi per interagire con il sistema ospitante
  - Progettati interamente ad oggetti
- Offrono
  - Funzioni di libreria
  - Interfaccia verso il sistema operativo

PACKAGE PRINCIPALI

- java.awt
  - Abstract Windowing Toolkit
  - Classi per creare interfacce utente di tipo grafico
- java.io
  - Input/Output
  - Classi per accedere a a flussi di dati, file e altri meccanismi di comunicazione

PACKAGE PRINCIPALI

- java.math
  - Estensioni matematiche
  - Classi per modellare numeri interi e reali con precisione illimitata
- java.net
  - Meccanismi di accesso alla rete
  - Socket, URL, connessioni HTTP, …

PACKAGE PRINCIPALI

- java.text: Trattamento multiculturale di numeri, date, testo
- java.util: Insieme variegato di classi ad uso generale
- java.lang: Contiene le classi fondamentali del linguaggio
  - Fornisce le classi fondamentali per la programmazione Java ed è importato automaticamente dal compilatore in tutti i programmi
  - Contiene – tra le altre – le classi Object, Throwable, String

JAVA RUNTIME ENVIRONMENT (JRE)

- Per eseguire programmi Java, è necessario installare JRE, che comprende:

1. Java Virtual Machine (JVM) Macchina software che esegue il bytecode.
2. Java Class Library Collezione di classi già pronte (e compilate), che offrono strutture dati e funzioni di base, molto utili al programmatore.

- JDK include JRE ed è scaricabile da: http://java.sun.com
- Java SE (Standard Edition) v1.8
- Java EE (Enterprise Edition)
- Java ME (Mobile Edition) diverse versioni a seconda del tipo di device

PERFORMANCE

Java è sempre stato etichettato come un linguaggio a scarse performance. Negli anni, il linguaggio si è trasformato portando migliorie da una versione all'altra tra il 50 e il 75%

Mai potrà attestarsi a livelli di un linguaggio a bassissimo livello e ovviamente non sceglieremo mai per sviluppare applicazioni real-time.

Alcune delle attività nascoste del linguaggio Java che gli consentono di avere prestazioni eccellenti (il programmatore può non conoscerle):

- Compilazione Just In Time (JIT)
- Hot Spot;
- Garbage Collection

JIT

la compilazione Just In Time si effettua un lavoro di ottimizzazione per eseguire un pezzo di codice Java compilandolo in codice nativo solo al momento della reale necessità.

HOTSPOT

Java Hot Spot introdotto in Java 2 (o 1.2), il cui compito è di individuare un metodo di una classe utilizzato spesso, lo compila e lo lascia già compilato in maniera che il prossimo utilizzo non necessita compilazione.

Algoritmi adattativi di intelligenza artificiale, sono in grado di prevedere le attività più frequenti nel ciclo di vita di una applicazione

Le ultime versioni portano delle migliorie in questi algoritmi adattativi, prendendo a vantaggio anche l'utilizzo del Class Data Sharing.

Il **Class Data Sharing** viene introdotto in Java 5 e consente di migliorare la velocità all'avvio di una applicazione riutilizzando eventuali package di librerie già caricati in memoria da altre JVM.

GESTIONE MEMORIA

Il primo linguaggio basato su un sistema di gestione della memoria chiamato **gargbage collection** in cui la memoria viene automaticamente assegnata e rilasciata a seconda delle esigenze del programma.

Con altri linguaggi non è così facile gestire la memoria

Possiamo pensare al garbage collector come a un processo asincrono che tiene una mappa di oggetti e referenze alle istanze utilizzate e che cancella lo spazio di memoria fisico quando uno di questi oggetti non ha referenze da parte di nessuno.

In Java 1.5 è stato introdotto il concetto di "young generation" e "old generation". Nella maggior parte delle istanze di una classe sono temporanee: create, utilizzate e scartate, con un ciclo di vita abbastanza giovane. Una piccola minoranza di istanze (circa il 10% secondo studi) ha un ciclo di vita maggiore, quasi quello dell'applicazione.

COMPILAZIONE

- Tutti i sorgenti sono messi nella stessa directory ed hanno estensione '.java'.
- Se un file si chiama X.java, allora contiene una classe pubblica che si chiama X.
- Per compilare, si usa il comando:

  javac <nome_del_file_compreso_.java>

- Partendo da X.java, viene generato il bytecode X.class.
- Per eseguire il bytecode, si usa il comando:

  java <nome_del_file_senza_estensione>

- Se il bytecode è X.class, si scrive java X

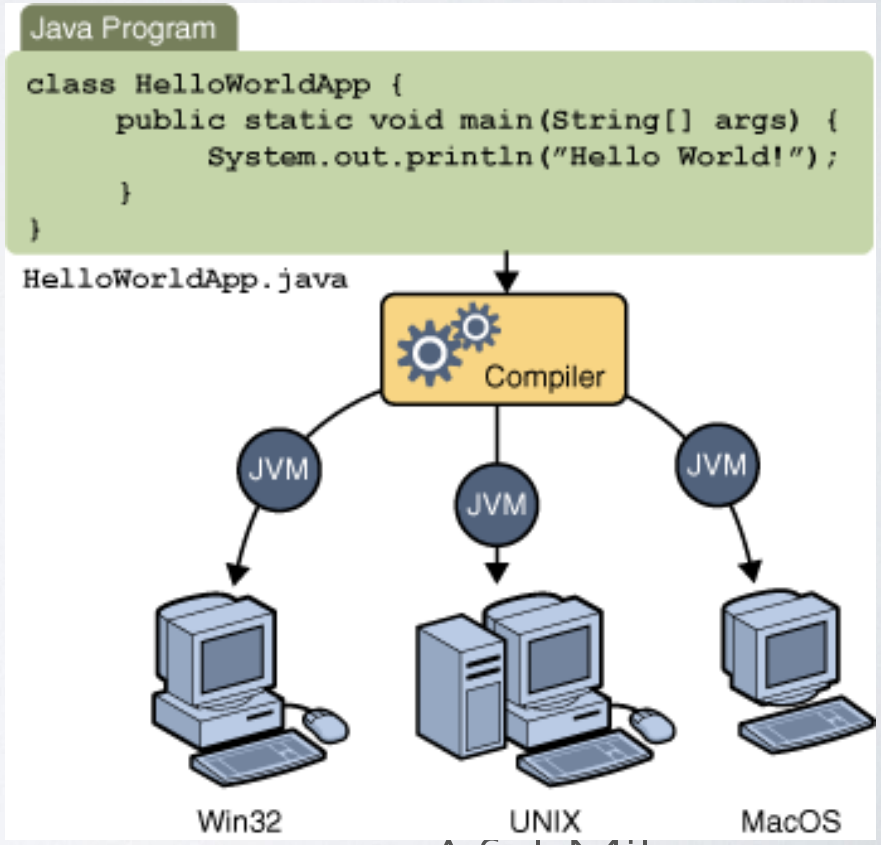
CICLO DI VITA (PROGRAMMA JAVA)

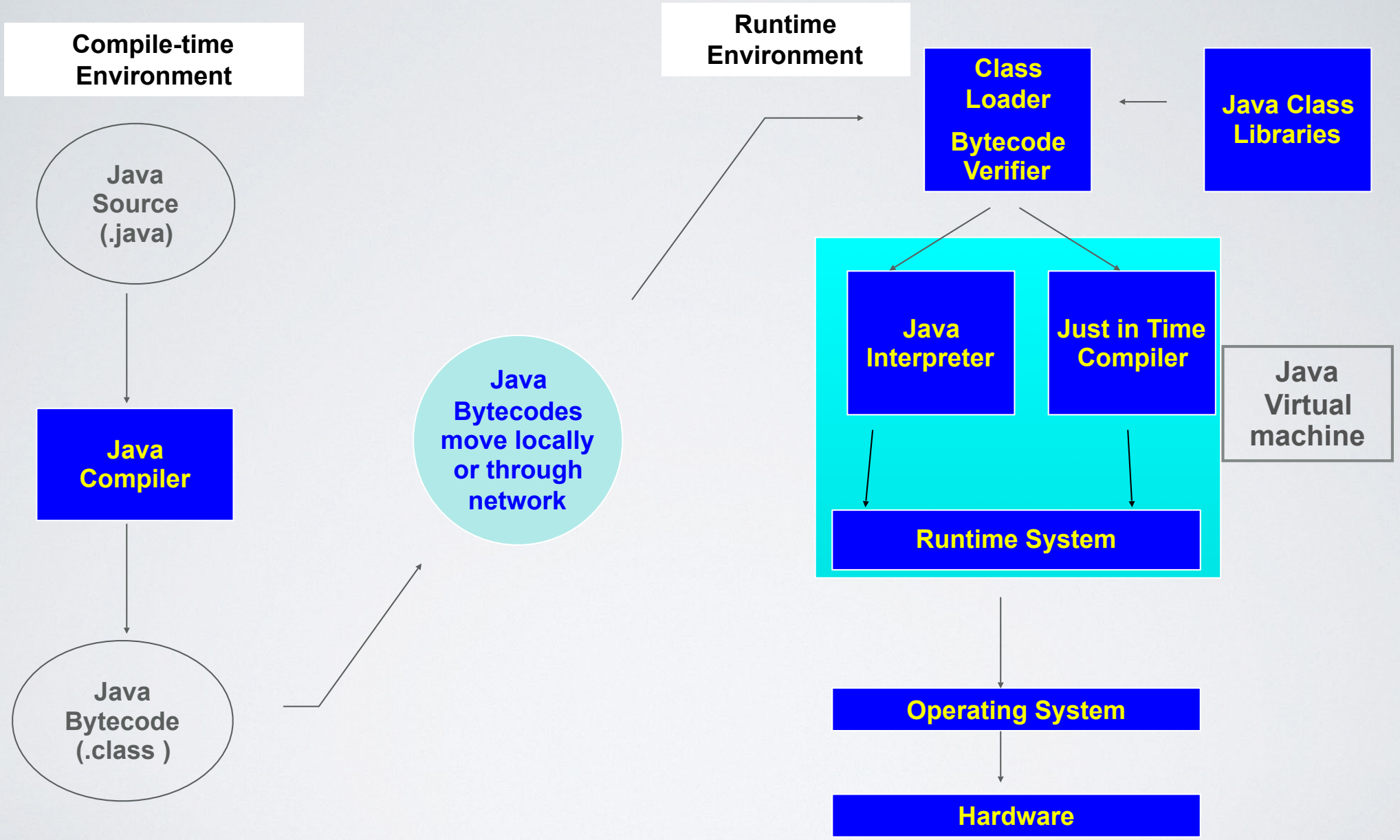
La JVM non interpreta il bytecode (sarebbe poco efficiente), ma utilizza un compilatore interno, chiamato JIT (Just In Time), che genera codice eseguibile dalla macchina fisica ogni volta che nuovo bytecode viene caricato.

Il codice generato dal compilatore JIT è più lento di un codice macchina generato da un linguaggio come il C++, ma ha prestazioni ragionevoli.

PORTABILITÀ DI UN PROGRAMMA JAVA

La portabilità è il principale vantaggio dell'utilizzo di Java, significa:

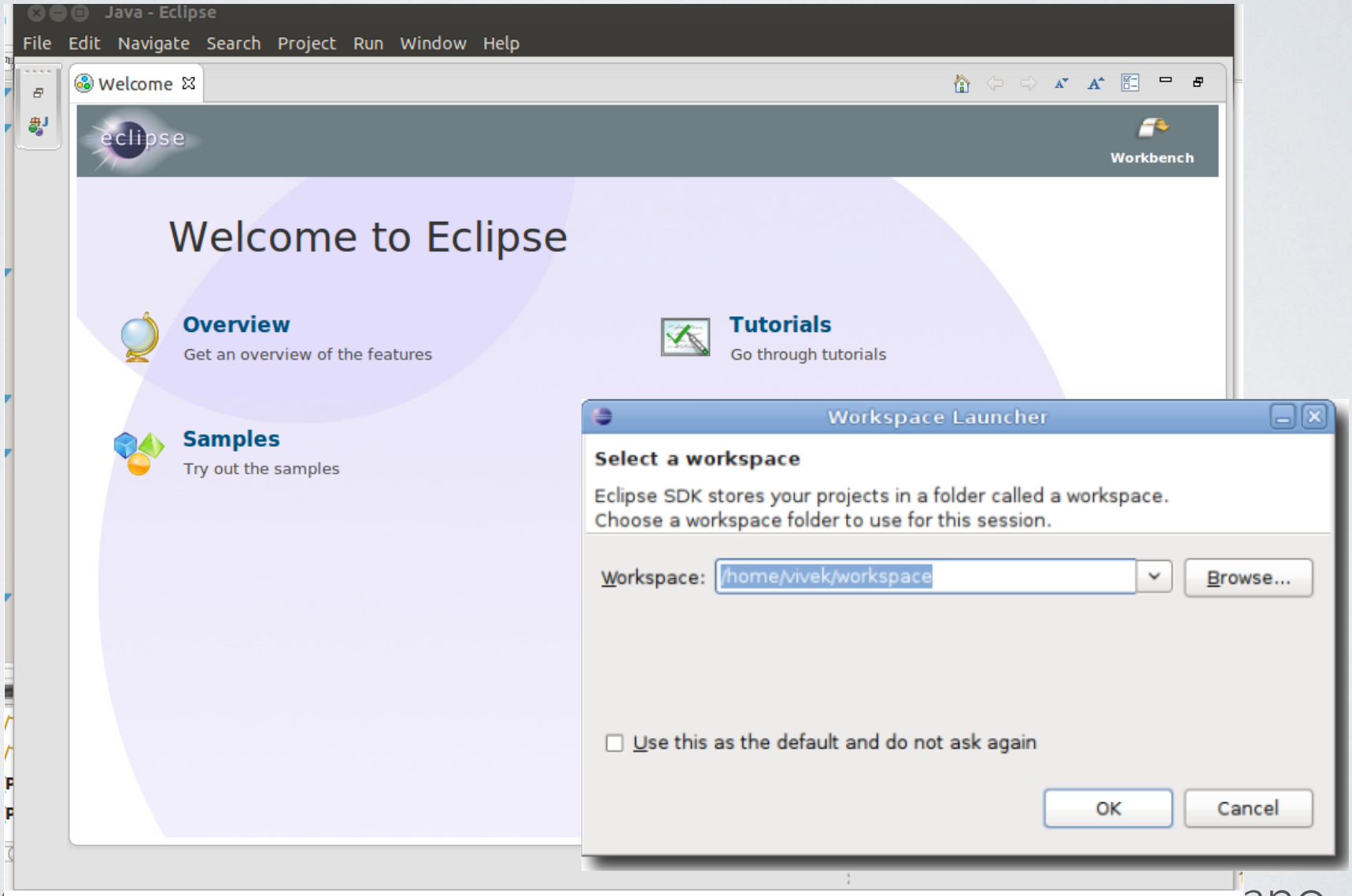



Compile-time Environment
Java Source (.java)
Java Compiler
Java Bytecode (.class)
Java Bytecodes move locally or through network
Runtime Environment
Class Loader
Bytecode Verifier
Java Class Libraries
Java Interpreter
Just in Time Compiler
Java Virtual machine
Runtime System
Operating System
Hardware

ECLIPSE

- Eclipse è un ambiente di sviluppo integrato (Integrated Development Environment – IDE)
- Racchiude in un unico ambiente tutti gli strumenti che servono a un programmatore: Editor, compilatore, debugger, ...
- Eclipse è uno tra i principali IDE disponibili al momento ed è tra i più usati in ambiente aziendale
- Può essere usato per programmare con molti linguaggi diversi (non solo Java)

ECLIPSE: PRIMO AVVIO



ECLIPSE

Ogni area della schermata principale di Eclipse viene detta Vista (View)

La vista centrale ci consente di scrivere il nostro programma

La vista "Package Explorer" (a sinistra) mostra tutti i file creati

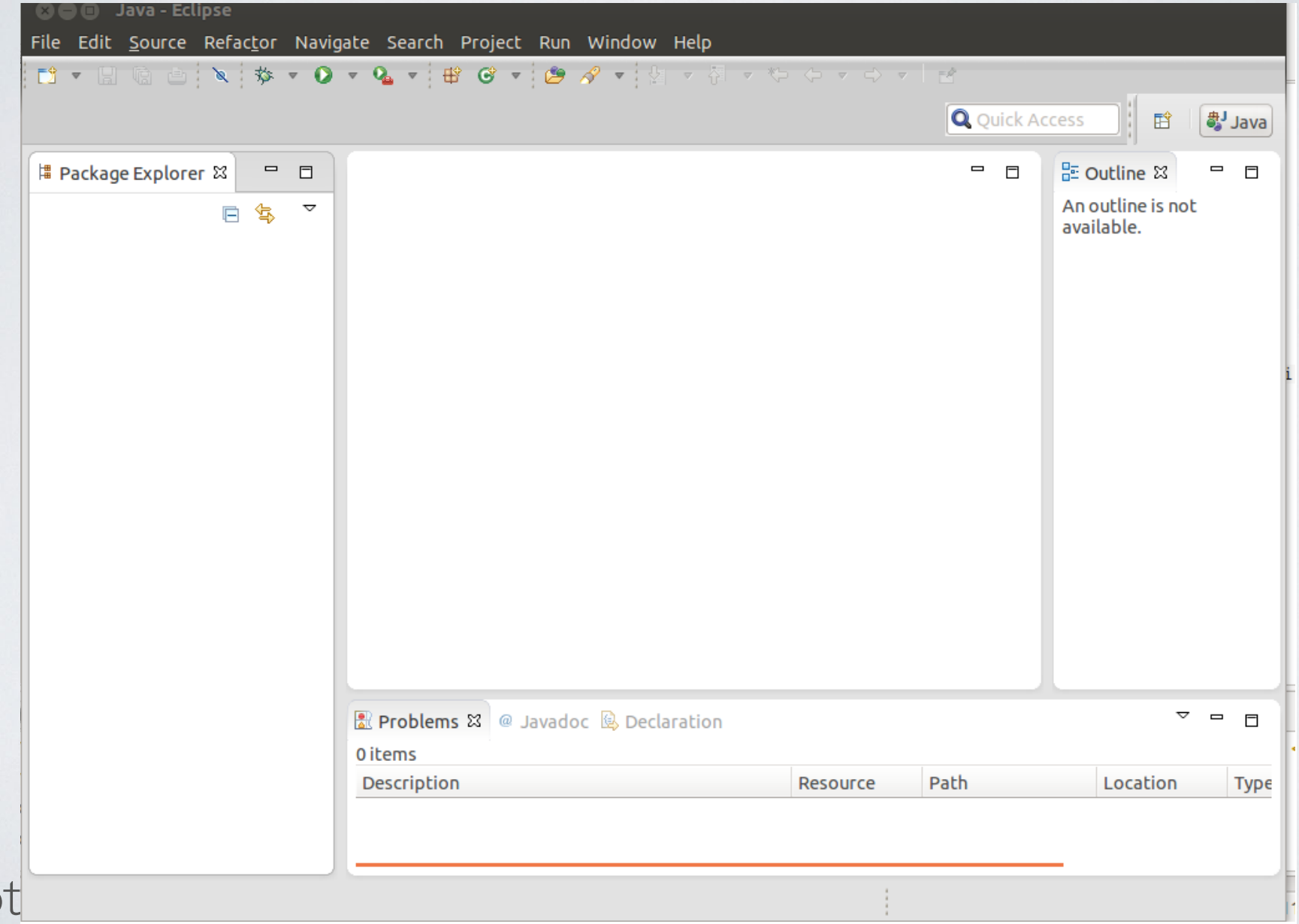
La vista "Outline" (a destra) mostra alcune informazioni sulla classe corrente

La vista "Problems" (in basso) riporta eventuali errori di compilazione

La vista "Console" (non in figura) ci consente di interagire con il programma in esecuzione

Un'insieme di viste prende il nome di Prospettiva (Perspective) Vedremo che oltre alla prospettiva mostrata in figura (Java) ne utilizzeremo un'altra (Debug) che include altre viste

ECLIPSE


Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type

CREARE UN PROGETTO

Per scrivere un programma dobbiamo innanzitutto creare un progetto che è un contenitore di classi Java le quali collegate tra loro.

Quando si realizza un programma complesso di solito si crea un progetto specifico che contiene tutte le sue classi

È possibile creare un progetto per raccogliere tutte le classi realizzate

Per creare un progetto:

File > New > Java project

File > New > Class

Si aprono delle finestre di dialogo

USO DI ECLIPSE

- L'editor di Eclipse ci segnala alcuni **errori** in tempo reale sottolineandoli in rosso Vengono invece sottolineati in giallo situazioni anomale (non necessariamente errori) dette **warning**

- Una volta corretti eventuali errori possiamo **compilare ed eseguire** il programma tramite: Run --> Run oppure cliccando sull'icona a forma di pallina verde con il triangolino bianco nella barra in alto

- Il risultato dell'esecuzione è nella vista **Console** (se non si apre in automatico la si può aprire con Window > Show view > Console)

- Anche l'eventuale input viene richiesto all'utente nella vista Console

DEBUGGER DI ECLIPSE

Il debugger è uno strumento essenziale per ricercare errori nei programmi e consente di far interrompere l'esecuzione del nostro programma in un punto prescelto, una volta interrotto, potremo vedere il valore delle tutte variabili in quel momento

potremo inoltre far procedere il programma un passo alla volta, monitorando la situazione

Per interrompere il programma usiamo i **breakpoint**, nel punto del programma da analizzare e per fare ciò o si clicca con il tasto destro nella barra verticale a sinistra, come risultato comparirà un pallino blu nel punto in cui abbiamo cliccato

Ora facciamo partire il debugger tramite: Run > Debug

oppure, più semplicemente, cliccando sull'icona a forma di scarafaggio nella barra in alto e cambiando prospettiva Debug In alto a destra (nella vista **Variable**) sono visibili tutte le variabili e i loro valori

ALGORITMO

- Un algoritmo è un procedimento che risolve un determinato problema attraverso un numero finito di passi elementari dette istruzioni
- Le prime istruzioni che vedremo di Java sono le istruzioni di dichiarazione, che sono delle dichiarazioni di variabili e assegnare loro un eventuale valore iniziale, in seguito condizioni, cicli e operazioni di input/output

STESURA CODICE: CONVENZIONI

- In Java ogni classe (public class) deve essere contenuta in un file il cui nome sia identico al nome della classe stessa
- In Java le classi possono avere i nomi più disparati e scritte in modo diverso: primo, PRIMO, PrImo123, primo_, _primo sarebbero stati tutti nomi validi
- Esiste una convenzione che prevede che i nomi delle classi inizino con un carattere maiuscolo, continuino con caratteri minuscoli e, se composti da più parole, siano capitalizzate le prime lettere di tutte le componenti (CamelCase).
- Il compilatore javac genererà dei files .class ogni volta che lo utilizzeremo su dei file .java.
- Il file .class contiene il bytecode del nostro programma java che possiamo di eseguire con la macchina virtuale java (JVM): ***java NomeProgramma***

PACKAGE

I package raggruppano le classi(moduli, unità, gruppi, categorie) in modo che il codice sia più organizzato e ogni componente di un package rappresenti una directory.

Dato che il compilatore genera sempre un file .class da ogni file .java e da un singolo file .java possono essere generati anche più file .class, i programmi compilati in java potrebbero esser scomodi da gestire perché composti da intere directory di file.

Con il compilatore e la JVM viene fornito il comando jar (java archiver): lo scopo è di trasformare una intera directory di class file in un unico file

PACKAGE JAVA.LANG

- La prima classe che vediamo è la classe System, la quale interfaccia il nostro programma Java con il sistema operativo sul quale sussiste la virtual machine

```
static PrintStream err
static InputStream in
static PrintStream out
```

- Rappresentano rispettivamente gli i flussi (stream) di informazioni scambiati con la console (standard error, standard input e standard output), ciascuno di essi è un oggetto e sfrutta i metodi della classe relativa. Ad esempio in scrittura invochiamo il metodo println della classe PrintStream, perché out è di tipo PrintStream.

PACKAGE JAVA.LANG

```
System.out.println("Ciao");
System.out.println(true);
System.out.println(10);
System.out.println('C');
```

JAVA.LANG.OBJECT

- Un'altra classe molto importante del package java.lang di Java è la classe Object, questa è la classe da cui ereditano tutte le altre classi di Java: in altre parole ogni altra classe di Java è una estensione della classe java.lang.Object.
- La class Object non contiene alcuna variabile, contiene gli 11 metodi che vengono ereditati dalle altre classi di Java, e che possono essere quindi utilizzati in qualsiasi oggetto. tra questi citiamo:
- **clone**: crea una copia dell'oggetto identica a quella di cui è stato invocato il metodo. In particolare è da notare che il programma Java stesso è un oggetto, quindi si possono creare un numero arbitrario di programmi tutti identici. Affinchè possa essere invocato il metodo clone() di un oggetto, questo deve implementare l'interfaccia Cloneable
- **getClass**: restituisce un oggetto di tipo Class, che rappresenta la classe di appartenenza dell'oggetto di cui è stato invocato il metodo
- **toString**: trasforma l'oggetto in una stringa, questo metodo deve essere implementasto negli oggetti creati dall'utente, ma è molto utile

JAVA.LANG.MATH

- Per terminare la nostra trattazione del java.lang, comunque parziale vista l'immensità del package, esaminiamo brevemente la classe java.lang.Math (diversa dalla classe java.math). Questa classe serve per fare calcoli matematici

- http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

JAVA.UTIL

- Questo package è molto utile, esso mette a disposizione 34 classi e 13 interfacce che implementano alcune tra le strutture dati più comuni, alcune operazioni su date e sul calendario, e altre cose
- Per struttura dati, in informatica si intende una struttura logica atta a contenere un certo numero di dati, nella quale è possibile inserire dati, toglierli, cercarli o ordinarli. Una semplice struttura dati che abbiamo gia visto è l'array, esso contiene un cero numero di dati che possono essere qualsiasi cosa, da byte ad oggetti complessi, su questa si possono fare inserzioni ricerche e cancellazioni, volendo un array può anche essere ordinato.
- Per vedere se in un array grande N è presente un dato X, occorre visitare tutto l'array ed effettuare confronti con ciascun elemento dell'array: un'operazione laboriosa. Esistono strutture dati che eseguono la stessa ricerca impiegando un solo accesso alla struttura
- Java fornisce alcune strutture dati con le relative operazioni e a noi non resta che usarle

COMANDI JAR

jar cf primoprogramma.jar

creerà il file primoprogramma.jar che contiene tutto il nostro albero di directory. Un archivio jar è la versione compressa della directory nel formato zip

Per controllare il contenuto del file .jar possiamo eseguire:

jar tvf primoprogramma.jar

PRIMO PROGRAMMA

```
package com.antonio.programma;

public class Saluta {

    public static void main(String[] args) {

        System.out.println("ciao " + args[0] + " !" );
    }
}
```

Eseguiamo il comando

```
java com.antonio.programma.Saluta "Antonio"
```

PRIMO PROGRAMMA

- Deve esser creata una classe composta da un metodo chiamato main e dichiarato public e static.
- main è precisamente il nome che deve avere il metodo che vogliamo far eseguire per primo alla JVM quando viene lanciata. dovrà avere anche la stessa firma (signature, ovvero gli argomenti ed il valore di ritorno).
- Il fatto che il metodo statico main ritorni int significa che sarà possibile restituire un intero al sistema operativo come risultato dell'esecuzione di un programma
- l'argomento args di tipo String[] (array di stringhe) indica che quel metodo potrà ricevere un numero arbitrario di argomenti di tipo stringa

NOZIONI DI BASE SULLA SCRITTURA DI CODICE JAVA

Ciascuna istruzione deve terminare con ; ad eccezione di quelle per il controllo di flusso e definizione dei metodi

Le { } aprono e chiudono blocchi di istruzioni all'interno di

- metodi
- iterazioni
- decisioni

I commenti vengono ignorati dal compilatore: Singole righe di commento devono essere precedute da // mentre blocchi di righe di commento devono stare tra /* e */

BLOCCO DEL CODICE

- Un blocco di codice in Java è un pezzo di codice che è circondato da una coppia di parentesi graffe: {}
- Le parentesi graffe {} sono usati per contrassegnare l'inizio e la fine di un pezzo di codice come un pezzo di codice che deve svolgere una determinata funzione.
- L'apertura di una parentesi graffa { viene considerata come BEGIN per segnare l'inizio di una sezione di codice. La parentesi graffa chiusa } contrassegna la fine di una sezione di codice. (Ci sono altri linguaggi che utilizzano le graffe per identificare la parte di un blocco di codice come il Pascal e Modula-2, C/C++, ObjectiveC, Swift)

BLOCCO DEL CODICE

- In un programma Java, di solito ci sono diversi blocchi di codice. Blocchi di codice possono essere "annidati" con un blocco di codice interamente all'interno di un altro:

```
public class Hello{
  public static void main(String arg[]){
    System.out.println("Hello.");
  }
}
```

BLOCCO DEL CODICE

- In questo caso, la prima apertura parentesi graffa corrisponde all'ultima chiusura della parentesi (quelli verdi)
- La seconda apertura parentesi graffa , dopo main () , corrisponde al secondo per durare parentesi di chiusura (quelle rosse)

NOZIONI DI BASE SULLA SCRITTURA DI CODICE JAVA

Se per un programma che stiamo scrivendo dobbiamo definire N nuove classi, scriviamo il codice di ciascuna classe in un file .java distinto.

Ciascun file deve cominciare con l'istruzione

```
package nome.del.package;  // nome che identifica gruppo di classi
```

In tutto avremo N+1 file, includendo quello della classe che contiene il metodo main(). Tale classe (detta classe principale) in genere ha il nome del programma. E' la JVM a istanziare la classe principale.

NOZIONI DI BASE SULLA SCRITTURA DI CODICE JAVA

Se una classe A usa altre classi che non stanno nello stesso package di A, all'inizio del file A.java dobbiamo scrivere l'istruzione

```
import package.della.classe.NomeClasse;
```

per ciascuna classe usata da A.

Per rendere più leggibile il sorgente, indentare in modo opportuno con il tasto TAB:

```
for (int i = 0; i < 10; i++)

    accumulator.add(i);
```

FONDAMENTI

- Introduzione
- Il mondo Java: caratteristiche e vantaggi
- Ambiente di sviluppo JVM,JDK...
- Variabili
- Tipi di dati primitivi, operatori matematici, di confronto e booleani
- Istruzioni condizionali e cicli iterativi
- Array e Stringhe
- Input / Output

VARIABILE

Le variabili sono dei valori modificabili, ovvero sono dei nomi che rappresentano un valore di un certo tipo, il valore associato al nome può essere variato

Ad esempio se dico che X è una variabile di tipo intero, e poi dico che X ha valore 10, se scrivo l'espressione 5 + X, è come se avessi scritto l'espressione 5 + 10

Una **variabile** è una coppia composta da:

- un nome simbolico (detto anche **identificatore**)
- e un *indirizzo di memoria* destinato a contenere una determinata quantità, **valore** della variabile.

Il nome simbolico serve ai programmi come riferimento all'indirizzo di memoria al fine di accedere e/o modificarne il valore durante l'esecuzione.

VARIABILE: DICHIARAZIONE

Ogni variabile ha associato anche un tipo (come Integer, String, boolean, etc.) che definisce le caratteristiche che avranno i valori che la variabile potrà assumere.

Il tipo di una variabile può essere uno dei tipi predefiniti (primitivi)

La sintassi per la dichiarazione di una variabile è:

[public|protected|private] [static] [final] Tipo identificatore [= value];

SCOPE DI UNA VARIABILE

È l'area del codice nel quale un identificatore resta associato ad un indirizzo di memoria, quindi l'area di codice entro il quale una variabile mantiene il suo valore.

In Java ogni blocco definisce uno scope e ogni variabile locale ha come scope l'area di codice che inizia dalla definizione della variabile stessa e termina con il blocco corrente.

SCOPE DI UNA VARIABILE

- Nei linguaggi dotati del concetto di blocco, una variabile visibile all'interno di un blocco è in generale visibile anche all'interno di eventuali blocchi annidati

- Le regole fondamentali di visibilità sono in genere modificate dalla regola speciale dello shadowing, secondo cui una variabile locale "nasconde" una eventuale variabile omonima definita nello scope superiore. In altre parole, se in un programma è definita una variabile globale e in un determinato sottoprogramma viene definita una variabile locale omonima, il sottoprogramma in questione perde la visibilità della variabile globale, nascosta da quella locale.

SCOPE DI UNA VARIABILE

- Esistono due ambiti di visibilità (scope) di un identificatore comuni a tutti i linguaggi di programmazione object-oriented:
- **A livello di blocco**. Caratterizza le variabili locali; sono gli identificatori dichiarati in un blocco. Lo scope inizia dalla dichiarazione dell'identificatore e termina con la fine del blocco stesso. Le variabili locali dichiarate all'interno di una funzione, così come i parametri di una funzione, hanno visibilità a livello di blocco. Nel caso di blocchi nidificati, se un identificatore del blocco esterno ha lo stesso nome di quello del blocco interno, l'identificatore del blocco esterno viene occultato fino alla fine del blocco più interno secondo la regola dello shadowing. Il blocco interno vede solamente il proprio identificatore locale.
- **A livello di file**. Caratterizza le variabili globali; un identificatore dichiarato all'esterno di una qualsiasi funzione ha visibilità a livello di file. È noto a tutte le funzioni che si trovano dopo la sua dichiarazione

SCOPE DI UNA VARIABILE

```
<begin blocco 1>
    var x:...                /* qui è visibile x */
    <begin blocco 1.1>
        var y:...            /* qui sono visibili x ed y */
    <end blocco 1.1>
    <begin blocco 1.2>
        var z:...            /* qui sono visibili x e z */
        <begin blocco 1.2.1>
            var y:...        /* qui sono visibili x, z e y */
        <end blocco 1.2.1>
    <end blocco 1.2>
<end blocco 1>
```

TIPOLOGIE DI SCOPE

- Possiamo distinguere due categorie di variabili:
- **Variabili globali**: sono quelle visibili (ovvero referenziabili) da qualunque punto di un programma. L'uso eccessivo di variabili globali è in genere sconsigliato nella pratica dello sviluppo del software, poiché la loro presenza facilita l'occorrere di effetti collaterali che possono portare a errori di programmazione molto difficili da individuare e correggere.
- **Variabili locali**: sono quelle visibili solo all'interno di un determinato sottoprogramma, o, nella maggior parte dei linguaggi strutturati, solo all'interno di un determinato blocco.

TIPOLOGIE DI SCOPE

- Nei linguaggi orientati agli oggetti, le variabili di classe o attributi sono visibili solo all'interno della istanze della classe in cui sono state dichiarate. In particolare, una variabile dichiarata come statica è unica per la classe, e condivisa da ogni sua istanza.

- In altre parole, in ogni istanza della classe tale attributo punta alla medesima area di memoria, pertanto una sua modifica effettuata in una istanza si riflette in tutte le altre istanze della medesima classe.

- Nei linguaggi orientati agli oggetti, gli attributi (non statici) di una classe sono elementi individuali di ogni distinta istanza. In tal caso, in ogni istanza tale attributo è presente localmente all'oggetto stesso, e distinto da quelli presenti in altre istanze della medesima classe

TIPOLOGIE DI SCOPE

- In molti linguaggi, è possibile dichiarare una variabile all'interno di un blocco di una struttura di controllo, o usare un costrutto di blocco solo per delimitare lo scope di una variabile locale, o ancora dichiarare una variabile all'interno di un'espressione

- Queste variabili sono visibili solo all'interno del blocco o dell'espressione in cui sono dichiarate. Tali pratiche facilitano il mantenimento in vita di una variabile solo per il tempo strettamente necessario, in modo da consentire economia di memoria e pulizia nel testo del programma.

ASSEGNAZIONE, INCREMENTO E DECREMENTO

L'operatore = detto operatore di assegnazione (o assegnamento). Alla sua sinistra ci deve essere il nome di una variabile, mentre alla destra ci può essere un singolo valore o espressione.

L'operatore imposta la variabile al valore dato. items = 12;

Forme equivalenti per l'incremento/decremento di 1:

items = items + 1; items++; items += 1;

items = items – 1; items—; items -= 1;

Per incrementare/decrementare di una quantità generica x:

items = items + x; items = items – x; items += x; items -= x;

FONDAMENTI

- Introduzione
- Il mondo Java: caratteristiche e vantaggi
- Ambiente di sviluppo JVM,JDK...
- Variabili
- Tipi di dati primitivi, operatori matematici, di confronto e booleani
- Istruzioni condizionali e cicli iterativi
- Array e Stringhe
- Input / Output

TIPI PRIMITIVE

int – tipo intero da 4 byte

byte – tipo intero da 1 byte

short – tipo intero da 2 byte

long – tipo intero da 8 byte

double – tipo in virgola mobile da 8 byte (±10308 e 15 cifre decimali)

float – tipo in virgola mobile da 4 byte (±1038 e 7 cifre decimali)

char – tipo che rappresenta caratteri Unicode (1 byte)

boolean – tipo per i due valori logici true e false

CAST DI TIPI

- In Java è lecito assegnare un valore intero a una variabile in virgola mobile:

```
int dollars = 100;
double balance = dollars;  // va bene
```

- Non è possibile fare il contrario:

```
double balance = 13.75;
int dollars = balance;  // errore
```

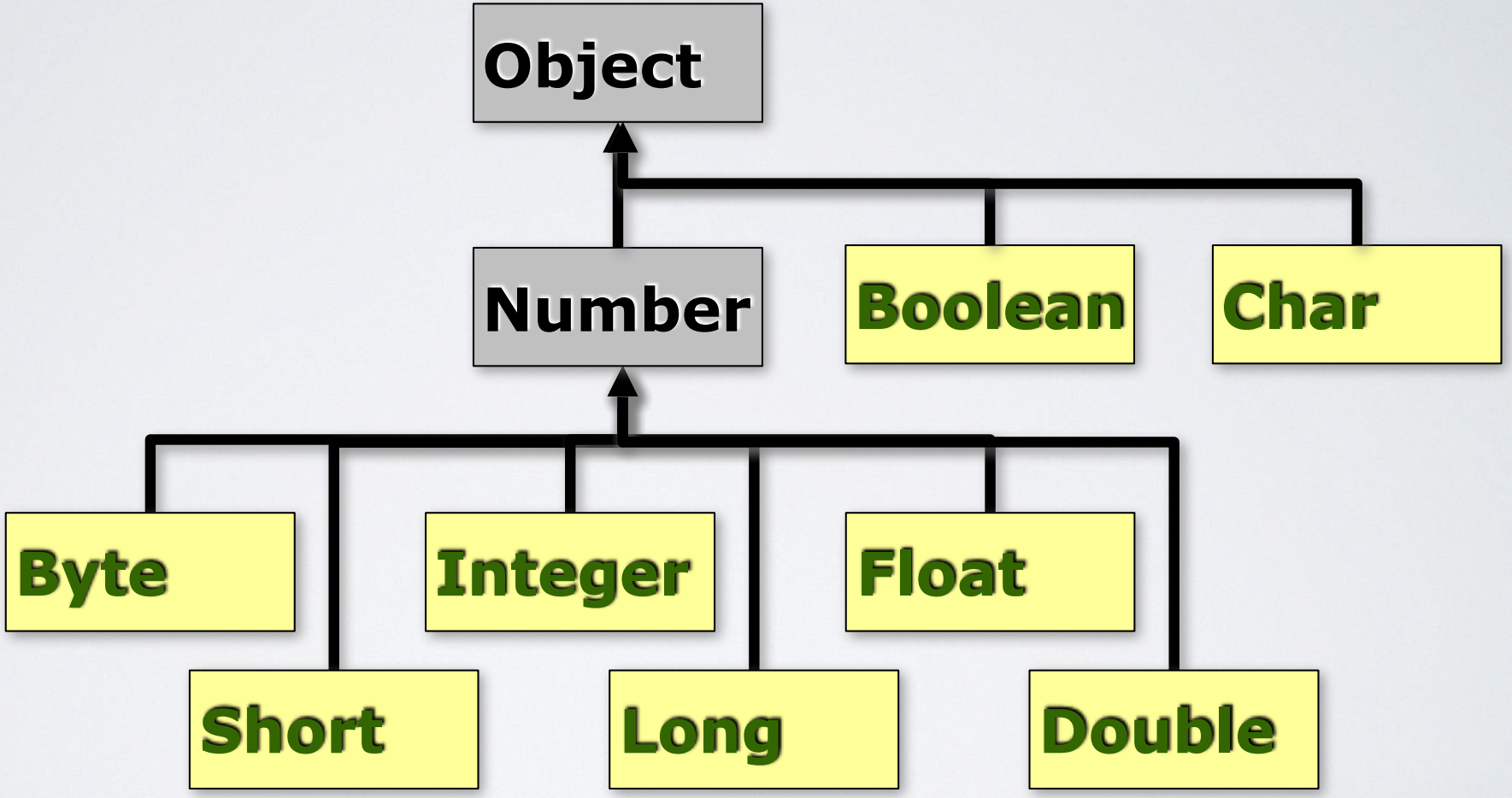
- Per risolvere questo problema, occorre usare un "cast" (forzatura):

```
int dollars = (int) balance;
```

CLASSI "WRAPPER"

- Utilizzate per trasformare in oggetti dati elementari
- Pattern generale dell'ingegneria del software

CLASSI "WRAPPER"



WRAPPER NUMERICI

- Sottoclassi di Number
- Metodi per
  - Trasformare una stringa in un numero e viceversa
  - Trasformare un numero in formati diversi (con possibile troncamento)
  - Rappresentazione testuale ottale, esadecimale, binaria

CHARACTER, BOOLEAN

- Character
  - Maiuscolo / minuscolo
  - Valore Unicode
  - Confronto, …
- Boolean
  - Conversione da/verso stringa
  - Confronto, …

ALTRI OPERATORI

- Moltiplicazione: *, Divisione: /
- Occhio alla precedenza degli operatori! (a+b)/c è diverso da a+b/c
- Uguaglianza: == Disuguaglianza: !=

```
if (a == b)
  System.out.println("a equals b");
```

Confronto tra espressioni booleane: &&, ||, !

```
if ( (a > b) && (b> c) )
   System.out.println("a > c");
```

Confronto bitwise: &, |, ^, ~

OPERATORI DI CONFRONTO

Operatore	Descrizione	espressione	Cosa restituisce
>	maggiore di	`x>y`	`true` se `x` è strettamente maggiore di `y`, `false` altrimenti
>=	maggiore o uguale di	`x>=y`	`true` se `x` è maggiore o uguale di `y`, `false` altrimenti
<	minore di	`x<y`	`true` se `x` è strettamente minore di `y`, `false` altrimenti
<=	minore o uguale di	`x<=y`	`true` se `x` è minore o uguale di `y`, `false` altrimenti
==	uguale	`x==y`	`true` se `x` è uguale a `y`, `false` altrimenti
!=	diverso	`x!=y`	`true` se `x` è diverso da `y`, `false` altrimenti

OPERATORI LOGICI

Operatore	Descrizione	espressione	Cosa restituisce
&&	and	`x&&y`	`true` se `x` e `y` sono entrambe vere, `false` altrimenti (almeno una delle due false)
||	or	`x||y`	`true` se almeno una tra `x` e `y` è vera (o entrambe), `false` altrimenti (entrambe false)
!	not	`!x`	`true` se `x` è falsa e `false` se `x` è vera.

OPERATORI BITWISE

Operatore	Descrizione	espressione	Cosa restituisce
&	and ‘bit a bit’	`x&y`	il valore determinato dal `and` tra i bit di `x` e `y`
|	or ‘bit a bit’	`x|y`	il valore determinato dal `or` tra i bit di `x` e `y`
^	xor (o or esclusivo) ‘bit a bit’	`x^y`	il valore determinato dal `xor` tra i bit di `x` e `y` (xor ritorna `true` solo se uno degli operandi è vero e l’altro è falso)
>> **<<**	shift destro (o sinistro) con segno	`x>>y`	sposta i bit della variabile `x` verso destra (o sinistra) di `y` posizioni, preservando il bit di segno,inserendo bit 0 in coda per >>. Con << si coinvolge l’ultimo bit che rappresenta il segno
>>>	shift destro senza segno	`x>>>y`	sposta i bit della variabile `x` verso destra di `y` posizioni, senza considerare il bit di segno
~	complemento	`~x`	inverte tutti i bit della variabile `x` (gli 0 diventano 1 e viceversa)

FONDAMENTI

- Introduzione
- Il mondo Java: caratteristiche e vantaggi
- Ambiente di sviluppo JVM,JDK...
- Variabili
- Tipi di dati primitivi, operatori matematici, di confronto e booleani
- Istruzioni condizionali e cicli iterativi
- Array e Stringhe
- Input / Output

CONTROLLI DI FLUSSO

```
if (<condizione>)
    <istruzione> o {blocco di istruzioni}
else
    <istruzione> o {blocco di istruzioni}
```

```
if (<condizione1>)
    <istruzione> o {blocco di istruzioni}
else if (<condizione2>)
    <istruzione> o {blocco di istruzioni}
…
else if (<condizioneN>)
    <istruzione> o {blocco di istruzioni}
else
    <istruzione> o {blocco di istruzioni}
```

Es.

```
if (a < 10)
    a += 10;
else
    a –= 10;
```

```
if (a < 10)
    a += 10;
else if ((a > 10) && (a < 20))
    a –= 10;
else
    a –= 20;
```

FLUSSO DI CONTROLLO

```
switch (<char> o <int>)
{
    case valore1:
        <sequenza di istruzioni>
        break;
    case valore2:
        <sequenza di istruzioni>
        break;
    ..
    default:
        <sequenza di istruzioni>
        break;
}
```

Es.

```
switch (digit)
{
    case 1:
        System.out.print("one");
        break;
    case 2:
        System.out.print("two");
        break;
    …
    case 10:
        System.out.print("ten");
        break;
    default:
        System.out.print("error");
        break;
}
```

FLUSSO DI CONTROLLO

```
while (<condizione>)
    <istruzione> o {blocco di istruzioni}
```

```
do <istruzione> o {blocco di istruzioni}
while (<condizione>);
```

```
for (<inizializzazione>; <condizione>; <aggiornamento>)
    <istruzione> o {blocco di istruzioni}
```

Es.

```
for (int i = 0; i < 20; i++)
   System.out.println(i);
```

WHILE

- Il ciclo while esegue una istruzione o un blocco di codice finché rimane verificata una certa condizione

- Diremmo: "fino a quando la condizione è vera esegui il blocco" ecco un esempio:

```
while(condizione) {
    // ...
}
```

- dove condizione deve essere una variabile o una espressione di tipo booleano. In questo caso quindi esprimiamo la volontà di eseguire tutte le istruzioni del blocco ripetutamente fino a quando condizione ha valore true

- Interessante capire cosa succede la prima volta che si accede al ciclo. Se condizione è false quando l'esecuzione arriva per la prima volta allo statement while il blocco di codice non sarà eseguito neanche una volta.

DO - WHILE

- il ciclo do-while è simile al while tanto da poter essere considerato una sua variante, ed infatti il frammento di codice:

```
do {
    // ...
} while(condizione);
```

- analogamente a quello sopra presentato, causa l'esecuzione del blocco tra graffe fino a quando la condizione è vera ma con la importante differenza che in questo caso condizione viene presa in considerazione alla fine del blocco. Quindi l'esecuzione del blocco di istruzioni viene effettuata almeno una volta, anche se la condizione risulta da subito false.

- Continuando il parallelo con la lingua italiana questo codice andrebbe letto: "esegui il blocco di codice e, poi, se condizione è vera fallo di nuovo, altrimenti smetti".

FOR

- Il ciclo for fornisce una semplice sintassi per accomodare:
- una espressione di inizializzazione eseguita solo una volta prima di iniziare il ciclo;
- una condizione di terminazione (o uscita) dall'esecuzione iterativa.
- una espressione di 'aggiornamento' (tipicamente un incremento) da eseguire al termine di ogni esecuzione del blocco;
- Con un codice simile al seguente:

```
for(inizializzazione; condizione; incremento) {
    // ...
}
```

ANALOGIE TRA CICLI

- È semplice convincersi con un esempio che un ciclo for ed uno while sono facilmente intercambiabili:

```
int i=0;

while(i < 10) {

    // ...

    i++;

}
```

- è identico a:

```
for(int i=0; i<10; i++) {

    // …

}
```

BREAK E CONTINUE

- Lo statement break serve per terminare l'esecuzione di uno o più blocchi di codice. In altre parole serve per "saltare fuori" da costrutti iterativi o switch-case
- Lo statement continue serve per saltare alla fine di un blocco ma in questo caso non viene terminato il ciclo ma solamente interrotta l'iterazione corrente e l'esecuzione salta immediatamente alla valutazione della condizione di terminazione. Eventualmente poi si procederà ad una nuova iterazione.
- Naturalmente non avrebbe senso utilizzare continue con lo statement switch ed infatti è sintatticamente proibito.

```java
int sum = 0;

for(int i = 1; i < 10; i++) {

    if(i%2 == 0) {
        break; // il blocco che viene interrotto con break
               // è quello "corrente" nel senso
               // "del ciclo corrente" (il for)
    }

    sum++;
}

System.err.println(sum);
```

```
int sum = 0;

for(int i = 1; i < 10; i++) {

    if(i % 2 == 0) {
        continue;
    }

    sum++;
}

System.err.println(sum);
```

FONDAMENTI

- Introduzione
- Il mondo Java: caratteristiche e vantaggi
- Ambiente di sviluppo JVM,JDK...
- Variabili
- Tipi di dati primitivi, operatori matematici, di confronto e booleani
- Istruzioni condizionali e cicli iterativi
- Array e Stringhe
- Input / Output

ARRAY

- Un array può essere definito come una "collezione organizzata di oggetti"
- Collezione: implica che tali oggetti siano dello stesso tipo, così, prendendo spunto dal mondo reale
- Organizzata: implica che sia possibile identificare univocamente tutti gli oggetti dell'array in modo sistematico.
- In Java viene fatto tramite l'uso di indici numerici che, in un array di dimensione N, vanno da 0 ad N-1

ARRAY

Due modi per costruire un array:

1) NomeTipo[] nomeArray = new NomeTipo[lunghezza];

   Es. int[] a = new int[10];

2) NomeTipo[] nomeArray = {elenco elementi}

   Es. int[] a = {1, 3, 5, 7};

Nel primo caso l'array viene creato in memoria ma è vuoto, i suoi elementi sono null.

Nel secondo caso l'array viene creato in memoria e i suoi elementi sono inizializzati.

ARRAY: ESEMPIO D'USO

```
public class ArrayManipulator {

   public static void main(String[] args) {
      for (int i = 0; i < args.length; i++) {
          System.out.println(args[i]);
      }
   }
}
```

Eseguendo: java ArrayManipulator Hello World!

L'output su schermo è:

Hello

World!

LA CLASSE STRING

- Classe che modella sequenze immutabili di caratteri
- Sintassi semplificata
  - String s1 = “Hello” ;
  - String s2 = s1 + “ Java”;
- Offre molti metodi: confronto, ricerca, derivazione di nuove stringhe, informazioni generali, …

STRINGHE: CONFRONTO E RICERCA

- public boolean equals(String s)
  - Restituisce true se il parametro contiene gli stessi caratteri dell'oggetto corrente
- public boolean equalsIgnoreCase(String s)
  - Idem, trascurando la differenza tra maiuscole e minuscole
- public int indexOf(String s)
  - Restituisce la posizione, all'interno della sequenza di caratteri, in cui inizia la stringa indicata come parametro (-1 se non esiste)

STRINGHE: DERIVAZIONE E INFORMAZIONI

- public String toUpperCase()
  - Restituisce una nuova stringa contenente gli stessi caratteri in versione maiuscola
- public String replace(char oldChar, char newChar)
  - Restituisce una nuova stringa in cui tutte le occorrenze di oldChar sono sostituite con newChar
- public String substring(int beginIndex, int endIndex)
  - Restituisce una nuova stringa formata dai caratteri che iniziano alla posizione beginIndex fino a endIndex escluso
- public int length()
  - Restituisce la lunghezza in caratteri della stringa corrente

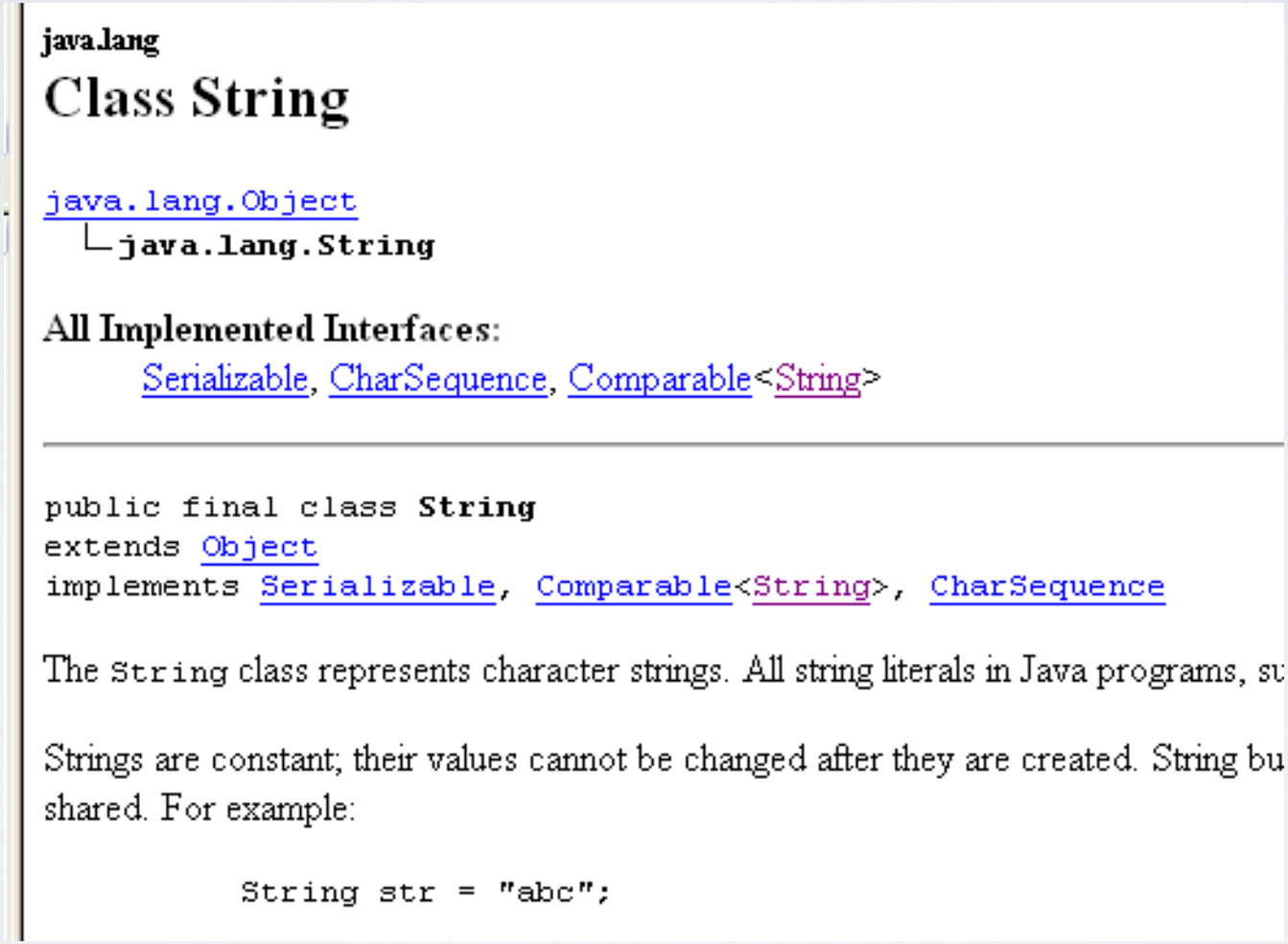
STRINGHE

Le variabili di tipo String, definita nella Java Class Library, sono sequenze di char che possono essere inizializzate utilizzando le virgolette:

Esempi:

```
String greetings = "Hello";   // creazione

String message = greetings + " Michele!"   // concatenazione

String part = greeting.substring(0,4);   // part è "Hell"

if (part.equals("Hell"))   // confronto

if (greetings.contains(part))   // altro tipo di confronto

int length = greetings.length();   // misura della lunghezza

char c = greetings.charAt(4);   // c è 'o'
```

STRINGHE



java.lang

Class String

java.lang.Object
  └ **java.lang.String**

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, su

Strings are constant; their values cannot be changed after they are created. String bu
shared. For example:

```
String str = "abc";
```

FONDAMENTI

- Introduzione
- Il mondo Java: caratteristiche e vantaggi
- Ambiente di sviluppo JVM,JDK...
- Variabili
- Tipi di dati primitivi, operatori matematici, di confronto e booleani
- Istruzioni condizionali e cicli iterativi
- Array e Stringhe
- Input / Output

INPUT E OUTPUT

Per scrivere in console si utilizza *System.out.println() visto in alcuni esempi*

Per leggere dalla console dando modo ad un utente di digitare parametri in modo interattivo si utilizza la classe *Scanner*

```
import java.util.Scanner;

public class FondamentiInputOutput {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Come ti chiami? ");
        String name = in.nextLine();
        System.out.print("Quanti anni hai? ");
        int age = in.nextInt();

        System.out.println("Ciao " + name + ". Il prossimo anno avrai "
                + (age + 1) + " anni!");
    }
}
```

Notiamo che per istanziare una variabile di tipo *Scanner* abbiamo utilizzato la parola chiave *new*, vedremo che questo sarà il default in tutti i casi in cui non stiamo usando tipi primitivi.

ESERCIZI

- Esercizio: Scrivere un programma SoloVocali che chiede all'utente di inserire una stringa e ne stampa le sole vocali. Per esempio, se si immette la stringa "Viva Java", il programma stampa "iaaa".

- Esercizio: Scrivere un programma Lunghezze che chiede all'utente di inserire una sequenza di stringhe e conclusa dalla stringa vuota, e poi stampa la somma delle lunghezze delle stringhe che iniziano con una lettera maiuscola. Per esempio, se si immettono le stringhe "Albero", "foglia", "Radici", "Ramo", "fiore" (e poi "" per finire), il programma stampa 16.

ESERCIZI

- Esercizio: Scrivere un programma SommaPariDispari che prevede un array di 10 numeri interi contenente valori a piacere (senza bisogno di chiederli all'utente) e stampa "Pari e dispari uguali" se la somma dei numeri in posizioni pari dell'array è uguale alla somma dei numeri in posizioni dispari, altrimenti il programma stampa "Pari e dispari diversi". (Il programma deve essere scritto facendo finta di non sapere quali siano i valori inseriti nell'array)

- Esercizio: Scrivere un programma SecondoArray che chiede all'utente di inserire 10 numeri interi e li memorizza in un array. Successivamente, crea un nuovo array di dimensione pari al numero di valori maggiori o uguali a zero inseriti dall'utente. Copia tutti i valori maggiori o uguali a zero nel nuovo array e ne stampa i valori in ordine inverso.

Java™

Dott. Antonio Giovanni Lezzi

JAVA

- Fondamenti
- Programmazione Object Oriented
- Java e OOP nel dettaglio
- Java Avanzato
- Network
- Database
- Web

Dott. Antonio Giovanni Lezzi

Afol Milano

PROGRAMMAZIONE OBJECT ORIENTED

- Classi: definizione attributi e metodi di classe con costruttore
- Introduzione alla programmazione ad oggetti e progettazione (OO)
- Ereditarietà e sue applicazioni
- Classi e metodi abstract
- Interfacce e loro impiego
- Polimorfismo e sue applicazioni pratiche

Dott. Antonio Giovanni Lezzi Afol Milano

CLASSE ED OGGETTI

- **Classe** è una collezione di uno o più oggetti contenenti un insieme uniforme di attributi e servizi, insieme ad una descrizione circa come creare nuovi elementi della classe stessa;

- Un **Oggetto** è dotato di stato, comportamento ed identità; la struttura ed il comportamento di oggetti simili sono definiti nelle loro classi comuni; i termini istanza ed oggetto sono intercambiabili.

MODELLARE LA REALTÀ

STATO
via1: verde
via2: rosso

STATO
motore:acceso
velocità: 0

COMPORTAMENTO
Parti!
Frena!
Sterza!

METODI

- Tutti i linguaggi di programmazione forniscono la possibilità di definire, sotto un nome, gruppi di istruzioni / insiemi di linee di codice / blocchi di espressioni
- Possiamo riutilizzare un blocco di codice in molte parti del programma, semplicemente richiamando il nome con cui l'abbiamo definito, senza dover riscrivere tutto il blocco ogni volta.
- Questi aggregati, a seconda del linguaggio, si chiamano funzioni, procedure, subroutines o sottoprogrammi.
- In Java utilizziamo la logica definita dai blocchi di istruzioni per rappresentare il comportamento di classi di oggetti, prendono il nome di **metodi**.

METODI IN JAVA

La sintassi per la definizione di un metodo è simile alla definizione di una variabile:

```
[public|protected|private] [static] [final] Tipo
    identificatore([Tipo1 parametro1, Tipo2 parametro2, ...,
TipoN parametroN])
    [throws Eccezione1, Eccezzione2, ...] {

        // blocco di codice appartenente al metodo

    return varTipo;
}
```

METODO: I PARAMETRI

La sezione ***[Tipo1 parametro1, ... TipoN parametroN]*** è detto insieme dei parametri (formali) del metodo in cui si dichiara il tipo ed il nome simbolico delle variabili che il blocco di codice dovrà ricevere dal programma chiamante per poter svolgere il proprio compito.

```
public double areaRettangolo(double base,double
altezza){
    double a = base * altezza;
    return a;
}
```

METODO: VALORE DI RITORNO

- Il valore specificato accanto a return deve essere del medesimo tipo specificato nella dichiarazione del metodo ma deve essere omesso se il metodo è stato dichiarato come void.

- Traducibile come "vuoto" dichiarare un metodo void significa dire che il metodo non ritornerà alcun valore ed in tal caso la keyword return può essere anche omessa.

```
public double areaRettangolo(double base, double altezza) {
     if(base == 0.0 || altezza == 0.0)
        return 0.0; // questo return causa l'uscita dal metodo

    double a = base * altezza;

    return a;
}
```

METODI: IL RICHIAMO

- Per chiamare il nostro metodo all'interno della classe in cui lo stiamo definendo (fuori dalla classe di definizione occorre una sintassi differente, che vedremo in seguito), sarà sufficiente scrivere qualcosa del genere:

```
double areaCalcolata;

areaCalcolata = areaRettangolo(4.0,2.0);
```

- Una volta eseguito questo codice la variabile *areaCalcolata* avrà valore 8.0.
- I valori che inseriamo al momento della chiamata del metodo sono detti *parametri attuali*
- Possiamo pensare che la macchina virtuale durante l'esecuzione del codice della chiamata al metodo salti letteralmente all'inizio della dichiarazione del metodo, inizializzi le variabili (formali) ***base*** ed ***altezza*** con i valori (attuali) 4.0 e2.0, esegua quindi il corpo del metodo fino a quando non incontra return, a quel punto prenda il valore dell'argomento di return e lo utilizzi per assegnare il valore alla variabile areaCalcolata.