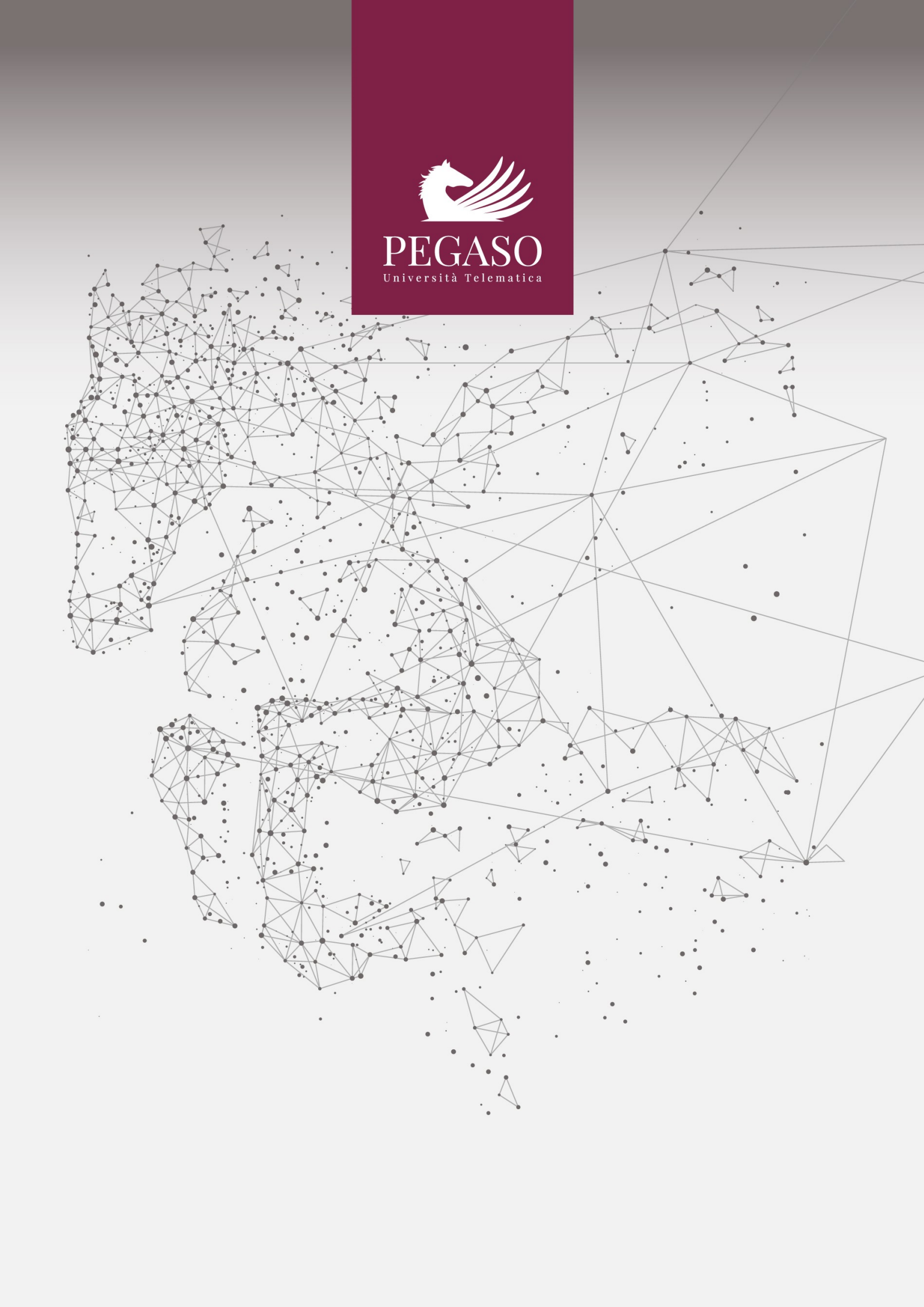
PEGASO
Università Telematica

Indice

1. Origini

Java è un linguaggio di programmazione nato agli inizi degli anni novanta da un gruppo di lavoro della Sun Microsystems. Il gruppo di lavoro guidato da James Gosling si era messa in attività sul nuovo linguaggio in gran segreto a partire dal 1991.

Erano decenni caldi per l'informatica con le comunità scientifiche e le realtà industriali in pieno fermento.

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 straordinariamente.

In quegli anni il C (Dennis Ritchie,1972) era nel pieno della sua diffusione ma la sua struttura procedurale ed il suo approccio "low level" (puntatori, allocazione statica della memoria, etc.) lo rendevano un linguaggio potente ma al tempo stesso impegnativo e di difficile adozione per progetti di sviluppo sempre più complessi e con team di sviluppatori sempre più grandi.

A cavallo fra gli anni '70 e '80 anche un nuovo player era entrato in campo: il linguaggio C++ introdotto da Bjarne Stroustrup. Il C++ si ispirava ai principi della Programmazione Orientata agli Oggetti e durante gli anni '80 si affermò come linguaggio di riferimento per i progetti di sviluppo più grandi e ambiziosi con piena diffusione negli anni '90.

In questo scenario, Sun Microsystem aveva individuato nella realizzazione di un nuovo linguaggio di programmazione uno dei componenti essenziali per catturare la "next-wave" nell'evoluzione dell'informatica. L'idea iniziale per questo nuovo linguaggio è che dovesse servire per scrivere programmi per il controllo di elettrodomestici (TV, frigorifero,...). Il C++ era pur sempre una evoluzione del C e quindi non presentava quelle caratteristiche di semplicità e portabilità che si ritenevano necessarie.

Il piano del gruppo di lavoro alla Sun 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 (il C++ era pur sempre una evoluzione del C), che fosse semplice da usare e si occupasse automaticamente della gran parte dei dettagli di gestione della memoria. Inoltre si richiedeva che fosse un linguaggio che poteva essere eseguito e potesse essere facilmente portato su processori e dispositivi diversi e che non richiedesse compilatori o interpreti troppo sofisticati (i produttori degli elettrodomestici non avrebbero investito risorse in quel settore).

L'idea fu di introdurre un unico, semplice linguaggio intermedio (chiamato byte-code) per il quale potessero facilmente essere scritti interpreti ad-hoc. L'interprete del byte-code Java è detto Java Virtual Machine (JVM). JVM permette di eseguire i programmi scritti nel nuovo linguaggio in qualsiasi sistema

operativo supportato dalla tecnologia di Sun. Il target del compilatore non era più il sistema operativo della macchina host, ma la virtual machine installata su di esso.

Dal gruppo di lavoro in Sun nacque nel 1992 il linguaggio Oak (in italiano "quercia"), nome che successivamente fu cambiato in Java (una varietà di caffè indonesiana; il logo adottato è una tazzina per tale bevanda) per problemi di copyright visto che un altro linguaggio di programmazione Oak esisteva già. Ma la chiave di volta per il successo di Java fu l'idea del gruppo di lavoro di catturare con il loro linguaggio la vera "next-wave" degli anni 90: Internet.

Se un primo momento Sun decise di destinare questo nuovo prodotto alla creazione di applicazioni complesse per piccoli dispositivi elettronici, dopo breve tempo, nel 1993 con l'esplosione di Internet, ci si rese conto che Java poteva essere usato per distribuire applicazioni su Internet e quindi come linguaggio di programmazione per Internet: il byte-code poteva essere distribuito via Web ed essere eseguito sui computer degli utenti. Risultato quindi essere essenziale l'indipendenza dalla piattaforma hardware e quindi vincente l'idea di perseguirla. Nacquero ed ebbero successo le applet: programmi Java eseguibili dentro al browser Web (la JVM installata come plug-in del browser).

Nel giro di pochi mesi il gruppo di lavoro in Sun riuscì a sviluppare un web browser interamente in Java (Sun's HotJava, 1994) 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. Nel 1995 Netscape Corporation inserì nel suo browser Navigator la JVM e quindi il supporto per le applet. Fu l'anno della svolta per Java e di una rivoluzione nel mondo di Internet: grazie agli applet le pagine web diventarono interattive a livello client, ovvero le applicazioni vengono eseguite direttamente sulla macchina dell'utente di internet e non su un server remoto. Per esempio gli utenti poterono utilizzare giochi direttamente sulle pagine web e usufruire di chat dinamiche e interattive.

La prima versione beta di Java fu rilasciata a luglio del 1995, mentre la prima release fu annunciata e resa disponibile il 23 gennaio dell'anno 1996.

Con il tempo altre tecnologie soppiantano Java nell'ambito di Internet (e.g. JavaScript) Java rimane comunque uno tra i principali linguaggi per lo sviluppo di applicazioni desktop e distribuite, in particolare in ambiente aziendale (enterprise).

Nel 2010 Sun Microsystems storica e gloriosa società americana è stata assorbito da Oracle. Oggi quindi Java è a tutti gli effetti un prodotto Oracle.

2. Introduzione a Java

Le proprietà generali di Java sono:

- **Sintassi semplice e simile al C/C++:** è un linguaggio piuttosto semplice, con una sintassi simile a quella del C e del C++. Anche il linguaggio di Microsoft C# nato nel 2002 per contrastare Java ha a sua volta una sintassi molto simile. Non si è voluto infatti a livello di sintassi porre alcun sforzo aggiuntivo di apprendimento.
- **Gratuità:** per scrivere applicazioni commerciali non bisogna pagare licenze a nessuno. Il codice Java infatti si può scrivere anche utilizzando un qualsiasi editor di testo come il blocco note e non per forza un complicato IDE con una costosa licenza. Esistono anche tanti strumenti di sviluppi a pagamento che possono accelerare lo sviluppo di applicazioni Java. Tuttavia esistono anche eccellenti prodotti gratuiti ed open source come Eclipse e Netbeans.
- **Robustezza:** grazie ad (i) una gestione delle eccezioni chiara e funzionale, (ii) un meccanismo automatico della gestione della memoria (Garbage Collection) che esonera il programmatore dall'obbligo di dover deallocare memoria quando ce n'è bisogno (punto fra i più delicati nella programmazione), (iii) un compilatore molto «severo» che segnala ogni situazione di utilizzo poco chiara o che esce dall'uso non già previsto del linguaggio, garantendo al programma maggiori chance di corretto funzionamento. Ad esempio il compilatore applica rigide regole sintattiche (il ; alla fine di ogni istruzione) e sui tipi. Jave è un linguaggio fortemente tipato: il programmatore è tenuto a specificare il tipo di ogni variabile, e il compilatore richiede e garantisce che i valori di tali variabili verranno sempre usati in modo coerente rispetto al tipo.
- **Ricca disponibilità di librerie e standardizzazione**: Java possiede un'enorme libreria di classi standard permettendo agli sviluppatori, anche neofiti, di creare applicazioni in breve tempo anche e soprattutto indipendentemente dalla piattaforma in cui si sviluppa. Per esempio, è relativamente semplice gestire finestre di sistema come interfacce grafiche, collegamenti a database, connessioni di rete. Inoltre, grazie alle specifiche di Oracle, non esistono per lo sviluppatore problemi di standardizzazione come compilatori che compilano in modo differente.
- **Facilità di sviluppo**: questo non significa che sia un linguaggio semplice. Sicuramente ad esempio il fatto che siano stata eliminata l'aritmetica dei puntatori è stato senza dubbio una semplificazione ma Java resta un linguaggio molto complesso considerando la sua potenza e tenendo presente che obbliga ad una programmazione ad oggetti pura e rigorosa. Nel tempo il linguaggio arricchito sempre più di nuovi costrutti (ad esempio i Generics o le Annotazioni) che

lo hanno reso più difficile da imparare ma al tempo stesso più potente per scrivere programmi di più ampia portata in un tempo relativamente breve. Quindi più propriamente dovremmo parlare di potenza di sviluppo. Java non è facile da imparare ma una volta comprese le logiche fondamentali si ha fra le mani uno strumento potentissimo di sviluppo.

Sulla base delle proprietà generali sopraelencate, i pilastri e le proprietà caratterizzanti del linguaggio sono:

- **Linguaggio fortemente orientato agli oggetti:** Java è più chiaro e schematico di altri linguaggi orientati agli oggetti al punto che praticamente obbliga lo sviluppatore a programmare ad oggetti in modo rigoroso. I paradigmi fondamentali della programmazione ad oggetti (ereditarietà, incapsulamento, polimorfismo) sono più facilmente apprezzabili e comprensibili. Tuttavia il linguaggio è pensato perché essi siano usati in modo stretto e rigoroso. In sostanza lo sviluppatore è costretto a pensare ad oggetti mentre scrive il programma. Tuttavia con il passare del tempo il linguaggio si è evoluto e si è aperto anche ad altri paradigmi di programmazione (es. funzionale).
- **Indipendenza dall'architettura:** grazie al concetto di macchina virtuale, ogni applicazione, una volta compilata, potrà essere eseguita su di una qualsiasi piattaforma (per esempio un PC con sistema operativo Windows o una workstation Unix). Questa è sicuramente la caratteristica più importante di Java. Infatti, nel caso in cui si debba implementare un programma destinato a diverse piattaforme per esempio non ci sarà la necessità di doverlo convertire radicalmente da piattaforma a piattaforma.
- **Byte-code e Java Virtual Machine:** Ciò che rende di fatto possibile l'indipendenza dalla piattaforma è la Java Virtual Machine (JVM) un software che svolge il ruolo di interprete (ma non solo) per i programmi scritti in Java. Si parla di macchina virtuale perché questo software è stato implementato per simulare l'hardware di un calcolatore ipotetico che ha caratteristiche simili (semplificate) a quelle delle architetture hardware più comuni. Il linguaggio Java si basa quindi su un approccio che combina compilazione (in byte-code) e interpretazione da parte di un JVM (del byte-code). Ogni piattaforma ovvero ogni sistema operativo girerà una propria JVM che consentirà l'interpretazione e quindi l'esecuzione del byte-code su quella piattaforma.

Il seguente programma (Fig. 1) visualizza sullo schermo un semplice saluto (Hello World!).

Attenzione: Java è un linguaggio case sensitive, ovvero distingue le lettere maiuscole da quelle minuscole. Quindi scrivere «Class» non è come scrivere «class» e in questo esempio causerà anche un errore in fase di compilazione.

```
1       public class HelloWorld
2       {
3           public static void main (String[] args)
4           {
5               //visualizza un messaggio di saluto
6               System.out.println («Hello World!»);
7           }
8       }
```

***Fig. 1**: Programma "HelloWorld"*

Si tratta del tipico primo programma che si scrive in un nuovo linguaggio di programmazione. Lo usiamo per iniziare a familiarizzare con la sintassi e qualche concetto fondamentale come quello di classe e di metodo. Vedremo anche come compilare e come andare in esecuzione questo programma. Analizziamolo nel dettaglio.

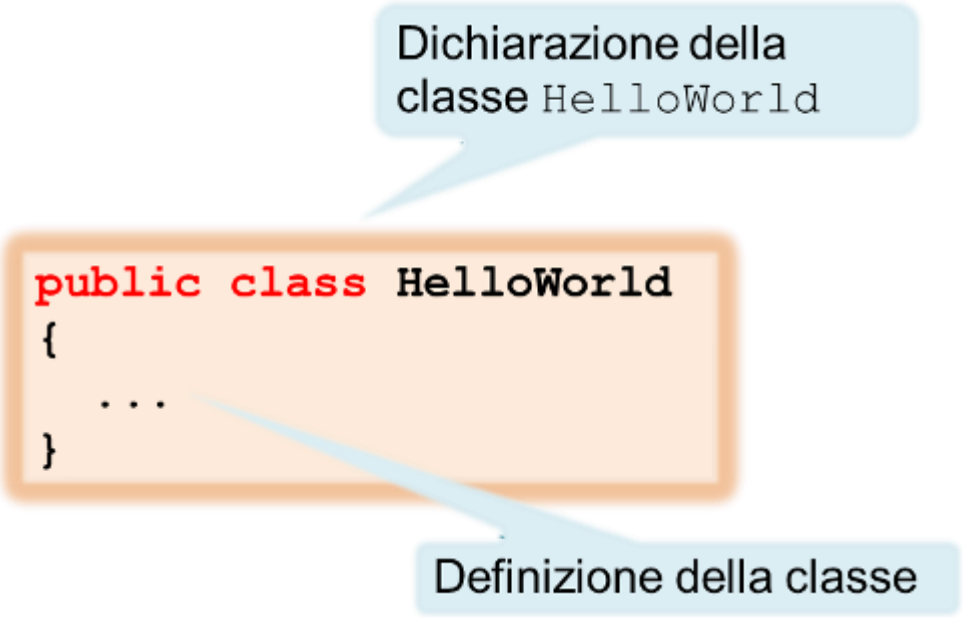

***Fig. 2:** Dichiarazione della classe HelloWorld*

La prima riga con le relative parentesi graffe (righe 1,2,8) dice che stiamo definendo la classe HelloWorld. Come mostrato in Fig. 2, la specifica della classe, ovvero la sua definizione, si trova tra le parentesi graffe. HelloWorld è l'identificatore (ovvero il nome) della classe.

Un programma Java è costituito da un insieme di classi (almeno una); public significa che questa classe è pubblica ovvero può essere utilizzata da qualunque altra classe del programma.

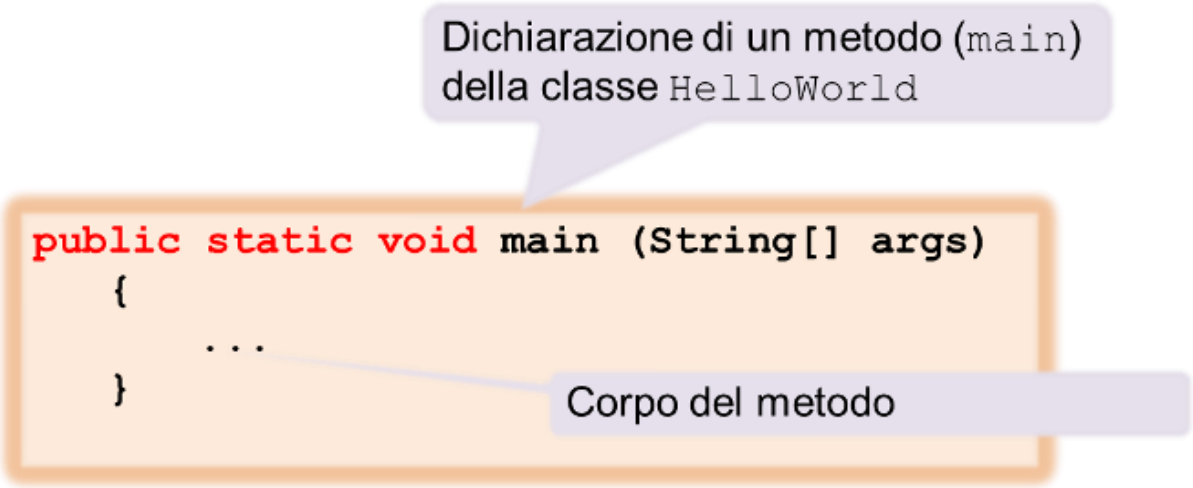

Fig. 3: *Dichiarazione del metodo main()*

La parte interna della dichiarazione di una classe è dedicata alla definizione dei metodi della classe. Un metodo inteso come l'implementazione di un'operazione è una funzionalità della classe messa disposizione del resto del programma o di altre parti (altri metodi) della stessa classe.

In questo caso alla riga 3 e le relative parentesi graffe (righe 4 e 7) si definisce un metodo della class HelloWorld denominato main (principale) come mostrato in Fig. 3. È un metodo particolare e va tenuto bene a mente. Infatti in ogni applicazione Java questo metodo deve essere definito in ogni applicazione Java perché rappresenta il punto di partenza dell'esecuzione di ogni programma. La prima istruzione che verrà quindi eseguita in fase di esecuzione sarà quella che la JVM troverà subito dopo l'apertura del blocco di codice che definisce questo metodo.

Oltre alla parola «main» la riga contiene le parole «public», «static» e «void». Si tratta di una lista di modificatori, ovvero di parole chiave che caratterizzano la classe come in italiano un aggettivo può caratterizzare un sostantivo.

public significa che questo metodo è visibile e quindi può essere invocato anche al di fuori della classe in cui è definito. Per questo primo programma che è costituito da un unico file ed un'unica classe, questo modificatore non è di alcuna utilità ma più avanti ne risulterà più chiara l'importanza e il significato.

Riguardo a static, la definizione è abbastanza complessa. Per il momento basta sapere che è essenziale per la definizione del metodo main().

void è il tipo di ritorno del metodo. Significa «vuoto» e quindi questo metodo non restituisce nessun tipo di valore. Il metodo main() non deve mail avere un tipo di ritorno diverso da void.

Alla destra di main, in generale alla destra del nome del metodo (detto anche identificatore del metodo), si definisce sempre una coppia di parentesi tonde che racchiude opzionalmente una lista di parametri (detti anche argomenti del metodo). Il metodo main() in ogni caso vuole sempre come

parametro un array di stringhe (gli array li vedremo più avanti). Si noti che args è l'identificatore (nome) dell'array, ed è l'unica parola che può variare nella definizione del metodo main(), anche se per convenzione si usa sempre args.

Il corpo del metodo è racchiuso tra le parentesi graffe.

Fig. 4: Commento in Java

Alla riga 5 è specificato un commento (vedi anche Fig. 4). I commenti servono per rendere più comprensibile il codice. In Java è un commento tutto ciò che si trova a destra di // (una sola riga) o tutto ciò che si trova tra /* e */ (anche su più righe).

Fig. 5: *Comando di stampa a video del messaggio "Hello World!"*

Alla riga 6 è mostrato il comando (vedi anche Fig. 5) che stamperà a video la stringa Hello World! Anche in questo caso si usano degli argomenti che verranno trattati più avanti. Per il momento basti sapere che si sta usando una libreria standard di Java e invocando un suo metodo chiamato println() sull'oggetto System.out.

System.out è un oggetto che rappresenta il canale di output standard del sistema (la console...). Si tratta di un oggetto istanza di una classe definita altrove e disponibile come libreria standard di Java.

println è un metodo dell'oggetto System.out che stampa un messaggio e va a capo. Come tutti i comandi, println deve essere terminato con punto e virgola dopo la specifica dei parametri previsti, in questo caso la stringa "Hello World!".

Per chi viene dalla programmazione procedurale troverà naturale usare Java scrivendo programmi costituiti da un unico listato che sarà posto all'interno di uno stesso metodo che è necessariamente il metodo main che è il metodo invocato per primo e da cui parte il flusso di esecuzione. Quando il programma da realizzare è articolato diventa conveniente identificare sottoproblemi che possono essere risolti individualmente, scrivere sottoprogrammi che risolvono tali sottoproblemi e richiamare i sottoprogrammi dal programma principale (main). In Java i sottoprogrammi si realizzano tramite metodi ausiliari che realizzano quello che in altri linguaggi di programmazione sono chiamati funzioni, procedure oppure (sub)routines. Quindi in aggiunta al main possiamo trovare definiti altri metodi per realizzare tali sottoprogrammi. Essi saranno definiti come «private» specificando così che si trattano di metodi ausiliari ovvero solo invocabile da un altro metodo della stessa classe.

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

...
   private static <identificatore metodo2> (<lista parametri>)
   {
     ...
   }
...

}
```

Fig. 6: *Struttura programma procedurale scritto in Java*

Ricapitolando, come mostrato in Fig. 6 si ha:

- una sola classe (con nome arbitrario del programma);
- il solo metodo main (scritto esattamente come nell'esempio);
- Il corpo del main conterrà tutti i comandi del programma;
- Il main potrà invocare metodi ausiliari definiti come private.

Siccome non ci possono essere metodi definiti al di fuori delle classe tutti questi metodo devono esserlo rispetto ad una classe che sarà chiamata con il nome del programma che stiamo scrivendo.

Tuttavia questo uso di Java purché tecnicamente possibile non è quello per cui è stato concepito il linguaggio anzi lo snatura completamente avendo usato la definizione di una classe per scrivere un

programma procedurale e definendo le sottoroutine come metodi. Funziona ma è un uso che definirei degenere di questo linguaggio.

Non ha senso imparare il linguaggio Java senza sfruttare il supporto che esso offre alla programmazione ad oggetti.

Quindi vedremo direttamente come creare applicazioni che saranno costituite da un certo numero di classi che daranno vita ad un numero di istanze di oggetti che interagendo realizzeranno quello per cui il programma è stato scritto.

Gli studenti che hanno esperienza con la programmazione procedurale troveranno profonde differenze con i concetti dell'object orientation. Il consiglio per questi ultimi è di non ancorarsi troppo a quello che già si conosce. Èinutile forzare una classe ad avere il ruolo che poteva avere una sottoroutine nella programmazione procedurale. Meglio far finta di partire da zero! Per chi invece già conosce altri linguaggi orientati agli oggetti, le nozioni che seguono dovrebbero risultare chiare e semplici.

Quello che resterà vero è che in un programma Java scritto secondo i dettami dell'object-oriented è il main posto all'interno di costrutto class che sarà nominato con il nome di tutto il programma. Per i primi programmi che presenteremo, identificheremo il programma con una classe avente lo stesso nome e contenente un metodo di nome main. La funzionalità del programma sarà quella realizzata dal metodo main.

3. Il modello astratto di Java: rappresentazione ad oggetti

Un attributo è una proprietà statica di un oggetto e per questo specificato a livello di definizione della classe:

- nome, età, peso sono attributi della classe Persona;
- colore, peso, anno, modello sono attributi della classe Auto;
- forma, peso, colore e tipo di materiale: attributi della classe Soprammobile.

I nomi degli attributi devono essere unici all'interno di una classe per evitare ambiguità. Ma se il nome è coerente con il significato semantico della proprietà non può essere altrimenti.

Ogni attributo ha un nome, un valore corrente, un insieme di valori ammessi (dominio). Ad esempio, l'attributo colore della classe Automobile può ha un dominio dei valori {bianco, nero, argento, grigio, blu, rosso, giallo, verde}. Il dominio per l'attributo peso è semplicemente quello dei numeri interi positivi. In situazioni più complesse, il dominio può essere dato a sua volta da istanze di un'altra classe. Ad esempio, sempre la classe Automobile può avere un attributo denominato motorizzazione istanza di una classe Motore, caratterizzato a sua volta da potenza, cavalliFiscali, numeroCilindri e tipoCarburante.

L'insieme degli attributi è comune a tutti gli oggetti di una stessa classe in quanto condividono la stessa semantica essendo realizzazioni concrete di uno stesso concetto, anche se in generale i valori degli attributi saranno in genere differenti nelle diverse istanze; possono anche coincidere ma è un puro caso. Ad esempio, l'attributo età ha il valore "24" nell'oggetto Franco Lorusso della classe Persona ed ha valore "32" nell'oggetto Mario Rossi sempre della classe Persona

L'insieme dei valori degli attributi ad un dato istante di tempo costituisce lo stato dell'oggetto. E' possibile assegnare ad un attributo un valore di default assegnato al momento dell'istanziazione dell'oggetto.

L'identità è un'altra proprietà degli oggetti di una classe che si differenzia dagli attributi per il carattere di unicità che presenta.

Nel mondo reale ogni oggetto è unico. Quando si parla di una particolare sedia si sa che si sta parlando della stessa cosa, anche se con passare del tempo alcune sua caratteristiche vengono modificate, ad esempio il costo. Eppure sempre della stessa sedia si parla. In altre parole, la sedia ha un'identità che si conserva nel tempo indipendentemente dalle modifiche che l'oggetto subisce. Viceversa, in un dato momento, oggetti diversi possono avere gli stessi valori per gli attributi ma restano comunque oggetti ben distinti, ad esempio le sedie che si usano ad uno stesso tavolo.

Tutti gli oggetti quindi hanno un'identità e sono distinguibili gli uni dagli altri per la loro esistenza intrinseca e non per le proprietà descrittive che possono avere. Analogamente, gli oggetti del mondo object-oriented hanno un'identità immutabile nel tempo che li distingue e li rende unici indipendentemente dal valore dei suoi attributi che possono anche essere di uguale valore in un determinato momento per oggetti diversi.

Dal punto di vista dell'implementazione, per gestire e disporre dell'identità di un oggetto non è consigliabile definire un attributo id specifico per passarlo come parametro in una chiamata di un'operazione su quell'oggetto. Invece, l'identità è preferibilmente gestita come se fosse un attributo predefinito, unico e immutabile per tutta la vita dell'oggetto realizzato generalmente mediante un codice numerico assegnato direttamente dal sistema al momento della creazione dell'oggetto. Tale codice numerico non può essere letto ma che viene usato per tradurre ogni riferimento all'oggetto. Trattasi di una rappresentazione compatta e usata per far riferimento all'oggetto a basso livello. Definire un attributo id non è più efficiente in quanto non vi è alcun risparmio di memoria, anzi lo spazio usato può essere anche in più avendo definite e fatto uso di un attributo non necessario.

Non bisogna confondere gli identificatori interni con gli attributi del mondo reale. Gli identificatori interni sono definiti per comodità di implementazione e supportati dalla maggior parte dei linguaggi orientati agli oggetti per far riferimento agli oggetti. Ma gli identificatori interni non hanno significato nel dominio del problema. Nel paradigma orientato agli oggetti l'unità elementare di scomposizione non è più la procedura, ma l'oggetto inteso come astrazione di dati ed operazioni incapsulati in un'unica entità con un comportamento e un significato semantico strumentare a risolvere il problema che si sta trattando. Gli oggetti rappresentano entità che possono svolgere operazioni, riportare e cambiare il loro stato e comunicare con gli altri oggetti del sistema.

In un programma di tipo procedurale, si è soliti iniziare a ragionare in maniera top-down, partendo cioè dal main e creando mano a mano tutte le procedure necessarie. Tutto questo non ha più senso nella programmazione ad oggetti dove è richiesto un diverso approccio di ragionamento.

In una visione ad oggetti del software, il software deve essere pensato con una visione diversa. Dobbiamo, cioè, essere in grado di identificare gli oggetti che entrano in gioco nel programma che vogliamo sviluppare e individuare e orchestrare le interazioni fra di essi per raggiungere lo scopo del programma.

L'approccio orientato agli oggetti deve essere visto come un nuovo modo di ragionare sul software. Questa è una prospettiva diversa di quella adottata per molti anni quando con il termine object oriented (orientato agli oggetti) si è indicato più un approccio alla programmazione. In effetti l'approccio object oriented è nato con lo sviluppo di sofware che faceva uso di linguaggi di programmazione object-oriented (come Java, ma anche Smalltalk e C++).

Tuttavia, ci si è anche resi conto che l'approccio l'object oriented riflette anche e soprattutto un modo di pensare alla soluzione software a 360° più che una specifica tecnica di programmazione.

Come nuovo modo di pensare la soluzione software è opportuno che sia adottato in tutte le fasi di produzione del software a partire già dalla fase di analisi dei requisiti e la progettazione dell'architettura sofware in modo che esprima tutta la sua potenza e utilità in termini di riuso del codice e manutenzione e gestione di progetti di grande dimensioni.

Si fa notare che la parte più gravosa dello sviluppo del software è la manipolazione dell'essenza del problema a causa della sua difficoltà intrinseca, ovvero l'indentificazione, l'esplorazione e l'organizzazione dei concetti del dominio dell'applicazione, piuttosto che il lavoro di codifica in un linguaggio di programmazione. Quindi la fase di l'identificazione delle classi è molto cruciale, così come capire che comportamento attribuire loro (astrazione, operazioni, interfaccia, ruolo nel contesto del programma) che poi si riflette il modo con cui poi si fanno interagire gli oggetti.

L'approccio orientato agli oggetti per la produzione del software non prende decisioni dettagliate di progetto e di implementazione prima di aver compreso bene il problema e punta ad uno stile di progettazione indipendente dal linguaggio che demanda i dettagli dell'implementazione allo stadio di codifica che è comunque relativamente meccanico.

Come nuovo modo di ragionare sul software il paradigma orientato agli oggetti deve riguardare l'intero processo di ingegneria del software per avere un impatto significativamente positivo sul processo di produzione del software e in ultima analisi sulla qualità del software.

L'approccio orientato agli oggetti usato nell'analisi, progettazione e sviluppo software è utile, in fase di analisi, per la comprensione dei problemi, e nella comunicazione con gli utenti o committenti del software. In questa fase, l'uso dell'astrazione porta chiarezza concettuale e alla formalizzazione di un vocabolario condiviso riguardo concetti del dominio dell'applicazione fra i progettisti e gli utenti o committenti che evita di incorrere successivamente in problemi strutturali del sistema software che comporterebbero un alto impatto in termini di costi (diretti, ulteriori effort o indiretti, perdita di credibilità con il cliente). In fase di progettazione, è utile per la descrizione di modelli di sull'architettura software generale (progettazione di alto livello) e di modelli sulla soluzione software (progettazione software) con tutti i dettagli necessari all'implementazione (metodi, interfacce) e al lavoro di codifica.

La visione del software object oriented richiede un approccio evolutivo all'ingegneria del software. Infatti, sarebbe troppo difficile definire tutte i tipi di oggetti necessari per un sistema o un prodotto software in una singola interazione. Man mano che i modelli ad oggetti del'analisi e della progettazione evolvono, la necessità per nuove classi diventa chiaro. E' per questa ragione che l'aproccio interativo appena descritto è quello che più si addice all'approccio orientato agli oggetti.

Da quanto detto, gli oggetti (e le relative classi) sono individuati e raffinati attraverso le varie fasi di produzione del software a partire dalla fase di analisi e progettazione (dove si opera all'interno del dominio dell'applicazione) fino all'implementazione del sistema sofware (dove si opera all'interno del dominio della soluzione)1. Questo porta ad avere due diverse espressioni di oggetto a seconda della fase e quindi del dominio in cui si opera.

Nella fase di analisi e progettazione di alto livello si opera all'interno del dominio dell'applicazione. In questo caso per oggetto si intende l'espressione di un concetto concreto o astratto e che abbia un significato semantico strumentale rispetto al problema che stiamo trattando. E' definito tramite caratteristiche generali come il comportamento e gli attributi significativi.

Nella fase di progettazione software si opera all'interno del dominio della soluzione. In questo caso per oggetto si intende un'entità del sistema software che incapsula dati e operazioni su tali dati e conseguentemente un'interfaccia per l'invocazinoe delle operazioni da altri oggetti. E' inteso soprattutto come astrazioni di un dato con le procedure associate per l'accesso dall'esterno in modo da rispondere a richieste di elaborazioni e/o di accesso ai dati (stato).

Ciò che cambia dalla fase di analisi e progettazione è il livello di astrazione (ovvero livello di dettaglio) ma non i fondamenti dell'approccio. Anche la notazione usata è la stessa, ovvero UML, ma usata da due prospettive differenti (ovvero da una prospettiva concettuale oppure software) e quindi con due diverse modalità (UML come abbozzo o UML come progetto, rispettivamente).

In Fig. 6 possiamo vedere dei frammenti dei requisiti che sono stati formulati alla fine di una serie di incontri tra il progettista e il capo dell'azienda. Da qui si può capire quanto la fase di analisi dei requisiti e la chiara formulazione di questi sia di fondamentale importanza. Può meritare spenderci un po' più di tempo pur di averli chiari e procedere fluidamente nella fase di identificazione delle classi e successiva progettazione.

… dovrà essere possibile cercare un cliente ed avere mostrati i dati anagrafici del cliente trovato

… la scheda cliente dovrà mostrare tutti i dati anagrafici ed un elenco di fornitori da cui il cliente ha già acquistato

… su richiesta dell'utente dovrà essere calcolato l'importo complessivo degli ordini fatti dal cliente nell'intervallo di tempo selezionato

… per ciascun ordine dovranno essere mostrati: nome fornitore, nome cliente, linea di appartenenza dei prodotti acquistati, importo complessivo

… il report mensile dovrà contenere per ciascun cliente: la provincia di appartenenza e il totale ordinato per ciascun fornitore

Fig. 7: *Frammenti di requisiti del software*

Si procede all'analisi grammaticale. Come mostrato in Fig. 7, in verde sono evidenziati i nomi e in rosso i verbi. Altri nomi sono stati evidenziati in marrone e come sostantivi sono stati interpretati come potenziali attributi in quanto qualità o informazioni da associare a qualche oggetto. In generale può essere necessario fare una selezione ignorando locuzioni che non hanno significato (es. di appartenenza, mano a mano) o raggruppando casi di omonimia.

*… dovrà essere possibile **cercare** un **cliente** ed avere **mostrati** i dati anagrafici del **cliente** trovato*

*… la **scheda cliente** dovrà **mostrare** tutti i **dati anagrafici** ed un elenco di **fornitori** da cui il **cliente** ha già acquistato*

*… su richiesta dell'utente dovrà essere **calcolato l'importo complessivo** degli **ordini** fatti dal **cliente** nell'intervallo di tempo **selezionato***

*… per ciascun **ordine** dovranno essere **mostrati**: nome **fornitore**, nome **cliente**, **linea di appartenenza** dei **prodotti** acquistati, **importo complessivo***

*… il **report mensile** dovrà contenere per ciascun **cliente**: la **provincia** di appartenenza e il **totale ordinato** per ciascun **fornitore***

***Fig. 8:** Frammenti di requisiti del software con parole chiave evidenziate ai fini dell'identificazione delle classi*

Dopo un processo di analisi del testo si giunge a questa lista di massima:

- Classi (in verde)
  - Cliente, Fornitore, Ordine, Prodotto, ReportMensile, SchedaCliente.
- Attributi (in marrone)
  - Dati anagrafici cliente, nome cliente, provincia cliente.
  - Nome fornitore.
  - Importo ordine.
  - Linea appartenenza prodotti.
- Metodi (in arancio)
  - Cercare un cliente.
  - Mostrare dati anagrafici e fornitori per un cliente.
  - Calcolare totale ordini per un cliente.
  - Selezionare ordini in un intervallo temporale.
  - Calcolare totale ordini per un cliente per ciascun fornitore per mese.

4. Messaggi, classi e istanze

La programmazione orientata agli oggetti (Object-Oriented Programming, OOP) rappresenta un paradigma che si è affermato a partire dagli anni '80 e ha trovato in Java uno dei suoi esempi più significativi e diffusi. A differenza dell'approccio procedurale tradizionale, centrato su funzioni e sequenze di istruzioni, la OOP si basa su entità autonome e comunicanti chiamate oggetti, che rappresentano concetti o entità del mondo reale o astratto.

In questo contesto, gli oggetti non sono semplici strutture dati, ma unità attive di elaborazione, in grado di eseguire operazioni, modificare il proprio stato interno e interagire con altri oggetti. Ogni oggetto incapsula uno stato, definito da variabili dette attributi, e un comportamento, rappresentato dai metodi. L'esecuzione di un programma orientato agli oggetti è quindi vista come un flusso di messaggi: ogni volta che un oggetto desidera che un altro compia un'azione, gli invia un messaggio, che viene gestito eseguendo il metodo corrispondente.

Un messaggio in Java, e più in generale in OOP, è una tripla composta da un riferimento all'oggetto ricevente, il nome del metodo da invocare e un insieme di argomenti o parametri. Quando un oggetto riceve un messaggio, cerca all'interno della propria definizione (cioè nella sua classe) il metodo che corrisponde a quel nome e con quella firma (tipi e numero di parametri) e lo esegue.

Il comportamento degli oggetti, quindi, non è definito in modo arbitrario per ciascun oggetto, ma è regolato dalla classe a cui l'oggetto appartiene. In Java, una classe è una struttura astratta che definisce sia lo stato possibile degli oggetti (cioè quali variabili avranno) sia le operazioni che essi possono eseguire (cioè i metodi disponibili). Le classi sono, per usare un'analogia comune, come progetti o stampi: da esse si possono generare istanze, cioè oggetti concreti che possiedono i dati propri (unici) ma condividono struttura e comportamento con altri oggetti della stessa classe.

È fondamentale distinguere quindi tra classe e istanza. La classe è una definizione, un modello: non occupa memoria per rappresentare dati di singoli oggetti, ma definisce la forma e le funzionalità comuni. Un'istanza è invece un oggetto reale, creato in memoria, con un proprio stato distinto. Ad esempio, una classe Automobile può definire che ogni auto ha una targa, un colore e un metodo accelera(). Le istanze della classe saranno oggetti come auto1 o auto2, ciascuna con la sua targa e il suo colore, ma entrambe potranno invocare accelera().

Descrivere oggetti uno per uno sarebbe inefficiente e controproducente: ogni oggetto richiederebbe una descrizione dettagliata e separata, rendendo il codice ridondante, difficile da mantenere e poco scalabile. Per questo motivo, si usano le classi come strutture generali e riutilizzabili, che

permettono la creazione di oggetti coerenti e la gestione sistematica del comportamento del sistema software. In particolare, in Java le classi sono fortemente tipizzate, e l'associazione tra oggetti e classi è rigida e ben definita dal compilatore.

L'approccio orientato agli oggetti favorisce inoltre l'incapsulamento, cioè la possibilità di nascondere l'implementazione interna di un oggetto e di esporre solo un'interfaccia pubblica attraverso cui gli altri oggetti possono interagire. Questo principio aumenta la modularità, la manutenibilità e la robustezza del software, permettendo agli sviluppatori di concentrarsi sull'interazione tra oggetti piuttosto che sui dettagli di implementazione interna.

Infine, la programmazione a oggetti rende naturale la modellazione di sistemi complessi, perché permette di rappresentare le entità del dominio applicativo come oggetti con ruoli e comportamenti ben precisi. In Java, tutto ruota intorno agli oggetti e alle classi: anche elementi fondamentali del linguaggio, come le strutture dati, le interfacce grafiche o la comunicazione in rete, sono costruiti su questo paradigma.

Bibliografia

- Il nuovo Java,, Claudio De Sio Cesari, Hoepli Informatica
- https://www.oracle.com/java/technologies/downloads/