

Formulario di

Programmazione 1

Rosario Terranova


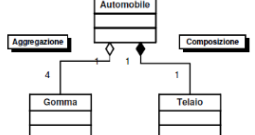
v 1.1.4

Dati primitivi

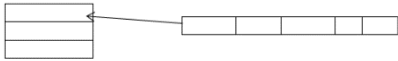
Programma Java	Un programma in Java è una sequenza di istruzioni che rientrano in una sintassi predefinita. Un'istruzione è una piccola unità d'operazione in Java. Le istruzioni terminano con un punto e virgola; sono organizzati in blocchi e delimitati dalle parentesi graffe {...}.				
Codice Java	<pre>public class Esercizio { public static void main (String [] args) { ; } }</pre>		<ul style="list-style-type: none">- Tutto in java è dentro una classe.- Il nome del programma deve essere uguale al nome con cui si salva il codice (*.java).- Un programma inizia sempre dal metodo main dove metteremo tutte le istruzioni del programma.- Le parentesi graffe contengono i blocchi dell'istruzione.- Ogni istruzione termina con un punto e virgola.		
Commenti	// ... /* ... */ /** ... */	commenti di riga commenti lunghi, anche su più righe commenti di documentazione		Operatori	+ somma – differenza * prodotto /divisione % resto
Tipi di dati primitivi (identificatori)	boolean	1 bit	true, false	<div>Dichiarazione di una variabile</div> <div>int x = 4;</div> <div>Tipo di dato Nome della variabile Valore della variabile</div>	
	char	16 bits	caratteri Unicode		
	byte	8 bits	[-128, 127]		
	short	16 bits	[-32768, 32767]		
	int	32 bits	[-2147483648, 2147483647]		
	long	64 bits	[-9223372036854775808, 9223372036854775807]		
	float	32 bits	[-3.4E38, 3.4E38]		
	double	64 bits	[-1.7E308, 1.7E308]		
Caratteri speciali della stampa su console System.out.print()	+	concatenazione		parole riservate	<div>abstract extends long synchronized</div> <div>boolean false native this</div> <div>break final new throw</div> <div>byte finally null throws</div> <div>byvalue float operator transient</div> <div>case for outer true</div> <div>cast future package try</div> <div>catch generic private var</div> <div>char goto protected void</div> <div>class if public volatile</div> <div>const implements rest while</div> <div>continue import return</div> <div>default inner short</div> <div>do instanceof static</div> <div>double int super</div> <div>else interface switch</div>
	'\b'	backspace			
	'\n'	salto riga			
	'\t'	spaziatura per tabella			
	'\''	apice '			
	'\"'	doppio apice "			
	'\\'	\			
	println	produce una stampa e a capo			
Costanti	final int MAX = 100 (il valore non può essere modificato)				
Funzioni Matematiche	double x = Math.sqrt (x)		radice		
	double x = Math.sin (x)		seno		
	double x = Math.cos (x)		coseno		
	double x = Math.tan (x)		tangente		
	double x = Math.pow (x,y)		potenza (x = base, y = potenza)		
	double e = Math.E		numero di nepero		
	double π = Math.PI		pi greco		
	double x = Math.abs (x)		valore assoluto		
	Math.random()		numero casuale 0.0 ≤ x ≤ 0.9 (restituisce un double)		
			numeri casuali da 1 a 8		int x = (int)(8*Math.random()+1)
			caratteri casuali a-z		char c = (char)((100*Math.random())/4+97);
Operatori predefiniti di assegnamento	Assegnamento		x = 1+2		
	Assegnamento con operazione		x += y vuol dire x = x+y		
	Confronto		5 == 2+3; //true		
	Incremento e decremento prefisso		++x; --x; // la variabile viene prima incrementata		
	Incremento e decremento postfisso		x++; x--; // la variabile viene incrementata dopo le operazioni		
Dichiarazioni di caratteri	Per dichiarare un char usiamo indicarlo dentro gli apostrofi char chr = 'q';		Possiamo indicare con tale notazione anche un carattere unicode char chr = '\123';		
Operazioni aritmetiche	Le operazioni in java sono a 32bits, se quindi si vuole calcolare: byte a, b=1, c=2 dobbiamo necessariamente applicare un casting a = (byte) (b+c)		In caso di superamento del limite del tipo accade che: int i = 2147483647, j = i+1; In output avremo j = -2147483648		
Operatori logici	!not	x(2!=3) restituisce true			
	&& and	A && B è vera se entrambi A e B sono veri, altrimenti è falsa		VV=V VF=F FV=F FF=F	&&
	or	A B è falsa se entrambi A e B sono falsi, altrimenti è vera		VV=V VF=V FV=V FF=F	
Promozione e conversione di identificatori	<div><div>byte char</div><div>↓ ↓</div><div>short int long</div><div>↓ ↓ ↓</div><div>float double</div></div> <div>Avviene automaticamente quando non si ha perdita di informazione</div>				
Casting	(tipo) (espressione);		conversione forzata con perdita di informazione e con troncamento dei valori		

Gestione dell'input/output (libreria javax.swing)	import javax.swing.*		importazione della classe input/output
	JOptionPane.showMessageDialog (null, "Messaggio", "TitoloFinestra", JOptionPane.PLAIN_MESSAGE);		crea una finestra con un messaggio
	String x = JOptionPane.showInputDialog("Input");		assegnare alla variabile x una stringa presa in input
	y = Integer.parseInt (x);		convertire la stringa x in un intero y
	y = Double.parseDouble(x);		convertire la stringa x in un double y
	int x = JOptionPane.showConfirmDialog (null, "domanda?", "finestra", JOptionPane.YES_NO_CANCEL_OPTION);		crea una finestra di conferma, da integrare con: if (x == JOptionPane.YES_OPTION) JOptionPane.showMessageDialog(null, "hai detto si!");
	System.exit(0);		uscire da un ciclo di messaggi
Costrutto di selezione if-then-else	if (condizione) { istruzione1;		

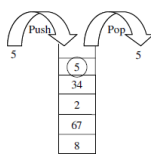
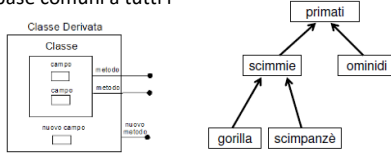
Dati non primitivi

Oggetto	Stato	serie di attributi o proprietà (Es. targa, colore, velocità)		
	Comportamento	insieme di operazioni che prendono il nome di metodi o operazioni (Es. che colore?; qual è la targa?; accelera; spegni motore)		
	Classe	descrizione di come devono essere costruiti gli oggetti e le azioni che possono compiere		
	Istanza	l'oggetto creato è un'istanza, ovvero un esemplare di una classe		
Creazioni di oggetti	oggetto obj1, obj2; obj1 = new oggetto(); obj2 = new oggetto();	<u>Principio di identità</u> ciascun oggetto può essere trattato distintamente da un altro <u>Principio di conservazione dello stato</u> durante l'esecuzione del programma gli oggetti mantengono le informazioni al proprio interno per un tempo indefinito		
	oggetto obj1 = new oggetto(); oggetto obj2 = new oggetto();			
Messaggi tra oggetti	gli oggetti scambiano messaggi tra loro modificando il loro comportamento; i messaggi che si possono spedire sono quelli relativi ad un metodo <i>pubblico</i> . <nomeoggetto>.<metodo>(<parametri>);		<i>Tipi di messaggi</i>	Informativo Interrogativo Imperativo
	Es. Auto opel = new Auto(...); opel.setTarga(XXX-XXX-XXX);			setTarga() getTarga() avvia()
Metodi che restituiscono valori	i metodi possono o meno restituire valori; il tipo di identificatore restituito deve essere dichiarato nella classe d'appartenenza.			
	getUltimoLancio(): byte;	public byte getUltimoLancio(){...}	vuol dire che il metodo getUltimoLancio() restituisce un byte	
	effettuaLancio(): void	public void effettuaLancio(){...}	vuol dire che il metodo effettuaLancio() restituisce un void (vuoto)	
Metodo costruttore	è un metodo che ha lo stesso nome della classe, e che viene sempre inviato all'oggetto appena creato al momento dell'istanziamento. Serve ad inizializzare lo stato e le variabili di istanza dell'oggetto appena creato. Il compilatore inserisce automaticamente la chiamata del metodo costruttore subito dopo la corrispondente new. Es. Dado () Moneta () , public Dato(){...} public Moneta(){...} Esso può contenere anche i parametri che verranno presi dal main.			// Es. <pre>class Motore { private boolean acceso; public Motore (boolean _acceso) { acceso = _acceso; } }</pre>
	L'Unified Modelling Language serve a rappresentare le classi, i vari metodi e i collegamenti nella OOP (object oriented programming).			
Notazione UML	Rappresentazione di una classe	Classe - variabili di stato + Costruttore() + Metodo():tipoRest	Gli attributi hanno una struttura del tipo: <i>visibilità nome (parametri): tipoRestituito;</i> i parametri a loro volta possono contenere dei valori da prendere Es. + setName(nome: String): void che in java vuol dire: public void setName(String nome){...}	
	Modificatori d'accesso	+ public - private # protected	<u>static</u> <i>abstract</i> <<abstract>>	Es. - attributoPrivato: int + <<attributoPubblicoAstratto: int>> #metodoProtected: void
	Ereditarietà	 il verso della freccia indica da sottoclasse a superclasse		
	Associazione	è la possibilità che ha un'istanza di inviare un messaggio ad un'altra istanza. Es. <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="border: 1px solid black; padding: 5px; text-align: center;"> Automobile - motore: Motore + accendi: void </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> Motore - Candele: boolean + accendiCandele: void </div> </div> <pre> public class Automobile{ private Motore motore; public void accendi(){ motore.accendiCandele() } } public class Motore{ private boolean Candele = false; public void accendiCandele(){ Candele = true; } } </pre>		
	Aggregazione	consiste nella creazione di una classe a parte, la quale rappresenta un'associazione uno a molti, in cui un oggetto di una classe diventa parte di un'oggetto dell'altra. In caso di <i>composizione</i> , la parte appartiene ad un solo oggetto, e nasce e muore insieme all'oggetto che la contiene.		
				
Esempio di classe ed esecuzione	Serbatoio Stato Serbatoio (capacità:float;quantità:float); getCapacità():float; getQuantità():float; deposita(litri:float):void; preleva(litri:float):float; svuotaTutto():float; riempiTutto():void; pieno():boolean; vuoto():boolean; piùDiMetà():boolean;		public class TestSerbatoio { public static void main(String[] args) { Serbatoio s = new Serbatoio(10.0,7.0); float inUscita = s.preleva(3); if (s.piuDiMeta()) System.out.println("Abbondante!"); else System.out.println("Scarseggia!"); inUscita += s.svuotaTutto(); s.deposita(8); } }	
	Mentre il metodo costruttore con i suoi parametri non restituisce nulla se non lo stato dell'oggetto, i metodi normali oltre al potere avere dei parametri possono anche restituire valori.			

Oggetti come parametri	Freccia	Bersaglio	Bersaglio b = new bersaglio (10); Freccia f = new Freccia()
	Stato	Stato	f.lancia(b);
	+ Freccia() + lancia(B: bersaglio): void + getX(): float + getY(): float	+ Bersaglio(dimensione: int) + punteggio(f:Freccia): byte	totale += b.punteggio(f)
Metodi che restituiscono oggetti	Frazione	//Dunque per calcolare $x = \frac{6}{5} - \frac{3}{4}$ basta scrivere:	
	Stato	Frazione x; Frazione f1 = new Frazione (6,5); Frazione f2 = new Frazione (3,4);	
	+ Frazione(num: int ; den: int) + numeratore(): int + denominatore(): int + più(f: Frazione): Frazione + meno(f: Frazione): Frazione	x = f1.meno(f2);	
Array di oggetti	Attenzione: la new necessaria per la creazione dell'array NON crea gli oggetti relativi alle singole componenti dell'array		Bersaglio b = new Bersaglio (10); Freccia F[] = new Freccia [20];
	Gli array possono essere passati come parametro ai metodi: int [] b = new int [30]; ClasseA obj = new ClasseA(); obj.metodo(b);		int tot = 0, i = 0;
	Gli array possono essere restituiti dai metodi: int [] b; ClasseA obj = new ClasseA(); b = obj.altroMetodo();		while (i<F.length){ //riempimento array con frecce F[i]= new Freccia(); F[i].lancia(b); i++; }
Costruzione di classi	La classe deve implementare: <ul style="list-style-type: none">Stato (variabili di istanza)Comportamento (metodi) Quando un metodo viene invocato un metodo il <i>flusso di controllo</i> passa a tale metodo ed il codice relativo viene eseguito; Quindi, il flusso ritorna al punto da cui è partita la chiamata e continua da lì. Le <i>variabili di stato</i> sono dei valori vuoti dell'oggetto che si andranno a definire applicando i metodi.		class A { //sezione delle variabili di stato <ModificatoreAccesso><TipoVar><NomeVar>; //sezione dei metodi <ModificatoreAccesso><TipoRestituito> <NomeMetodo> (<Parametri>); { ... //corpo del metodo } ...//altri metodi } // fine classe A
	Moneta	Dado	class Dado { private byte valoreUltimoLancio; public void effettuaLancio() { ...//corpodelmetodo } ...//altrimetodi }
	- ultimaFaccia: char	- valoreUltimoLancio: byte	
	+ Moneta() + effettuaLancio(): void + testa(): boolean + croce(): boolean + getFaccia(): char	+ Dado() +effettuaLancio(): void + getUltimoLancio(): byte + commentaLancio(): void	
Riferimenti (puntatori)	i riferimenti contengono gli indirizzi di memoria allocata dove sono contenuti gli oggetti veri e propri.		Codice elenco delle istruzioni che andremo ad eseguire
	La memoria che viene assegnata alla nostra applicazione quando la lanciamo è divisa in 4: →		Stack gestisce le chiamate dei metodi
			Heap zona di memoria utilizzata per allocare gli oggetti
Costruzione di metodi	<ModificatoreAccesso><TipoRestituito><NomeMetodo>(<Parametri>); { ... //corpo del metodo }		// Metodo che fa il cubo di un numero import javax.swing.*; public class MetodoCubo { public static int cubo(int a) { a = a*a*a; return a; } //test public static void main(String[] args) { String NumInput = JOptionPane.showInputDialog("Inserisci un numero"); int x = Integer.parseInt(NumInput); int c = cubo(x); JOptionPane.showMessageDialog(null,"il cubo di "+x+" è "+c); } }
	La lista dei parametri formali specifica il tipo e il nome di ciascun parametro che prende il metodo		
	Es. public float converti (byte codiceValuta, float valoreValuta, boolean commissione) static , se presente, indica che i metodi possono essere richiamati senza istanziare oggetti, usando il nome della classe.		
Istruzione return	Il tipo restituito da un metodo indica il tipo del valore che il metodo restituisce al chiamante <ul style="list-style-type: none">Void: nessun valore restituitoreturn <Espressione>; //restituisce un valore desiderato L'espressione specificata dal return deve essere conforme alla specifica di tipo chiamata nell'intestazione.		class Serbatoio { private float capacitaTotale; private float capacitaPresente; private float getCapacita() { return capacitaTotale; } }
	Possiamo anche dare da return un identificatore, basta che sia lo stesso tipo di quello dichiarato dal metodo.		public int Metodo (int a) { return int; }
	Dopo che viene utilizzato il return, il metodo termina e si uscirà da esso.		public int[] trova (int Array[]) { return array; }
Parametri	I parametri nell'intestazione di un metodo si dicono parametri formali . Nella chiamata di un metodo si dicono parametri attuali .	il passaggio di parametri può avvenire anche per valore (con variabili); i parametri attuali all'atto sono copiati nei parametri formali se c'è corrispondenza tra codice e metodo.	

Visibilità (o scope)	In Java un qualunque <i>blocco</i> {...} può contenere dichiarazioni di variabili al suo interno, tali variabili si dicono <i>locali</i> al blocco. Una variabile locale ad un blocco – è <i>visibile</i> solamente nel blocco in cui è definita, e in ogni blocco contenuto nel blocco. – ha il seguente <i>ciclo di vita</i> : viene creata ogni volta che si entra nel blocco e viene distrutta quando si esce dal blocco		
This	Il this fa riferimento all'istanza corrente di una classe. Esso ha 3 utilizzi: - Se in un metodo ho definito una variabile locale con lo stesso nome di una variabile d'istanza definita nella classe allora posso far riferimento alla variabile della classe mediante <i>this.nomeVariabile</i> . - Se devo passare ad un metodo l'oggetto stesso allora utilizzo <i>istanza.nomeMetodo(this)</i> . - Se invece voglio indicare un altro metodo costruttore risparmiandomi la ripetizione delle istanze di esso, posso usare il <i>this()</i> come prima istruzione.		<pre>class Esercizi_this{ int m; char r; public Esercizi_this(int m, char r){ this.m=m; this.r=r; } }</pre> <pre>public Terna(int a, int b, int c){ int array[] = {a, b, c}; xyz = array; } public Terna(int a[]){ this(a[0],a[1],a[2]); //richiama il costruttore precedente }</pre>
Modificatori d'accesso	public	membro visibile da tutti gli oggetti indipendentemente dalla classe (a patto che anche la classe sia public)	Nel caso la classe sia public e abbia variabili di stato pubbliche, esse possono essere chiamate tramite la seguente notazione: classe.variabile; opel.targa;
	private	membro visibile solo dagli oggetti della medesima classe. Tuttavia è possibile accedervi creando dei metodi set() o get()	
	protected	membro visibile solo alle classi derivate	
	non qualificato	visibili solo da oggetti dello stesso “package”	
Overloading di metodi	se inseriamo in una classe: <pre>double doppio (int x) { return 2*x; } double doppio (double x){ return x*x; }</pre> non ci sarebbe nessun problema: il metodo si comporteranno diversamente a seconda del tipo di parametri ricevuto in input		L' overloading di funzioni avviene quando si usa lo stesso nome per indicare funzioni diverse. Il compilatore segue delle regole precise per la determinazione della funzione della famiglia di overloading. In particolare: • verifica se esiste un accoppiamento esatto, ed in caso contrario • tenta le conversioni automatiche di tipo. <pre>class Serbatoio { private float capacitaTotale; private float quantitaPresente; public Serbatoio (float _capacita) { capacitaTotale = _capacita; quantitaPresente = 0f; } public Serbatoio (float _capacita, float _quantita) { capacitaTotale = _capacita; quantitaPresente = _quantita; } ... //altri metodi } //End class Serbatoio</pre> Può avvenire anche l'overloading di costruttori per poter specificare differenti informazioni al momento dell'istanziamento (le informazioni non specificate saranno sostituite all'interno del costruttore con dei valori di default stabiliti dal progettista della classe), e per poter specificare le informazioni utilizzando differenti formati
	ATTENZIONE: il metodo in overloading deve sempre restituire lo stesso tipo di dato		
Static	gli attributi di una classe possono essere: - <u>variabili di istanza</u> : attributi che cambiano valore da istanza ad istanza; - <u>variabili di classe (o statiche)</u> : tutti gli oggetti della classe condividono lo stesso valore. Ciò è di grande importanza per una implementazione efficiente dentro la RAM. Dichiarando delle variabili con lo static essi faranno già riferimento alla memoria, mentre quelle di istanza lo faranno solo con l'istanziamento		Es. <pre>public class Dadi{ static int primo; static int secondo; } //MAIN println(Dadi.primo) println(Dadi.secondo)</pre>
	I metodi possono essere dichiarati di tipo static (metodi di classe) se la loro esecuzione non dipende dall'oggetto destinatario del messaggio. Nel corpo dei metodi static si possono referenziare solo variabili static e non di istanza.		/*questi campi statici esistono già nella classe, non sono come quelli dinamici che richiedono l'istanziamento. Possiamo anche creare dei metodi statici che possono essere accessibili anche senza creare l'istanza, ma solo usando variabili statiche*/
	Il blocco static è come il metodo costruttore, e serve ad inizializzare i campi statici. Es. static {primo =1; secondo=2;}		
Array Frastagliati	 Gli array multidimensionali sono praticamente degli array monodimensionali che si riferiscono ad altri array monodimensionali Il riferimento può quindi avere array di lunghezza diversa, quindi <i>array frastagliati</i> .		
	Dichiarazione Array Frastagliato	int [][] tabella = new int [3][];	array di 3 righe e colonne indefinite
		tabella [0] = new int [2];	la prima riga ha 2 colonne
		tabella [1] = new int [3];	la seconda riga ha 3 colonne
		tabella [2] = new int [1];	...
	Numero di righe	tabella.length	
Numero di colonne	non possiamo sapere il numero di colonne quindi per riempire automaticamente l'array mettiamo: for (int i=0;i<tabella.length;i++) for (int j=0;j<tabella[i].length;j++) tabella[i][j] = //fai qualcosa		

Concetti avanzati della OOP

Algoritmi Notevoli	<pre>class Ricerca_Numero_Massimo { public int searchFirstMax(int array[]){ int tmp = 0; for (int i=0; i<array.length;i++){ if (array[i]>tmp) tmp = array[i]; } return tmp; } }</pre>	<pre>class Ricerca_Secondo_Numero_Massimo { public static int searchSecondMax(int array[]){ int tmp = 0, x = 0; for (int i=0; i<array.length;i++){ if (array[i]>tmp) { x = tmp; tmp = array[i]; } } return x; } }</pre>	<pre>public class Ricerca_Binaria{ int binarySearch(int array[], int key) { int low = 0; // limite sinistro int high = array.length - 1; // limite destro int middle; // centro while (low <= high) { middle = (low + high) / 2; if (key == array[middle]) // trovato! return middle; else if (key < array[middle]) high = middle - 1; // cerca nella 1 metà else low = middle + 1; //cerca nella 2 metà } return -1; // non trovato! } }</pre>
	<p>La ricerca binaria (o dicotomica) in un array ordinato consiste nel confrontare l'elemento centrale M dell'array con la chiave di ricerca e</p> <ul style="list-style-type: none">- se sono uguali, l'elemento è stato trovato- se la chiave di ricerca è <M, essa prosegue nella 1° metà dell'array- se la chiave di ricerca è >M, essa prosegue nella 2° metà dell'array <p>si fanno \log_n confronti</p>		
Algoritmi di ordinamento	<p>Swap di due elementi di un array</p> <pre>static void swap (int a[], int primo, int secondo) { int tmp; tmp = a[primo]; a[primo] = a[secondo]; a[secondo] = tmp; }</pre>	<p>BubbleSort: si controllano i valori dell'array diverse volte a due a due; se il primo valore è maggiore del secondo i due elementi vengono scambiati di posto, ripetendo il processo finché l'array è ordinato</p>	<pre>static void bubbleSort (int array[]) { for (int pass = 1; pass < array.length; pass++) for (int i = 0; i < array.length-1; i++) if (array[i] > array[i+1]) swap (array, i, i + 1); }</pre>
	<p>SelectionSort: ricerca il più piccolo valore dell'array e lo scambia con l'elemento in prima posizione, poi cerca il secondo e così via</p> <pre>public static void selectionSort (int[] array) { int min; for (int index = 0; index < array.length-1; index++) { min = index; for (int i = index+1; i < array.length; i++) if (array[i] < array[min]) min = i; swap(array, min, index); } }</pre>	<p>InsertionSort: ogni elemento dell'array viene spostato in un sotto-array che viene ordinato, facendo poi shiftare il processo al numero successivo</p> <pre>public static void insertionSort (int[] array) { for (int index = 1; index < array.length; index++) { int key = array[index]; int position = index; // shifta i valori più grandi di key a destra while (position > 0 && array[position-1] > key) { array[position] = array[position-1]; position--; } array[position] = key; } }</pre>	
Ricorsione	<p>si basa sul <i>principio di induzione</i>: per dimostrare una preposizione si dimostra il caso base, e poi bisogna dimostrare se la proprietà vale per n, e se vale anche per n+1 essa vale per qualsiasi n. Una funzione si dice <i>ricorsiva</i> se è definita in termini di se stessa. Es: fattoriale che ha 1 se $n = 0$ e $n \cdot (n - 1)!$ se $n > 0$</p>		
	<p>È come un ciclo, sono metodi che richiamano se stessi più volte fino al caso base, per poi ritornare qualcosa dall'ultima chiamata alla penultima, alla terz'ultima, ecc.</p> <p>Es. in matematica una funzione ricorsiva è: $n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n - 1)! & \text{se } n > 0 \end{cases}$</p>		<p>//esempio ricorsività</p> <pre>int fattoriale (int n) { if (n == 0) // caso base return 1 else return //passo induttivo n*fattoriale(n-1); }</pre>
	<p>Ogni definizione ricorsiva è caratterizzata dal:</p> <ul style="list-style-type: none">- Caso base (condizione di terminazione): la condizione per cui la funzione termina, cioè smette di richiamare se stessa. Se non ci fosse, la funzione sarebbe un loop infinito.- Passo induttivo (chiamata ricorsiva): la soluzione ottenuta viene combinata con altra informazione per produrre la soluzione al problema originale.		
Stack (o pila)	<p>Struttura dati LIFO (Last-In First-Out, ovvero l'ultimo ad entrare è il primo ad uscire), in cui le due sole operazioni ammesse per modificare i dati sono:</p> <ul style="list-style-type: none">- Push (inserimento)- Pop (estrazione)		
	<p>La ricorsività fa uso di uno <i>stack delle chiamate</i>. Ad ogni chiamata di funzione viene creato un <i>record di attivazione</i>, che viene inserito nello stack. In particolare, ogni record di attivazione contiene</p> <ul style="list-style-type: none">- il <i>punto di ritorno</i> alla funzione chiamante- i <i>parametri formali</i> della funzione chiamata (in cui sono già stati copiati i parametri attuali)- le <i>variabili locali</i> alla funzione chiamata- i valori dei <i>registri</i> della CPU		
Eliminazione della ricorsione	<p>Qualsiasi funzione <i>ricorsiva</i> può essere espressa in forma <i>non ricorsiva</i>.</p> <ul style="list-style-type: none">- Se la ricorsione è di <i>coda</i> (FIFO = il primo ad entrare è il primo ad uscire), cioè la chiamata ricorsiva è l'ultima azione eseguita dalla funzione, allora la ricorsione può essere eliminata con una semplice iterazione.- Se la ricorsione è <i>non di coda</i>, la ricorsione può essere eliminata solo con l'ausilio di uno stack esterno. <p>In genere, tutte le funzioni con una <i>doppia chiamata ricorsiva</i> non possono essere riscritte con una banale iterazione.</p>		<p>//eliminazione ricorsione di coda</p> <pre>int fattoriale (int n){ int f = 1 while (n>0) f*=n--; return f; }</pre>
Ereditarietà	<p>Una classe B eredita la classe A se assume tutti i suoi dati e metodi e gli estende, modifica o migliora ulteriormente. Es. classe A = primati (classe base o superclasse) classe B = uomo (classe derivata o sottoclasse)</p> <p>Una caratteristica di una superclasse si può trovare anche nella sottoclasse, ma non il contrario.</p> <ul style="list-style-type: none">- SuperClasse: classe generale, molto ampia, con alcune caratteristiche di base comuni a tutti i suoi componenti;- SottoClasse: classe più particolare, di minor estensione, con numerose caratteristiche proprie solo a tutti i suoi componenti. <p>Le classi dove sono presenti il maggior numero di caratteristiche sono le sottoclassi.</p> <p>Ogni sottoclasse può a sua volta essere la superclasse di una sua sottoclasse.</p>		
	<p>Regola ISA (is a, "è un"): un elemento della sottoclasse deve essere anche un elemento della superclasse (Es. una scimmia è anche un primato, uno scimpanzé è anche una scimmia, se questo non vale c'è un errore nella relazione di ereditarietà)</p>		
Extends	<p>scrivendo extends in una sottoclasse possiamo utilizzare tutte le variabili e i metodi della superclasse</p> <p>Es. class triangolo {...} class isoscele extends triangolo {...} class equilatero extends isoscele {...}</p> <p>Con extends, una sottoclasse eredita tutti gli attributi e i metodi della sua superclasse.</p>		

Overriding	tutti i discendenti ereditano gli attributi e i metodi dei genitori. Non è quindi necessario ricreare i metodi, a meno che non siano diversi dalla classe madre: in questo caso è comodo sovra-scrivere i metodi che possono essere migliorati in casi speciali. Per farlo è sufficiente ripetere la dichiarazione del metodo di un progenitore e farla seguire da un nuovo corpo.	Es. class triangolo{ ... public double perimetro(){ return lato1+lato2+lato3; } }	class isoscele extends triangolo{ ... public double perimetro(){ return 2*lato1+lato3; } }	
	ATTENZIONE: per sovrascrivere un metodo è necessario che i parametri e il valore ritornato del metodo sovrascritto siano gli stessi di quello della superclasse			
	Eccezioni: una classe final non può essere estesa; un metodo final non può essere sovrascritto; una variabile final non può cambiare valore.			
Super	viene usato per indicare i metodi della classe progenitrice (in caso di overriding) o i costruttori in due maniere: - <i>super(<parametri>)</i> passa i parametri del costruttore della sottoclasse al costruttore della super classe, la quale gli elaborerà poi col suo costruttore; - <i>super.metodo()</i> indica ad una sottoclasse di utilizzare un metodo della superclasse.	Es. class triangolo { double lato1, lato2, lato3; public triangolo(double A, double B, double C) { //costruttore lato1=A; lato2=B; lato3=C; } ... //altri metodi	class isoscele extends triangolo { public isoscele(double A, double B) { //costruttore di isoscele super(A,A,B); // finito! } ... //altri metodi, anche overriding se necessario	
	A differenza dei metodi, i costruttori NON vengono ereditati. Ogni costruttore della sottoclasse chiama implicitamente un costruttore della superclasse con il <i>super()</i> se il metodo della sottoclasse non è implementato	Nessun bisogno di ripetere di nuovo la dichiarazione delle variabili d'istanza: esse sono ereditate da triangolo		
Compatibilità dei tipi	il riferimento ad un tipo di oggetto non può essere ad un oggetto di classe diversa. In Java è possibile che un riferimento relativo alla classe di un progenitore punti ad un discendente. Non è possibile il viceversa. Es. triangolo A = new isoscele (10,30) è corretto isoscele A = new triangolo (10,30,22) è errato Data una super classe A e una sotto classe B, se B ha uno stesso metodo di A ma sovrascritto, al momento della chiamata del metodo di una istanza di B verrà chiamato sempre l'ultima definizione del metodo, quindi il metodo sovrascritto.			
Polimorfismo	si riferisce al fatto che una espressione il cui tipo sia descritto da una classe A può assumere valori di un qualunque tipo descritto da una classe B sottoclasse di A (<i>polimorfismo per inclusione</i>). Gli oggetti agiscono in base a ciò che sono; possono assumere diverse “forme” dentro la propria gerarchia. Se invece si dichiara una variabile B come isoscele e poi si assegna un triangolo che non ha due dei suoi lati di eguale misura è una bugia! E non passa inosservata all'occhio vigile del compilatore!			
Casting degli oggetti derivati	Un oggetto di una sottoclasse può essere trasformato in un oggetto di una superclasse, non si perdono informazioni. Es. quadrato A = new quadrato(20); rettangolo B = A;	Possiamo creare un array di oggetti di superclassi ed instanziarli poi come sotto classi. Es. A a = new A[30]; int casuale = 0; for (int i =0; i<a.length;i++){ casuale = (int)(Math.random(3+1); if (casuale == 1) a[i] = new B(...); else if (casuale ==2) a[i] = new C(...); else if (casuale ==3) a[i] = new D(...); }		
	Un oggetto di una superclasse non può essere sempre trasformato in un oggetto della sottoclasse, possibile perdita di informazioni. Es. rettangolo A = new rettangolo(20,23); quadrato B= (quadrato) A;			
	Dato un'array di sottoclassi, posso invocare un metodo comune se esso è stato dichiarato in tutte le sottoclassi! Es. a[i].getInfo(); Nella superclasse può anche essere astratto			
getClass()	il metodo getClass() è un metodo predefinito di tutti gli oggetti di Java che ci indica di qualche classe appartiene un oggetto Es. rettangolo A = new rettangolo (12,13); System.out.println(“classe di A: ”+A.getClass());			
instanceof	il quale produrrà all'esecuzione: classe di A : class rettangolo			
	è un operatore predefinito di Java. Ha la forma: Nome_Oggetto instanceof Nome_Classe; esso restituisce un boolean, con true se l'oggetto appartiene alla classe, e false se l'oggetto non appartiene alla classe			
Gerarchia di metodi	creando un ciclo for, vogliamo calcolare l'area dei quadrilateri conservati nell'array A di oggetti misti tra quadrati e rettangoli. Il comando dunque sarà: A[i].area(); Non è chiaro se però viene invocato il metodo della classe rettangolo o quadrato. La JVM opera con la tecnica del “Binding Dinamico”: il codice di un metodo viene legato agli oggetti nel momento della esecuzione dalla JVM. Questo consente di chiamare per i rettangoli il metodo della sua classe, e stessa cosa per i quadrati.			
Classe cosmica Object	JAVA prevede una classe dalla quale TUTTE le altre classi vengono derivate: Object . Ogni altra classe deriva, entro una qualche gerarchia, da Object . Se una classe viene definita senza essere esplicitamente derivata da altre, essa è implicitamente da considerarsi un'estensione della classe Object . Object è una classe a tutti gli effetti e possiede numerosi metodi importanti che sono ereditati da TUTTI gli oggetti.	Es. //Scrivere Class Demo{...} //equivale a scrivere Class Demo extends java.lang.Object {...}		
Alcuni metodi di Object	a.equals(b)	serve per confrontare due oggetti, spesso usato per le stringhe se hanno gli stessi caratteri		
	toString()	viene chiamato ogni volta che si fa una print di qualcosa; in particolare per gli oggetti stampa il nome della classe, una @ e il riferimento di memoria dell'istanza dell'oggetto		
Overriding dei metodi di Object	Volendo possiamo sovrascrivere i metodi della classe Object con un overriding e mettere altre informazioni a noi utili Es. sostituiamo l'equals confrontando tutti i giocatori di una squadra, ecc. o sostituiamo il toString facendo stampare ordinatamente il nome dell'oggetto e i suoi parametri			

Classi e metodi astratti abstract public <tipo>	Java consente al programmatore di creare una superclasse astratta, una classe non definita con dei metodi non definiti, ma a patto che che poi tali metodi siano dichiarati con tutto il body in tutte le sottoclassi.		Es. abstract class poligono{ int lati; public poligono (int lati){ this.lati = lati; } abstract public int perimetro(); }	class quadrato extends poligono{ int lati; public quadrat(int lati){ super (lati) } public int perimetro(){ int x = lato*lati; return x } }
	Una casse abstract può avere metodi di due tipi: <ul style="list-style-type: none">- Metodi standard: perfettamente definiti e implementati- Metodi abstract: fatta solo con la intestazione (nome, tipo restituito, parametri) ma senza body			
	Se le classi e i metodi abstract non sono poi dichiarati con il body nelle sottoclassi, il compilatore darà errore!			
Ereditarietà multipla	JAVA ammette solo ereditarietà SINGOLA . Questo è un limite che i creatori di JAVA si sono imposti per semplificare il lavoro di javac. Il C++ ammette ereditarietà multipla. Le “interfacce” di JAVA ammettono anche ereditarietà multipla.			
java.util.Random	il comando <i>import java.util.Random;</i> viene utilizzato, come il Math.Random, per generare valori casuali. Dichiarazione: <i>Random r = new Random(seed);</i>			Es. /*Si vuole ottenere un valore intero tra 1 e 10 (entrambi inclusi)*/