JAVA

Fra le sue principali caratteristiche sono presenti:

-Linguaggio Object Oriented

-è più semplice

-è indipendente dalla piattaforma(cioè è indipendente dall'architettura)

-robusto

-concorrente

-sicuro

Il java è un linguaggio di programmazione che sposa completamente il paradigma OO.

Si tratta di un modo di ragionare e non di un linguaggio. Rispetto a C/C++ elimina l'aritmetica dei puntatori e la (de)allocazione esplicita della memoria, presente la funzionalità del garbage collector automatico

Un programma Java si scrive una volta e gira su tutte le piattaforme che hanno una macchina virtuale

Il compilatore è un SW in grado di rilevare eventuali errori e traduce il codice sorgente nel codice macchina.

Il bytecode è un linguaggio macchina virtuale(cioè nessun processore lo supporta) i file class vengono eseguiti dalla java virtual machine

I programmi che si eseguono su tutte le piattaforme devono girare ALLO STESSO MODO

Rilevazione di errori e del typechecking a tempo di compilazione e di esecuzione. Mascheramento

dei puntatori all'utente. Gestisce le eccezioni e della memoria da parte dell'utente.

Java presenta il multithread cioè mandare in esecuzione diversi pezzi dello stesso programma contemporaneamente.

JAVA 2 STANDARD EDITION

Pacchetto di programmi con cui lavorare in Java, primi fra tutti il

compilatore e la JVM per la piattaforma scelta.

Le applicazioni Java sono tipicamente programma general-purpose eseguiiti su ogni macchina dove è installato un JRE.

Esistono due ambienti principali di deploy:

-JRE fornita da Java 2 Software Development Kit(SDK) che contiene il set completo di classi

per tutti i package della tecnologia Javacontains

-L'altro principale ambiente di deploy e di runtime è sul web browser.

La maggior parte dei browser forniscono un interprete per la tecnologia Java ed un ambiente di runtime.

La JVM è una machcina virtuale che è implementata per emulare il software su di una macchina reale. Fornisce le specifiche della piattaforma HW sulla quale

interpretare ed eseguire tutto il codice scritto in Java.

Il bytecode p un tipo di luinguaggio che può essere inteso solo dallla JVM ed è indipendente

da ogni particolare HW cosicchè ogni computer con un interprete Java può eseguire il programma compilato

senza preoccuparsi del tipo di computer su cui è stato compilato.

Il Garbage collector si occupa di liberare automaticamente la memoria che può essere liberata.

La sicurezza del codice è ottenuta in Java attraverso l'implementazione del suo ambiente

di runtime (JRE), che esegue il codice compilato per una JVM ed esegue il caricamento

delle classi, la verifica del codice ed infine l'esecuzione del codice.

Il Class Loader carica le classi necessarie al programma Java.

SINTASSI DI BASE

L'inizio di un programma java è la funzione main, che è il punto da cui

parte il flusso di controllo(dalla prima istruzione del main)

Il nome del file deve essere uguale al nome della classe con estensione.java

Ci sono 8 tipi primitivi di java:

-numeri interi(int, short, long, byte)

-numeri in virgola mobile(float, double)

-tipo carattere (char)

-tipo booleano

Memoria per i vari dati:

-int: 4 byte

-short: 2 byte

-long: 8 byte

-byte: 1 byte

-float: 4 byte

-double: 8 byte

-char: 2 byte

I caratteri contengono un unico carattere. Le costanti di tipo char sono racchiuse

tra apici singoli. Si tratta di un carattere dello schema

Unicode(cioè raccoglie tutti i caratteri esistenti).

I tipi boolean hanno due valori: true e false.

Le variabili devono essere dichiarate e sono definite da un identificatore(il nome)

e dal tipo della variabile. Gli identificatori sono case-sensitive.

E' obbligatorio dichiarare il tipo e l'accesso ai dati è semplice siccome

non ci sono i puntatori.

Nei blocchi di istruzioni non è possibile utilizzare lo stesso nome per variabili

in blocchi annidati. ES di cosa non si può fare:

{

int a = 10;

{

int a = 20; //NO!

}

}

Lo spazio di esistenza di una viariabile è il suo SCOPE. Può essere usata solo

all'interno del suo scope.

Con casting si intende il forzare un valore di un tipo ad assumere un tipo diverso.

In questo modo è possibile ottenere gli asegnamenti di interi a reali e

di reali double a reali float.

per renderli possibili bisogna usare i cast:

double v1 = 10.0;

float v2;

v2 = (float)v1;

Nel caso in cui si provi ad eseguire il cast implicitamente il compilatore risponde

con un errore. Java impone al programmatore di fare scelte consapevoli.

Nelle operazioni tra tipi diversi, il risultato è del tipo più alto, tranne nel caso in cui

byte + byte = int;

float + float = double;

L'istruzione switch è utilizzabile sono con valori di tipo char e valori di tipi numeri int

All'interno dei cicli, l'istruzione continue permette di saltare un'iterazione

del ciclo nel caso in cui accada una determinata condizione, passando alla successiva

all'interno del ciclo.

Per tutti i tipi primitivi il passaggio dei parametri è per valore. Le modifiche su tali variabili

non sono visibili al di fuori alle quali vengono passate.

Con l'introduzione dei Varargs la possibilità di dichiarare metodi che

accettano zero o più argomenti.

ES:

static void setArgs(int... x) {}

Con gli ENUM è possibile definire un tipo di dato personalizzato.

Non si può istanziare sono implicitamente public static final. Estende implicitamente

ENUM quindi non può estendere un'altra classe. Un ENUM non può implementare un'interfaccia

e i costruttori sono implicitamente privati

ARRAY

Sono un tipo speciale che aggrega un gruppo di variabili dello stesso tipo. La dimensione di un array

è fissa ed è stabilita in fase di creazione.

ES:

int[] arrayOfInt

Il tipo array viene dichiarato mediante [] e costruito mediante l'operatore new

int[] arrayOfInt = new int[100]; //crea un array che può contenere 100 numeri interi

Gli elementi di un array sono accessibili mediante indice.

Per conoscere le lunghezza di un array si può essere la proprietà length

Per copiare un array è disponibile il metodo

System.arraycopy(Object from, int fromindex, Object to, int toindex, int count)

dove:

from è l'array sorgente

fromindex è l'indice del cui cominciare a copiare

to array di destinazione

toindex è la posizione in cui copiare nell'array destinazione

count è il numero di elementi da copiare

Non esistono veri array multidimensionali, ma si parla di array di array

double[] [] matrix = new double[4][4]

Nel caso in cui si tenti di accedere ad un elemento che non esiste, cioè fuori dal

range dell'array si avrà un errore a run-time

CLASSE STRING

La classe String è final, cioè gli oggetti istanziati a partire da esse NON si possono

modificare. ES:

String s = "hello"

Presenta l'operatore di concatenazione.

Inoltre:

-String() crea una stringa vuota

-String(String s) crea una copia della stringa s//copy constructor

-int length() restituisce la lunghezza della stringa

-char charAt(int i) ritorna l'i-esimo carattere

I confronti si fanno tramite:

-equals(String anotherString) oppure

-compareTo(String anotherString) che ordina le stringhe rispetto all'ordine alfabetico

La classe StringBuffer fornisce stringhe che possono essere modificate. Viene utilizzata per

creare e manipolare dinamicamente sequenze di dati. La lunghezza è fissata alla creazione, ma

può essere estesa e recuperata.

Metodi:

-StringBuffer() crea un buffer vuoto di lunghezza 16 char

-StringBuffer(int length) crea un buffer vuoto di lunghezza lenght char

-StringBuffer(String str) crea un buffer con la stringa di lunghezza (str.length()+16) char

-StringBuffer append(tipoDato var) appende in coda il tipoDato contenuto in var, trasformandolo in String

-StringBuffer insert(int offset, tipoDato var) inserisce alla posizione offset il

tipoDato contenuto in var, trasformandolo in String

LE CLASSI

Una classe è costituito da:

-il modificatore di accesso

-gli attributi: variabili dichiarate fuori da tutti i metodi ma all'interno della classe

-costruttore: da un valore ad ogni attributi

-altre funzionalità offerte dalla classe

Nello specifico i costruttori hanno lo stesso nome della classe. Non specifica il tipo di ritorno

ed ha il compito di valorizzare tutti gli attributi della classe.

Se non si valorizza un attributo il compilatore assegna valori di default:

-null per gli oggetti (String compreso)

-false per i boolean

-0 per i numeri

Se non viene scritto nessun costruttore, il compilatore aggiunge un costruttore di default con

zero parametri e tutti gli attributi settati con i valori di default

METODI

Specificare il tipo di ritorno (void se non c'è). Se il tipo di ritorno è diverso da void

il metodo deve avere un return

INCAPSULAMENTO

La consistenza e la coerenza di un oggetto nel tempo dipende dal valore dei suoi attributi.

Le operazioni di scrittura sugli attributi sono le più pericolose perchè impattano sull'archietettura dell'oggetto. L'incapsulamento è la tecnica con cui Java consente di rendere sicure le classe progettate

e consiste nel vietare l'accesso diretto agli attributi di un oggetto.

Per fare ciò in Java si usa il modificatore private sugli attributi e si implementano

i metodi getter e setter.

NB: nel metodo di lettura, se il tipo non è primitivo, è preferibile, ritornare una copia

dell'oggetto, non il suo riferimento.

I modificatori di accesso descrivono chi può usare i metodi, variabili di istanza e le classi

MODIFICATORE PUBLIC:

Se viene applicato sugli attributi permette a chiunque abbia un riferimento ad un

oggetto di questo tipo di modificare queste variabili. Sui metodi permette a

chiunque può richiamare questi metodi e sulle classi qualunque classe può utilizzare questa classe

NB: un file può contenere al max 1 classe pubblica che da il nome al file.

MODIFICATORE PROTECTED

Consente la visibilità di attributi e metodi di una classe ad una famiglia di classi, cioè le

sottoclassi e tutte le classi del pacchetto.

MODIFICATORE PRIVATE

Applicato su attributi e metodi li rende visibili solo nella classe dove sono dichiarati/definiti. Mentre sulle classi non ha senso definire una classe private (A meno che non sia una inner class).

Quando si imposta una classe è buona regola:

-impostare gli attributi privati

-inizializzare sempre i dati(tramite il costruttore)

-inserire metodi di accesso/modifica solo se opportuni

-non utilizzare troppi tipi essenziali

-suddividere le classi con troppe responsabilità

-assegnare a classi e motedi significativi

MODELLI E UML

Un modello dell'analisi è un'astrazione di cosa il sistema deve fare, non di come deve farlo. Viene realizzato

tramite notazione grafiche semplici e precise in modo da essere facilmente interpretato.

vengono mostrate tutte le componenti del problema in modo da poter ragionare sulla sua globalità.

Per fare ciò si utilizza UML, un linguaggio e notazione universale. Si tratta di un linguaggio di progettazione e non è un metodo di sviluppo. Non prescerive sequenze di processo ed aiuta ad apportare modifiche ad un sistema.

Prima si definisce il nome della classe, poi gli attributi ed infine i metodi.

Ad ogni classe viene messa un’associazione che indica una connessione logica tra le istanze di classi. Nello specifico:

-Ruolo: indica una "direzione logica" nell'associazione.

-Molteplicità: quanti oggetti possono partecipare all'associazione

È presente anche la navigabilità tra le classi, cioè quella proprietà definisce un verso di percorrenza fra le associazioni.

La responsabilità è su un solo lato e viene utilizzata a livello di design, non ha una traduzione implementativa immediata (nei diagrammi viene rappresentata tramite una freccia)

Aggregazione

Indica una collezione di oggetti, un insieme ben definito, presenti in un'altra classe. La classe aggregato e può

esistere senza alcune classi parte. Si tratta di una relazione non forte, cioè una relazione nella quale le classi parte

hanno un significato anche senza che sia presente la classe tutto.

Di una classe devono generalmente esistere più istanze, quindi bisogna vagliare le eventuali classi di cui esiste

un solo oggetto. Se una classe ha un solo attributo, si valuti se non possa essere considerata come attributo di altre

classi più complesse.

EREDITARIETA' E POLIMORFISMO

Uno dei punti di forza di Java. Consente di massimizzare il riutilizzo del codice e aggiungere

funzionalità al codice preesistente, senza modifiche al diagramma delle classi.

Per definire una sottoclasse si utilizza la seguente sintassi

class SottoClasse extends SuperClasse

La parola chiave extends indica proprio che si sta creando una nuova classe derivante da una

già esistente.

Il costruttore deve richiamare un costruttore della superclasse con la sintassi

super([parametri]);

Questa deve essere la prima istruzione del costruttore della sottoclasse, dopo le istruzioni

specifiche. Es:

public Manager(String n, double s, Date d, String ns) {

super(n,s,d);

secretaryName = ns;

}

Se non si scrive alcun costruttore la JVM ne aggiunge automaticamente uno con la chiamata

a super() con zero parametri. Se la superclasse ha il costruttore a zero parametri, questo

viene invocato. Se invece non è presente tale costruttore, la JVM da errore di compilazione.

L'overriding significa sovrascrivere un metodo della superclasse. Per fare questo bisogna usare lo stesso

nome del metodo e gli stessi parametri. In questo modo la sottoclasse maschera il vecchio metodo con il nuovo

scritto apposta per lei.

Esiste una classe da cui tutte le classi ereditando per default: la classe Object. E' la superclasse

di tutte le classi, non ha attributi possiede un numero limitato di metodi.

I metodi Object sono quindi invocabili su qualunque oggetto Java, ma affinché siano efficaci è necessario

sovrascriverli.

Il binding dinamico consiste nello stabilire i collegamenti tra funzioni a run-time. Nel momento in cui

viene eseguita l'invocazione di un metodo, la JVM determina la classe in cui cercare il metodo.

Il compilatore verifica che un oggetto dispone o meno di un metodo, analizzando tutta la

gerarchia di classi cui fa parte. LA JVM sceglie il metodo da invocare su un oggetto secondo il

seguente criterio:

-verifica che la classe che ha istanziato tale oggetto possieda un metodo con tale nome e parametri

-in caso affermativo lo utilizza. In caso contrario cerca nella classe superiore un metodo con quel nome e con quei parametri finché non ne trova uno valido.

Per impedire la creazione di sottoclasse da una classe occorre dichiararla classe con il modificatore final

public final class Nome {}

Anche un metodo di una classe può essere impostato final. In questo caso non di potrà fare l'overriding del metodo.

Il POLIMORFISMO rappresenta la possibilità che una entità possa assumere molte forme. Consente di fare riferimento ad oggetti di classi diverse mediante la stessa

entità e svolgere la stessa azione in modi diversi a seconda della particolare istanzanzione.

A run-time verranno invocati i metodi di cui l'oggetto è effettiva istanza. Non è necessario conoscere in anticipo

tutti i tipi diversi da gestire.

Ad un oggetto può essere assegnato un puntatore di un tipo diverso dalla classe che lo ha

istanziato? Si, solo se il puntatore è di un tipo presente nell'albero di ereditarietà del tipo

dell'oggetto.

Assegnare un puntatore di un tipo più generale non è mai un problema. (ovviamente gli attributi e i metodi specifici della classe "figlio")

Il caso contrario è possibile solo se l'oggetto specifico stia puntando un oggetto generico. Ed anche se ciò sia vero

è comunque necessario un cast. In questo caso i metodi ed attributi nascosti all'oggetto specifico diventano di nuovo

visibili.

Per gestire in maniera corretta questi tipi, l'oggetto porta con sè il nome della classe dalla quale è stato istanziato.

Tale informazione è conservata in un campo speciale e non cambia nel tempo. Quindi l'oggetto ha una propria identità, indipendentemente

dal tipo del puntatore.

Per leggere il valore di questo campo, si applica il metodo getClass().

Manager m = new Manager([param]);

Impiegato i = m;

i.incrSalario(800);

Nel caso in cui entrambe le classi abbiano il metodo increSalario, la JVM controlla il tipo reale dell'oggetto e cercherà il metodo

a partire da questo tipo, quindi in questo caso verrà chiamato quello di manager.

CLASSI ASTRATTE

Sono classi generiche al punto di essere considerate struttre portanti. Non posso avere istanze

ed una classe con uno o più metodi astratti deve essere dichiarata abstract. Si possono creare

sottoclassi della classe astratta e da questa generare oggetti.

L'interfaccia invece è un meccanismo con cui alcune entità che non si relazionano fra di loro

possono interagire. In java l'interfaccia rappresenta un tipo, definisce metodi e piò essere vista come

una classe astratta degenere, dove tutti i metodi sono abstract.

L'interfaccia consente di gestire il problema della mancata ereditarietà multipla, e di definire

tipi generici, prima di implementare realmente le classi.

Forniscono gli schemi di comportamento che altre classi sono tenute ad implementare. Utilizzando variabili

di tipo interfaccia potrò applicare metodi di tale interfaccia senza sapere di quali oggetti sis tratta. Infatti

per il polimorfismo si invoca il metodo specifico determinato dalla vera natura dell'oggetto.

Si possono dichiarare solo metodi pubblici e costanti di classe. Si usa la keyword interface al posto di class

Una classe può implementare uno o più interfacce ma eredita da un singolo superclasse.

Una classe che intenden implemnentare un'interfaccia deve obbligatoriamente ridefinire tutti i metodi dell'interfaccia.

Non possono essere create con new ma si piò definire una variabile con il tipo dell'interfaccia.

Interfacce segnaposto: non richiedono l'implementazione di metodi e forniscono informazioni sulla natura della classe.

La classe che implementa Cloneable afferma che è possibile clonare oggetti istanziati da essa, secondo il

criterio del metodo Object clone (), cioè campo a campo.

Si clona un oggetto per evitare di restituire riferimenti di campi privati che possano determinare

un indebolimento dell'incapsulamento. Per definire una modalità di clonazione per una classi si può implementare.

LA GESTIONE DEGLI ERRORI E ASSERTION

Il meccanismo di gestione degli errori permettere di trasferire i comandi dal punto in cui si è

verificato l'errore a un gestore degli errori in grado di affrontare la situazione.

In generale si restituisce uno speciale codice di errore che il metodo di chiamante analizza.

Java invece permette di uscire dal metodo con un percorso di uscita alternativo che non restituisce nessun valore.

Se si verifica un errore si ricerca un gestore dell'errore in grado di far fronte alla situazione.

Il gestore di rivolgerà alle varie entità in campo nella speranza che qualcuno si accolli il problema

Gli errori che si possono gestire sono rappresentati da oggetti derivati dalla Classe Exception che si

chiamano eccezioni.

\_Throwable è la classe che modella tutto quello che può essere sollevato dalla JVM. Da questa partono

due gerarchie:

\_Error

\_Exception

La famiglia di Error descrive errori interni ed esaurimento delle risorse nell'ambito del sistema di esecuzione

Java(di solito non gestibili)

La famiglia di Exception è invece una situazione imprevista che il flusso di un'applicazione può incontrare

Le exception quindi si dividono in

\_unchecked (derivano da RuntimeException)

\_checked (non derivano da RuntimeException)

Le eccezioni checked vincolano il programmatore. quelle unchecked no. Una RuntimeException è un errore

che viene intercettato dalla JVM durante il suo funzionamento. Spesso si verifica perché è stato commesso un errore di programmazione

Quindi le RuntimeException:

\_ Non rientrano nel meccanismo vincolante delle Exception

\_ Di solito non si gestiscono

Nell'header del metodo si indica l'eccezione che potrebbe essere sollevata attraverso

lo specificatore throws

es: public String readLine() throws IOException;

Se ci sono più eccezioni, vanno indicate tutte nell'header, separate dalla virgola

throws EOFException, MalformedURLException.

Un metodo deve dichiarare ogni eccezione che lancia, così come deve specificare il suo eventuale tipo di ritorno.

Le eccezioni personalizzate devono estendere exception oppure una sua sottoclasse, e dovrebbero

avere nomi significativi per modellare errori/anomalie dell'applicazione

Se un metodo invoca un metodo che solleva un'eccezione, esso potrà scegliere tra:

\_ dichiarare l’eccezione (equivale a rilanciarla)

\_ gestirla (affrontare il problema)

Nel primo caso l'eccezione si propaga al metodo chiamante e così via. Il metodo più adatto per gestire il problema

(in base al ruolo della classe che lo implementa), si dovrà incaricare di intercettare l'eccezione. Se neanche il main

gestisce l'eccezione, la JVM termina stampando lo stacktrace della propagazione dell'eccezione.

Per gestire le eccezioni si usa il blocco try/catch e si elimina lo specificatore throws

Nel blocco try si inserisce il codice dove si potrebbe presentare la situazione di errore.

Nel blocco catch invece si inserisce il gestore dell'eccezione.

Se all'interno del blocco try si genera un'eccezione, il programma salta il rimanente codice

del blocco try, ed esegue il codice del gestore nel blocco catch.

In un blocco try possono essere gestite più eccezioni, utilizzando più blocchi di istruzioni catch distinte.

Con l'overriding di un metodo non si possono aggiungere al nuovo metodo più specificatori

throws di quelli presenti nel metodo della superclasse. Invece si può estendere il tipo di

un'eccezione della super sollevare quello.

E' possibile inoltre gestire l'eccezione, ma avere comunque necessità di avvisare anche il chiamante

del problema in questione. Uso il throw dell'eccezione all'interno del catch.

Il codice nel blocco finally viene eseguito indipendentemente che si verifichi o meno un'eccezione.

NB: il finally viene utilizzato per rilasciare risorse di sistema, chiudere file, ecc..

Ci sono tre situazioni nelle quali il programma eseguirà l'istruzione finally:

\_il codice non lancia nessuna eccezione

\_il codice lancia un'eccezione che viene intercettata dall'istruzione catch

\_il codice un'eccezione che non viene intercettata da nessuna istruzione catch.

Se ci sono istruzioni break, continue o return all'interno del try o di un catch verranno

eseguite solo dopo l'esecuzione del blocco finally.

ASSERZIONI

Un'asserzione è un'istruzione che permette di testare eventuali comportamenti che

un'applicazione deve avere. La parola chiave per utilizzare un'asserzione è: assert

Il formalismo è il seguente:

1)assert espressione\_booleana;

2)assert espressione\_booleana: espressione\_stampabile;

Se l'espressione è true il programma prosegue normalmente, altrimenti viene lanciata

un'AssertionAError

NB: per utilizzare le assert bisogna usare java -ea nomeProgramma

GENERICS

Con i generics viene introdotta la possibilità di poter dichiarare una classe nella seguente modalità:

public class NomeClasse <T>

dove T viene definito parametro di tipo formale.

La variabile di tipo può essere usata in qualsiasi punto della classe e può assumere qualsiasi tipo di dato. Può essere:

-una classe

-un’interfaccia

-un tipo primitivo

In una dichiarazione di una classe generica si possono inserire più parametri di tipo formale, ma ciascun parametro deve essere unico nella sua dichiarazione.

ES:

public class NomeClasse <T, T> genera un errore alla seconda occorrenza

public class NomeClasse<T, U> è ammesso

I parametri di tipo più utilizzati sono:

\_ E elemento (utilizzato molto dal Java Collections Framwoerk)

\_K Chiave

\_N Numero

\_T Tipo

\_V Valore

\_S,U,V ecc Secondo terzo quarto tipo

Si possono presentare situazioni in cui si desidera limitare i tipi consentiti da passare a un parametro di tipo. Per esempio un metodo che lavora sui numeri può desiderare di accettare solo istanze di Number o delle sue sottoclassi. Per questo motivo esistono i parametri di tipo Bounded. La dichiarazione funziona nel seguente modo: elencare il nome del parametro di tipo, seguito dalla parola chiave extends, seguita dal “bound superiore” <U extends Number>

Se modifichiamo il metodo visto in precedenza

Public <U> void insppect(U u). In questo modo public <U extends Nummber> void inspect(U u) e passando al metodo una stringa, al compilazione fallisce.

Nei generici è possibile rappresentare un tipo sconosciuto utilizzando il carattere jolly “?”.

Scatola <? Extends Classe> var… Questa istruzione si legge: “Un tipo sconosciuto che è un sottotipo della Classe”.

NB: con la parola chiave extends mettiamo un vincolo superiore per poter mettere un vincolo inferiore si usa la parola chiave super.

<? Super Classe> si legge “un tipo sconosciuto che è un supertipo di classe.

È anche possibile specificare un tipo sconosciuto con un “carattere jolly unbounded <?>.

Equivale a <? Extends Object>

Le Collection

L’interfaccia collection modella insiemi. È la root della gerarchia delle Collection. Non esiste nessuna implementazione diretta, ma delle sottointerfacce specializzate che sono state implementate. Alcune Collection consentono i duplicati, altre invece no. Le collection prevedono generici metodi per

* Aggiungere, rimuovere o cercare elementi
* Verificare la presenza e il numero di elementi
* Recuperare il “navigatore” della collection

L’iterator è un’interfaccia del pacchetto java.util. Esso consente la navigazione all’interno di una collection e per ogni collection deve essere previsto un valido iterator. Un iterator si ottiene chiamando il metodo iterator() di Collection. I metodi dell’interfaccia Iterator sono:

-boolean hasNext()

-Object next()

-void remove()

Interface List

Public interface List extends Collection.

Si tratta di una collezione sequenziale di oggetti e l’accesso avviene tramite indice. **AMMETTE DUPLICATI** ed alle funzionalità ereditate da Collection vengono aggiunte funzionalità specifiche per l’inserimento e la ricerca. Esiste un iteratore speciale, **LISTITERATOR** che permette lo scorrimento bidirezionale

Class Vector

Implementa List attraverso array di dimensione variabile. Ha vari costruttori:

-Vector(): costruisce un vettore di dimensione iniziale 10

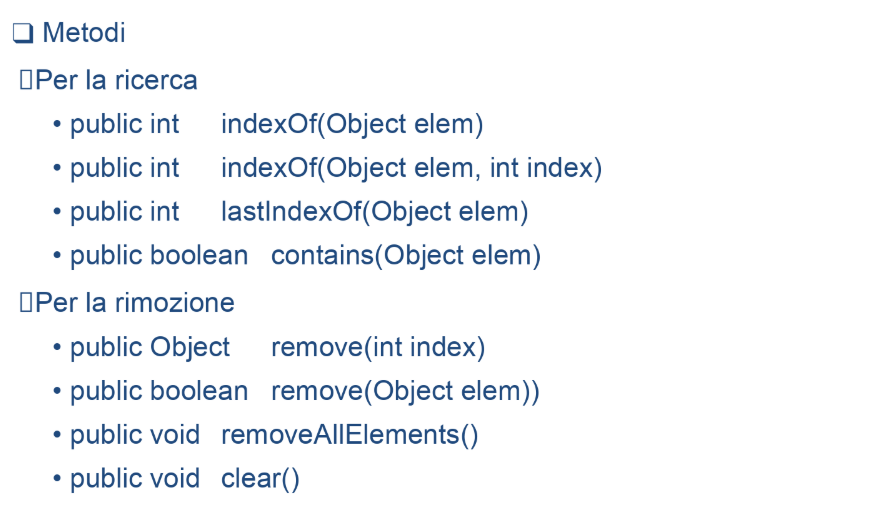
-Vector(Collection c): costruisce un vettore a partire da una collezione data di oggetti

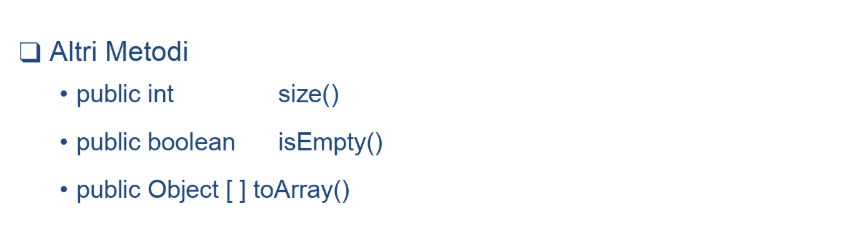
-Vector(int initialCapacity): costruisce un vettore vuoto di dimensione iniziale data

-Vecotr(int initialCapacity, int capacityIncrement) costruisce un vettore vuoto specificando la capacità iniziale e di incremento

Alcuni metodi della classe vector per l’utilizzo:







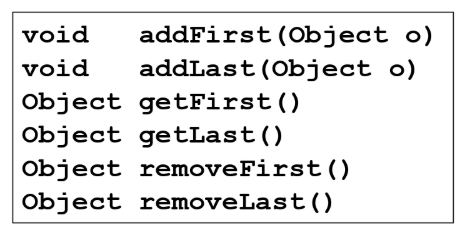
ArrayList

Implementa List attraverso un array di dimensione variabile. Funziona come un Vector e si differenzia con l’utilizzo dei Thread.

NB: la size restituisce il numero di elementi INSERITI nell’arrayList, non la dimensione.

LinkedList

Implementa List attraverso oggetti linkati fra loro. I metodi di LinkList comprendono quelli di List e altri metodi per la getsione di code e pile:



Generics e Collections

I generics consentono di creare collezioni o contenitori che memorizzano solo il tipo di oggetti specificato. Il tipo di oggetto si indica tra le parentesi angolari. ES:

*Collection <Tipo> collection = new MyCollection <Tipo>();*

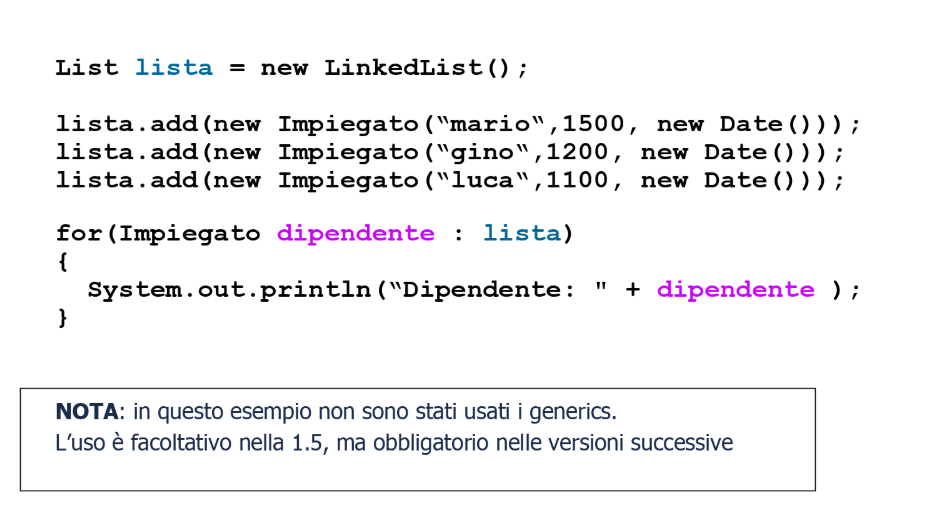
Non è possibile inserire oggetti di natura diversa da quelli indicati. I vantaggi sono:

-non sono più necessari cast per estrarre gli oggetti dalla collezione, se sono del tipo definito

-si evita l’eccezione di ClassCastException

Le collection possono usare una versione “evoluta” del costrutto for, il foreach.

Per usarlo non bisogna dichiarare oppure incrementare gli indici, né richiamare metodi dell’Iterator (per generiche collection). Inoltre, il codice verrà ripetuto automaticamente per ogni elemento che abbiamo nella collezione. ES:



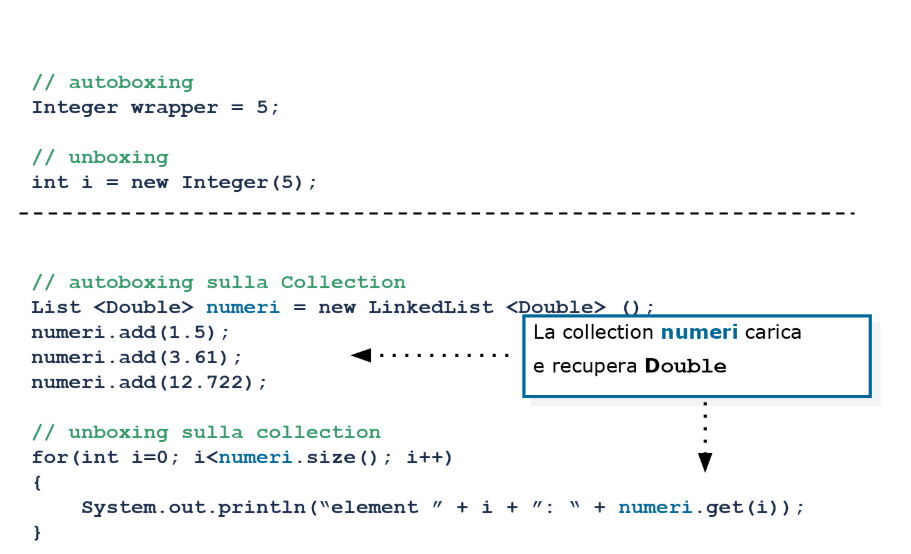
Le problematiche sono:

* Conversioni da primitivi a wrapper relativi
* Le collection di Java supportano solo oggetti e non primitivi

La soluzione è la seguente:

* L’autoboxing consente di assegnare primitivi a wrapper e il caricamento automatico di primitivi in una collection
* L’unboxing consente di assegnare wrapper a primitivi e il recupero del wrapper(creato dal primitivo) e caricato sulla collection

In automatico avvien la costruzione del tipo wrapper(prima del caricamento) e il cast(per la lettura)



Interface Set

public Interface Set extends Collection.

Rappresenta una Collection che non accetta elementi ripetuti. Formalmente un Set non contiene una coppia di elementi e1, e2 tali che e1. equals(e2) sia true. Modella gli insiemi matematici ed alcune implementazioni di set sono HashSet e il TreeSet

HashSet

Si occupa della gestione di insiemi **senza ripetizioni** ma non ordinati. La struttura dati che lo rappresenta è un array i cui elementi sono liste, ciascuna lista è un bucket. Gli oggetti dell’insieme dovrebbero ridefinire:

-il metodo equals()- per distinguere i doppioni

-il metodo hashCOde()-per la “gestione dei buckets”

In modo che 2 oggetti con gli stessi valori per gli attributi hanno stesso Hash Code. In pratica 2 oggetti uguali, ma distinti devono produrre stesso codice hash. Il numero di bucket totali e il fattore di carico per ciascun bucket si può impostare col costruttore

TreeSet

Si occupa della gestione di insiemi ordinati mediante alberi binari. Gli oggetti dell’insieme devono essere Comparable, cioè devono fornire il metodo compareTo. In aggiunta (per avere più criteri) si implementano classi di tipo Comparator. Se gli elementi dell’insieme sono tipo classici, per essi è già definito un ordine naturale. Indipendentemente da come vengono inseriti nell’insieme, vengono memorizzati mantenendo sempre l’ordine stabilito.

LE MAP

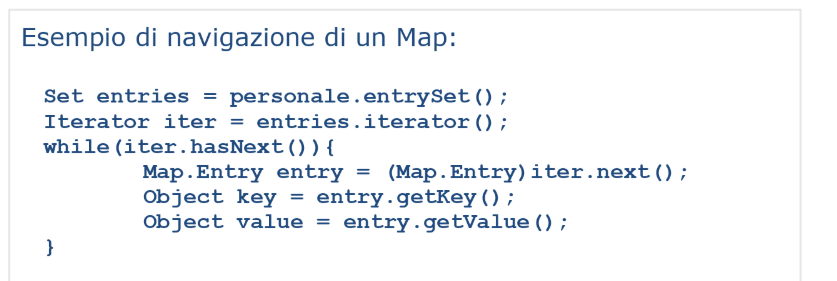
L’interfaccia Map descrive insiemi su cui elementi si può eseguire una ricerca per chiave. Il generico elemento di una Map si dice **entry** della mappa ed è modellato con un oggetto di tipo **Map.entry.** Esso rappresenta la coppia K, V.(chiave, valore). Ogni oggetto V possiede una sola chiave K. Non esistono oggetti V con la stessa chiave K.

La navigazione di una Map può essere vista in 3 modi:

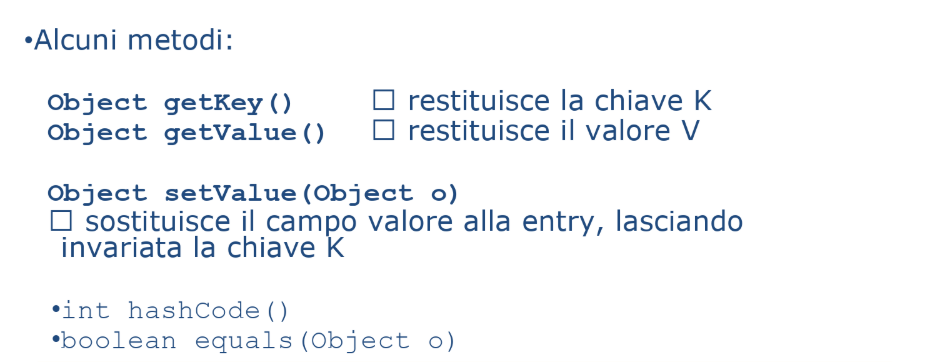
-set di chiavi

-collection di valori

-set di oggetti chiave-valore



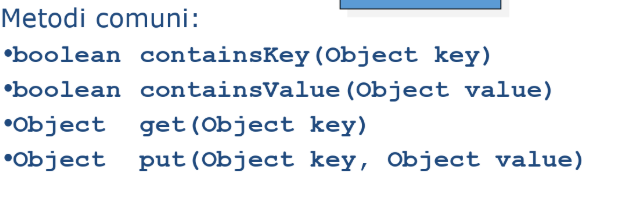
La map.Entry è un’interfaccia. Ogni mappa concreta dispone di un concreto oggetto di tipo Map.Entry.



Tra le classi che implementano il Map troviamo:

-HashMap

-TreeMap



**Classe HashMap**

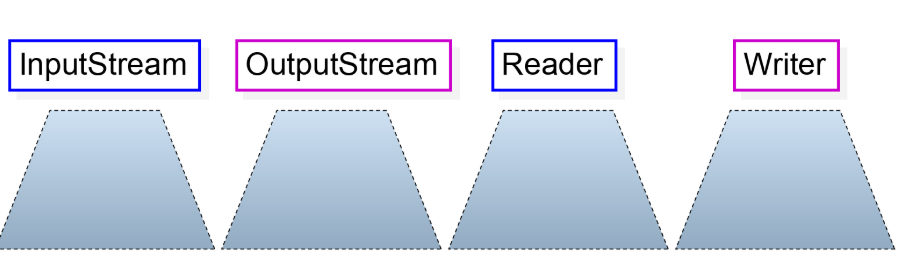
Viene utilizzato per la gestione di Map non ordinati, utilizzando le Hash Table. Le chiavi relative agli oggetti della mappa dovrebbero ridefinire il metodo hashCode in modo che 2 oggetti con gli stessi valori per gli attributi devono produrre lo stesso HashCode e il metodo equals (per distinguere le chiavi doppie). Per ottimizzare le operazioni, la struttura dati che lo rappresenta è un array i cui elementi sono liste, dove ciascuna lista è un *bucket.* Il numero di elementi(buckets) e il fattore di carico per ciascuno si possono impostare col costruttore.

**Classe TreeMap**

Questa classe viene utilizzata per la gestione di Map ordinati e utlizza una struttura ad albero. Le chiavi relative agli oggetti della mappa devono fornire il metodo compareTo dell’interfaccia standard Comparable, oppure in aggiunta si implementano classi di tipo Comparator per definire l’ordine delle chiavi. Per i tipi classici è già definito un ordine naturale. Indipendentemente da come vengono inseriti nell’insieme vengono memorizzati mantenendo sempre l’ordine indotto dalle chiavi.

JAVA.IO

I flussi sono sequenze di byte che viaggiano da un’origine a una destinazione lungo un percorso di comunicazione. Il pacchetto *java.io* comprende le seguenti principali gerarchie di classi.



Lo stream di input è l’oggetto dal quale si possono leggere una sequenza di bytes, mentre quello di output è quello nel quale è possibile inviare una sequenza di bytes. Vengono modellati con le classi astratte *InputStream e OutputStream*  che hanno alberi gerarchici simmetrici e che sono presenti nella JDK dalla 1.0.

Gli Stream sono orientati al byte, cioè leggono e scrivono singoli Bytes. Successivamente nella versione JDK 1.1, sono state create le classi astratte *Reader* e *Writer*. Queste invece sono orientate ai caratteri e sono ideali per la lettura/scrittura di caratteri e stringhe.

InputStream

La classe InputStream ha un metodo astratto

*Public abstract int read() throws IOException*

Legge sequenzialmente byte e restituisce il byte letto su int oppure -1(se incontra la fine della sorgente di input). L’oggetto System.in è l’input stream standard. È un oggetto di tipo InputStream e consente di leggere sequenze di caratteri dalla tastiera. È un oggetto statico, quindi è sempre disponibile, e non bisogna istanziarlo.

OutputStream

La classe OutputStream definisce il metodo astratto

*Public abstract void write(int b) throws IOException*

Invia il parametro nello stream. L’oggetto system.out è l’output stream standard. È un oggetto di tipo OutputStream e consente di visualizzare dati sul dispositivo di default (di solito video). È un oggetto statico, quindi è sempre disponibile, e non bisogna istanziarlo.

Una volta conclusa la lettura o la scrittura lo stream deve essere chiuso con il metodo close() per rilasciare le risorse ad esso dedicate.

Esistono 4 principali gerarchie di stream in java e poi esiste un’ulteriore suddivisione:

\_Stream base: istanziabili direttamente

\_Stream filtrati: che derivano dalle classi

FilterInputStream/FIlterOutputStream che necessitano di uno stream base per essere costruiti ed operare(tecnica di “composizione”, detto filtraggio)

Gli stream di base indicano solitamente la sorgente/destinazione della lettura/scrittura, mentre gli stram filtrati invece specificano il tipoDati/modalitàTrasporto. Componendo le due tipologie di stream si ottengono tutte le possibili combinazione di sorgenti/destinazioni e dati. Questa tecnica si chiama *Filtraggio*.

FileInputStream e FIleOutputStream permettono di leggere e scrivere dati da file. Es:

*FIleInputStream fin = new FileInputStream(“prova.txt”);*

oppure

*File f = new File(“prova.txt”);*

*FIleInputStream fin = new FIleInputStream(f);*

ed infine

*int b = fin.read();*

Legge in modo sequenziale i byte del prova.txt. Il file prova.txt deve essere già esistente per lo Stream di input ma non per lo Stream di output.

La Classe File

La classe File serve a manipolare file e directory in modo indipendentemente dalla piattaforma. Oggetti di tipo File rappresentano i nomi dei file e non i file stessi: un oggetto di tipo File può esistere anche se il file che esso rappresenta non esiste affatto!

Per creare fisicamente un file si deve costruire uno stream di output(come FileOutputStream) con un oggetto di tipo File che ne rappresenta il nome. La classe File può rappresentare sia file che directory. Quando il file esiste allora è possibile invocare i metodi di File per effettuare oerazioni come: rinominare, cancellare, cambiare permessi, ecc…

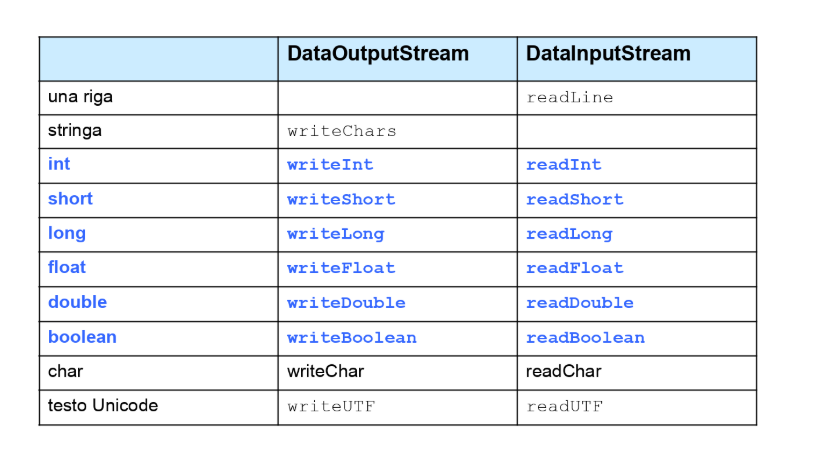
Il DataInputStream è uno stream filtrato. Permette di leggere primitivi da una sorgente e deve essere composto con Stream di tipo sorgente.

Es:

FileInputStream fin = new FileInputStream(“prova.dat”);

DataInputStream din = new DataInputStream(fin);

double s = din.readDouble();



BufferedInputStream

È uno stream filtrato. Serve per migliorare le prestazioni, in quanto inserisce e gestisce un buffer nello stream. Il BufferedOutputStream vine maggiormente usato e chi scrive decide come usare il buffer. ES:

*FileInputStream fis = new FileInputStream(“prova.dat”);*

*BufferedInputStream bis = new BufferedInputStream(fis);*

*DataInputStream dis = new DataInputStream(bis);*

*double d = dis.readDouble();*

Stream per gli Oggetti

Sono Stream filtrati e consentono di scrivere e leggere lo stato di un oggetto. Le classi sono ObjectOutputStream e ObjectInputStream. ES:

FileOutputStream fis = new FileOutputStream(“Oggetti.txt”);

ObjectOutputStream out = new ObjectOutputStream(fis);

Impiegato imp = new Impiegato(“Harry Hacker”, 35000, new Date());

out.writeObject(imp);

Per leggere, filtriamo ObjectInputStream con FileInpuStram

*FileInputStream fis = new FIleInputStream(“Oggetti.txt”);*

*ObjectInputStream in = new ObjectInputStream(fis);*

*Impiegato imp = (Impiegato)in.readObject();*

Gli ObjectStream trasportano solo istanze di classi che implementano l’interfaccia Serializable. Serializable è un’interfaccia di markup: non include nessun metodo, ma è necessario verificare che ogni attributo della classe sia serializzabile.

Un oggetto è serializzabile se si può convertire in una sequenza di byte, in altre parole se si può ricondurre ad un tipo primitivo: “Smontare e rimontare in pezzi primitivi”.

Questo meccanismo funziona anche in ambiente di rete. La serializzazione compenso differenze fra S.O.

Tutte le classi che manipolano caratteri discendono da Reader e Writer. Per leggere caratteri da un file si possono usare:

-FileReader

-FileWriter

Mentre per leggere/scrivere e convertire byte in caratteri e viceversa ci sono: InputStreamReader ed OutputStreamWriter.

InputStreamReader: converte da stream di byte a stream di caratteri. Legge byte e li trasforma in caratteri (seconda codificata specificata)

OutputStreamWriter: converte da stream di caratteri a stream di byte e legge i caratteri (secondo codifica specificata) e li trasforma in byte.

InputStreamReader e OutputStreamWriter sono anche detti “adattatori” perché vengono usati per convertire StreamReader/StreamWriter

Per la lettura da tastiera si utilizza System.in. E’ un InputStream orientato al byte: è agganciato alla tastiera e consente la lettura di un solo carattere per volta.

*InputStreamReader isr = new InputStreamReader(System.in);*

*BufferReader buff = new BufferReader(isr);*

*String l = buff.readLine();*

Per la stampa a video si utilizza System.out. Si tratta di un OutputStream di base: è agganciato al video e consente la scrittura/stampa di primitivi, String ed oggetti.

Lettura Stringhe

Per leggere una stringa da una sorgetnte, bisogna:

\_ottenere/creare lo stream base agganciato alla sorgente

\_filtrarlo con InputStreamReader(ed eventualmente bufferizzarlo con BufferedReader)

ES:

*InputStream stream = new FileInputStream(“nomeFile.txt”);*

*InpuutStreamReader isr = new InputStreamReader(stream);*

*BufferedReader buff = new BufferedReader(isr);*

*String l = buff.readLine();*

readLine() è un metodo di DataInputStream, ma qui siamo quello di BufferedReader.

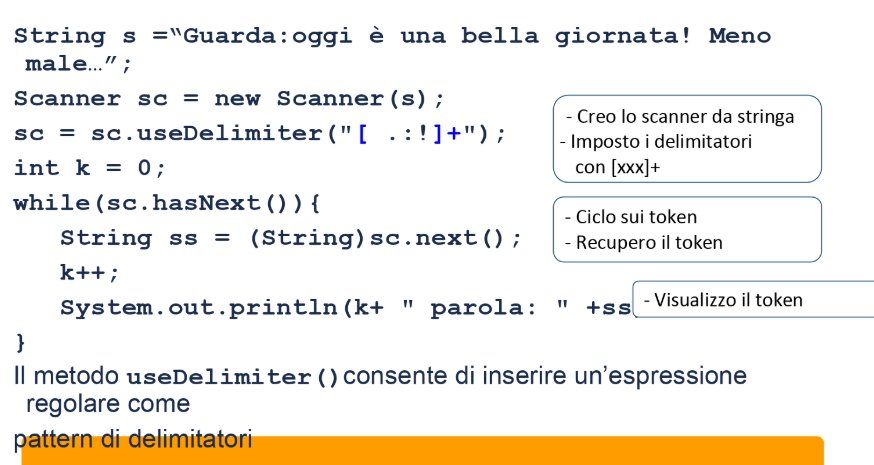
Scanner

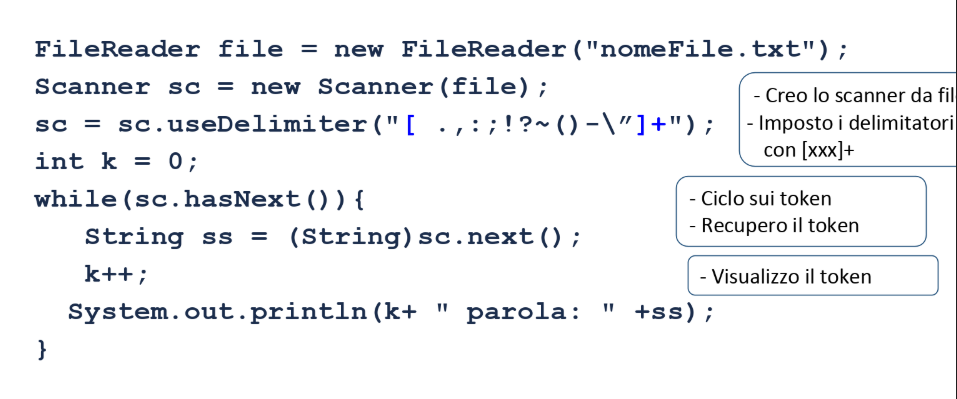
Un oggetto di tipo java.util.Scanner non è uno stream, ma permette di scorrere un testo ed estrarre tipi primitivi e stringhe. Lo Scanner spezza l’inpu in token usando un delimitatore: per default è lo spazio bianco. È possibile usare uno Scanner per leggere da file di testo con varie modalità:

\_Singole parole

\_Singole righe

\_Intero file





È possibile leggere da tastiera con un oggetto Scanner agganciato a System.in

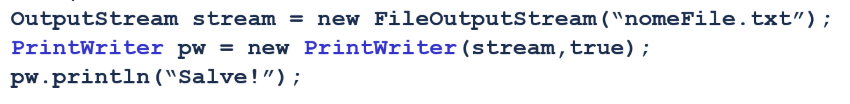
\_per gli interi si usa il nextInt()

\_per le stringhe si usa nextLine()

Per scrivere una stringa verso una destinazione, bisgona:

\_ottenere/creare lo stream base agganciato alla destinazione

\_filtrarlo con PrintWriter(stream già bufferizzato)



NB: il parametro booleano serve per indicare il comportamento del buffer:

\_true: auto-flush (Ogni dato inviato è subito disponibile al destinatario)

\_false: flush manuale(I dati sono nel buffer e non sono disponibili al destinatario fino a quando non viene effettuata l’operazione di flush (metodo flush() sull’oggetto PrintWriter)

Il costruttore PrintWriter(OutputStream stream) usa il boolean a false

Gli oggetti che implementano un qualche tipo di formattazione sono quelli di tipo PrintWriter(per stream di caratteri) o PrintStream(per stream di byte).

\_System.out e System.err usano PrintSstream

\_Per stream formattato di caratteri usano PrintWriter

Gli oggetti di tipoPrintStream e PrintWriter implementano :

\_metodi write basilari

\_un insieme di metodi per convertire i dati in output formattato:

\_ print e println formattano singoli argomenti in modo standard

\_ format formatta argomenti multipli tramite stringa di formato con varie opzioni

Specificatori

Tutti gli specificatori di formato obbligatoriamente iniziano con il % e terminano con il fattore di conversione(1-2 caratteri).

Dettaglio:

\_**d**  formatta un intero

\_**f** formatta un numero in virgola mobile

\_**x** formatta un intero come valore esadecimale

\_**s** formatta qualsiasi valore come stringa

