Programmazione orientata agli oggetti

Astrazione in POO

l'Astrazione dei Dati viene utilizzata per gestire al meglio la complessità di un programma, ovvero viene applicata per decomporre sistemi software complessi in componenti più piccoli e semplici che possono essere gestiti con maggiore facilità ed efficienza.

Le astrazioni possono essere ottenute con la specificazione

(disaccoppiamento) e/o parametrizzazione (riutilizzo)

Astrazione dei dati significa nascondere i dettagli sui dati e controllo/processo significa nascondere i dettagli di implementazione. Nell'approccio orientato agli oggetti, si possono astrarre sia i dati che le funzioni.

I package, le classi, le variabili e i metodi

I nomi dei package, delle classi, delle variabili e dei metodi devono rispettare la notazione CamelCase (esempio MioPackage) ed essere significativi

Una classe è la definizione di un tipo di oggetto.

```
public class Student {
        // instance variables
        private String name;
        private int id;


        public void setId(int newId){
            id = newId;// or this.id = newId;
        }

        // a class variable
        static int numberOfEnrolledStudents = 0;
        …
        Student.numberOfEnrolledStudents ++;

        // a constant
        public static final String degreeName = “Laurea Informatica”;

        …
    }
```

L'invocazione di un metodo potrebbe essere: nomeOggetto.funzione1(...);

Package

Le classi sono raccolte all'interno di contenitori detti package. Ogni package contiene classi che sono in qualche modo correlate fra loro o adatte per uno stesso scopo. In Java esistono molti package predefiniti, ma un programmatore può crearne di nuovi. L'appartenenza di una classe a un package viene indicata specificando il nome del package all'inizio del file sorgente. In assenza di questa direttiva, la classe viene assegnata a un package generico predefinito.

Quantificatori d'accesso

Gli elementi private sono visibili solo nella classe in cui sono dichiarati. È possibile definire anche elementi protected, visibili in tutte le classi dello stesso package e nelle classi derivate, anche se appartenenti ad altri package. Al momento della dichiarazione di una classe, è inoltre possibile specificare un qualificatore public davanti al termine class. Ciò consente di rendere pubblica la classe, ovvero essa può essere utilizzata da qualsiasi altra classe, anche di package diversi.

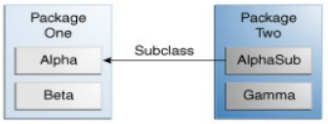
Una classe può essere dichiarata anche astratta utilizzando la parola chiave abstract davanti al termine class. Una classe astratta non può essere istanziata e contiene uno o più metodi astratti, ovvero non definiti. Un altro qualificatore utilizzabile davanti al termine class è final. L'utilizzo del termine final davanti a un attributo consente invece di dichiararlo come costante.

For **methods** and **fields**

Subclasses outside the package

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N



Rules:
- whenever possible use "private"
- for fields ALWAYS use private
- use "public" only for constants meant to be exported elsewhere

Attributi e metodi static

Gli attributi e i metodi di una classe possono essere dichiarati static. Un attributo static permette di definire una proprietà valida per tutti gli oggetti di una classe. L'accesso a un attributo static avviene utilizzando il nome della classe. Analogamente, i metodi statici vengono richiamati usando il nome della classe.

Allocazione dinamica della memoria

In Java gli oggetti possono essere istanziati solo mediante l'operatore new, che richiama il costruttore della classe.

Interfacce

In Java è possibile definire interfacce: si tratta di costrutti simili alle classi, ma che presentano alcune limitazioni. In primo luogo, un'interfaccia non può essere istanziata. Inoltre, tutti i suoi metodi devono essere astratti e pubblici. . Un'interfaccia non può essere utilizzata in sé stessa, ma deve essere implementata mediante una classe. Una classe che implementa un'interfaccia deve definire tutti i metodi dell'interfaccia, per poter essere a sua volta istanziata. Le interfacce realizzano di fatto i tipi di dato astratti (ADT, Abstract Data Types), fornendo le caratteristiche e il comportamento di un tipo di dato in modo indipendente dall'implementazione.

```
/**
 * Interface to be used as a parameter to
 * implement a particular kind of comparison
 * between int.
 */
interface IntComparator {
    /**
     * @param x
     * @param y
     * @return true or false depending on what meaning we want to
     * give to compare. Eg. compare(x,y) can imply x<y, or x=2*y, or ...
     * In the context of sorting only x<y or x>y are useful choices.
     */
    boolean compare(int x, int y);
}
```

```
private static void doParametricSort(int[] a, sortDirection dir) {
    IntComparator ic = null; // the actual comparator that we will be using
    switch (dir) {
    case INCREASING:
        ic = new IntComparator() {
            @Override
            public boolean compare(int x, int y) {
                return (x < y);
            }
        };
        break;

    case DECREASING:
        ic = new IntComparator() {
                @Override
                public boolean compare(int x, int y) {
                    return (x > y);
                }
```

Costruttore

Il costruttore viene richiamato automaticamente quando si crea un'istanza della classe. Il metodo predefinito viene chiamato costruttore di default. In una classe è possibile definire altri costruttori, anche dotati di parametri diversi fra loro, tipicamente per inizializzarne gli attributi.

```
public class Prova {
  ...
  public Prova() {...}
  public Prova(int x, int y) {...}
}
```

EREDITARIETÀ

Nella programmazione a oggetti, l'ereditarietà è un concetto fondamentale che esprime la possibilità di creare nuove classi (classi derivate o sottoclassi) a partire da una classe già esistente (classe base o superclasse). Le nuove classi ereditano gli attributi e i metodi della classe base e possono aggiungere nuovi attributi e metodi.

L'ereditarietà favorisce il riutilizzo del codice, dal momento che è possibile definire nuove classi partendo dalle caratteristiche di classi già esistenti. In tal modo la scrittura di un nuovo programma a oggetti non parte da zero ma riutilizza ed estende classi già esistenti, con il vantaggio di partire da classi già collaudate, e quindi funzionanti, risparmiando tempo di sviluppo.

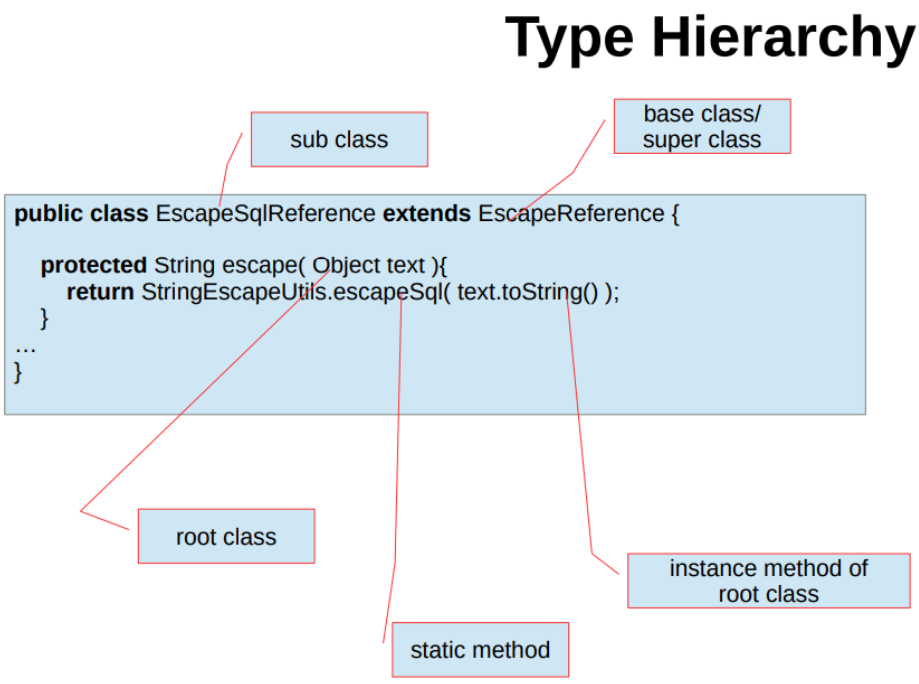
Java supporta solo l'ereditarietà singola, ovvero ogni classe può derivare da una e una sola superclasse. Per dichiarare una sottoclasse si usa la parola chiave extends.

I membri di una classe dichiarati come private non sono ereditati dalla sottoclasse e quindi non possono essere utilizzati. Se invece una classe dichiara attributi o metodi protected o public.

Extends: in Java, la parola chiave extends viene utilizzata per indicare che la classe che viene definita è derivata dalla classe base utilizzando l'ereditarietà. Quindi, in pratica, la parola chiave extends viene utilizzata per estendere la funzionalità della classe genitore alla sottoclasse. In Java non sono consentite eredità multiple.

Implements: in Java, la parola chiave implements viene utilizzata per implementare un'interfaccia.

Nota: una classe può estendere una classe e implementare un numero qualsiasi di interfacce contemporaneamente.



Get e Set

Sono anche detti getter e setter.

Vanno creati per ogni campo/variabile della classe in cui è necessario un dialogo con l'esterno.

Per ogni variabile di istanza, un metodo getter restituisce il suo valore, mentre un metodo setter ne imposta o ne aggiorna il valore.

Esempio di get e set

```
class Rettangolo {

private int base;

private int altezza;

public void setBase(int b) {

base=b;

}

public int getBase() {

return base;

}

}
```

Java Iterator

In informatica, un iteratore è un oggetto che consente di visitare tutti gli elementi contenuti in un altro oggetto, tipicamente un contenitore, senza doversi preoccupare dei dettagli di una specifica implementazione.

- hasNext restituisce true se esiste ancora un elemento della collezione da esaminare, false altrimenti;
- next restituisce il prossimo elemento da esaminare, e lancia un'eccezione se questo non esiste;
- remove cancella dalla collezione l'elemento restituito dall'ultima chiamata al metodo next, e può essere invocato una sola volta dopo ogni next.

Ogni classe che implementi l'interfaccia Iterable possiede infatti un set di metodi di supporto: un metodo listIterator() che crea l'iteratore, un metodo next() che lo fa avanzare ed un metodo hasNext() che verifica l'esistenza del nodo successivo ed opzionalmente un metodo remove().

```
//crea una lista concatenata che sarà visitata grazie all'iteratore

   LinkedList<String> lista = new LinkedList<String>();

   //supponiamo che la lista venga nel frattempo riempita

   ListIterator<String> it = lista.listIterator();

   //crea l'iteratore

   while (it.hasNext()) {

      //poiché il metodo next() restituisce un Object è possibile effettuare un
casting

      String temp = (String) it.next();

      System.out.println(temp);

   }
```

Enum

```
public enum Giorno {

    LUNEDI,

    MARTEDI,

    MERCOLEDI,

    GIOVEDI,

    VENERDI,

    SABATO,

    DOMENICA // opzionalmente può terminare con ";"

}

Giorno giornoDellaSettimana;
```

Mutability

Gli oggetti sono mutable o immutable
- Mutable – il loro stato può cambiare nel tempo – ad es. array, Studente, ...
- Immutable – il loro stato non cambia mai – ad es. Stringa, …
- Oggetto condiviso: – il suo riferimento è memorizzato in 2+ variabili

Output formatting

long n = 461012;
System.out.format("%d%n", n); // --> "461012"
System.out.format("%08d%n", n); // --> "00461012"
System.out.format("%+8d%n", n); // --> " +461012"
System.out.format("%,8d%n", n); // --> " 461,012"
System.out.format("%+,8d%n%n", n); // --> "+461,012"
double pi = Math.PI; System.out.format("%f%n", pi); // --> "3.141593" System.out.format("%.3f%n", pi); // --> "3.142"
System.out.format("%10.3f%n", pi); // --> " 3.142"
System.out.format("%-10.3f%n", pi); // --> "3.142 "
System.out.format(Locale.ITALY, "%-10.4f%n%n", pi); // --> "3,1416 "
Calendar c = Calendar.getInstance();
System.out.format("%tB %te, %tY%n", c, c, c); // --> "May 29, 2006" System.out.format("%tl:%tM %tp%n", c, c, c); // --> "2:34 am"
System.out.format("%tD%n", c); // --> "05/29/06"

Primitive types

- byte: 8-bit signed 2 complement integer
- char: 16-bit Unicode
- int: 32-bit signed 2 complement integer
- short: 16-bit signed 2 complement integer
- long: 64-bit signed 2 complement integer
- float: 32-bit floating point
- double: 64-bit floating point
- boolean: true/false

Conversioni

Number:

byte byteValue()

short shortValue()

int intValue()

long longValue()

float floatValue()

double doubleValue()

static Integer decode(String s)

static int parseInt(String s)

static int parseInt(String s, int radix)

String toString()

static String toString(int i)

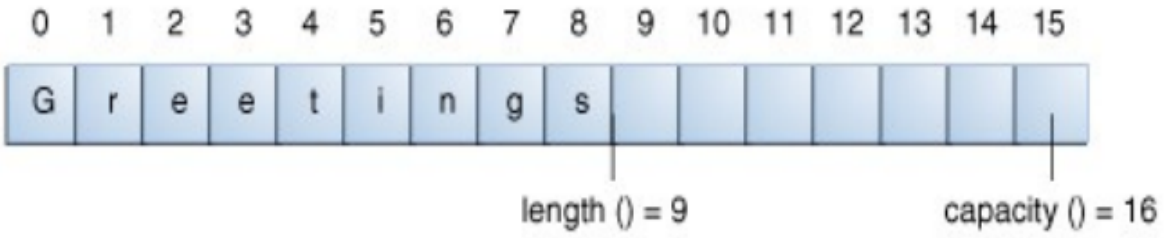
static Integer valueOf(int i)

static Integer valueOf(String s)

static Integer valueOf(String s, int radix)

StringBuilder

StringBuilder sb = new StringBuilder(); sb.append("Greetings");



- String è immutable
- StringBuilder è mutable

Operazioni sulle stringhe

Concatenazione di stringhe : string1+string2 oppure string1.concat(string2)

Confronti tra stringhe : if (string1.equals(string2)) {...}

Selezionare uno specifico carattere della stringa : s.charAt(i);

Dichiarazione di una variabile String : String str = "uno";

Dimensione della stringa : int n = stringa.length();

Substring: string1.substring(int beginIndex, int endIndex);

Metodi di confronto tra stringhe:

boolean equals(object anObject) confronta la stringa con la stringa indicata e restituisce true o false;
boolean equalsIgnoreCase(String anotherString) confronta la stringa con la stringa indicata ignorando la differenza tra maiuscole e minuscole e restituisce true o false;
boolean startsWith(String prefix) restituisce true se la stringa inizia con prefix;
boolean endsWith(String suffix) restituisce true se la stringa termina con suffix.
boolean isEmpty()verifica una stringa vuota

Conversione in stringa:

String s = Integer.toString(n);
String s = Double.toString(n);
String s = Long.toString(n);
String s = Float.toString(n);

Eliminare gli spazi bianchi in una stringa:

string1.trim()

Trovare un determinato carattere nella stringa:

int indexOf(char ch) Ritorna l'indice della prima occorrenza del carattere indicato.

Convertire una stringa in minuscolo:

string1.toLowerCase()

Convertire una stringa in maiuscolo:

string1.toUpperCase()

Sostituire caratteri:

string1.replace(char oldCh, char newCh)
Ritorna una nuova stringa dove tutte le occorrenze di oldCh sono sostituite con newCh.