

Basi di JavaeStrutture Dati

Lucio Benfante



Collegatevi alle slide

user: student

password: Pa\$\$w0rd

<http://192.168.16.114:4100>

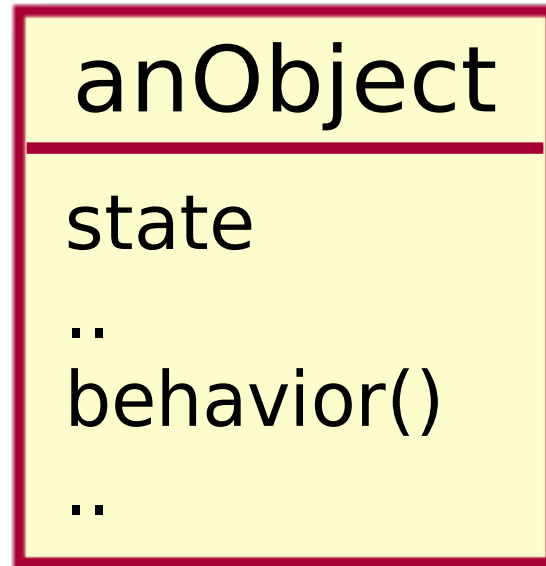
Slide prodotte con backslide <https://github.com/sinedied/backslide>

Programmazione ad Oggetti

(OOP - Object-Oriented Programming)

Gli oggetti

Sono il mattone fondamentale di un sistema software implementato mediante il paradigma OOP.



Encapsulation

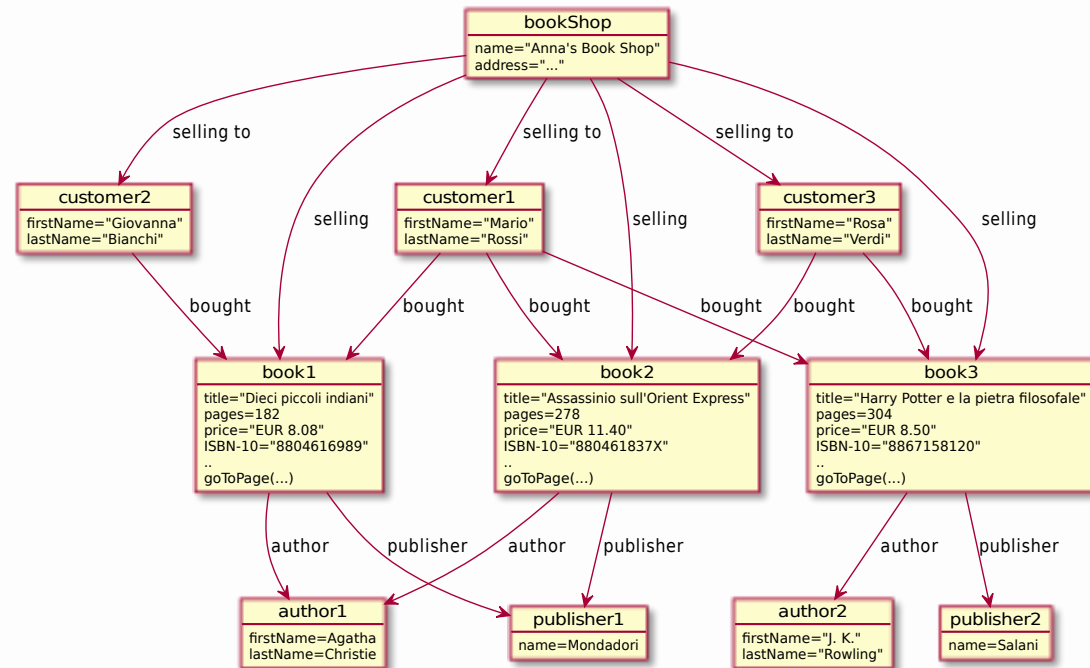
È la capacità di un oggetto di inglobare al suo interno il suo stato e il suo comportamento, cioè, tipicamente, i meccanismi che modificano tale stato, impedendone le manipolazioni non desiderate.

Information hiding

Impedendo l'accesso diretto alle parti interne degli oggetti, se ne impedisce la manipolazione indesiderata, e si permette inoltre la sostituzione di oggetti con altri che abbiano funzionamenti analoghi.

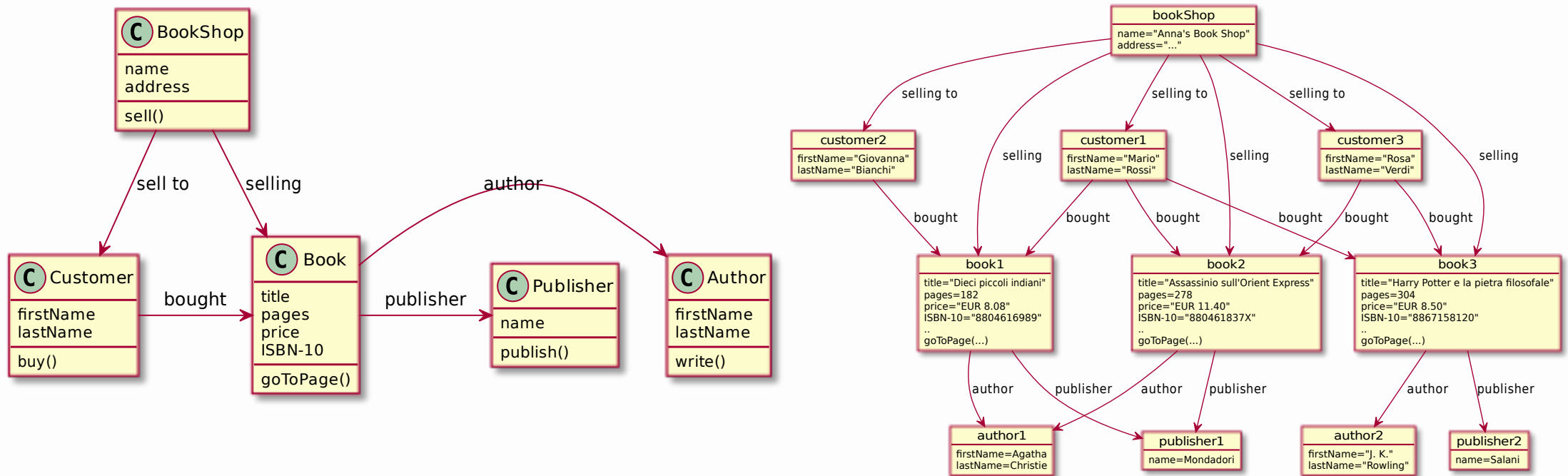
Software come aggregazione di oggetti

Componendo oggetti più semplici otteniamo astrazioni che ci permettono di descrivere sistemi complessi.



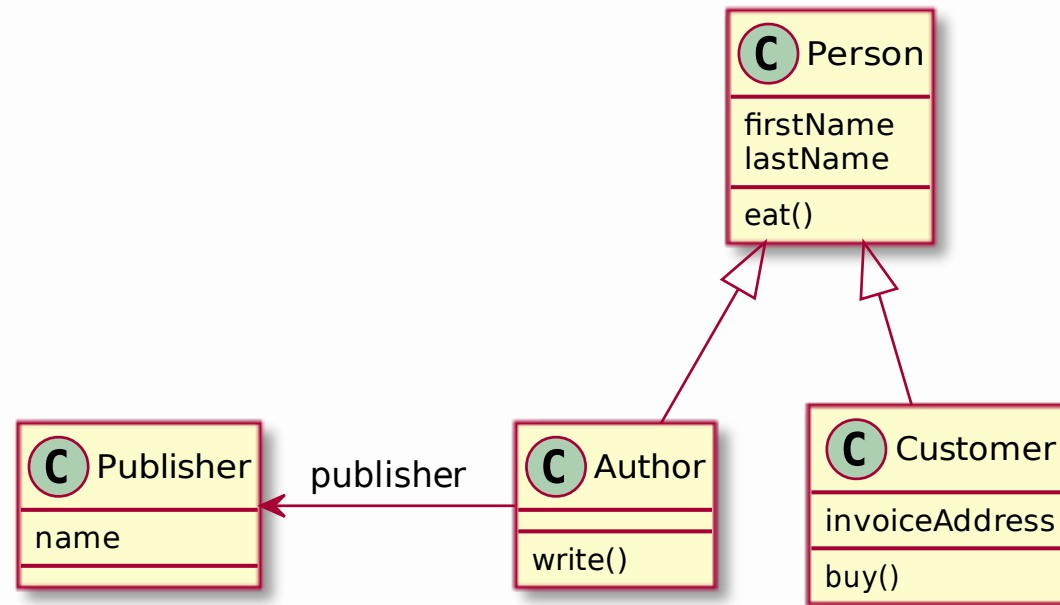
La classe

La classe costituisce il *progetto* che viene usato per la costruzione degli oggetti, che ne sono quindi casi (*instance*) particolari.



Inheritance (Ereditarietà)

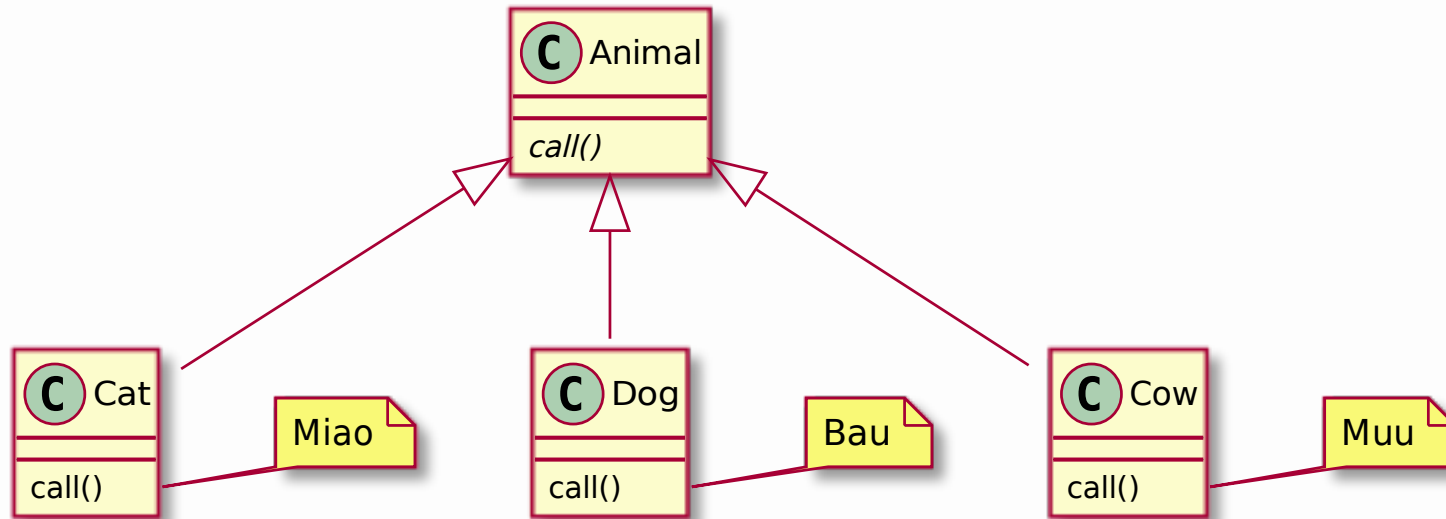
L'ereditarietà è il processo mediante il quale un oggetto acquisisce le caratteristiche di un altro oggetto. Diventa quindi possibile descrivere degli oggetti mediante delle astrazioni più generali, che possono essere riutilizzate più volte.



Polymorphism

Il polimorfismo è la capacità di eseguire una stessa azione in maniera diversa, a seconda dell'oggetto a cui viene applicata.

Ciò permette quindi di descrivere operazioni, la cui esatta esecuzione potrà variare ed evolvere in base ai particolari oggetti coinvolti.

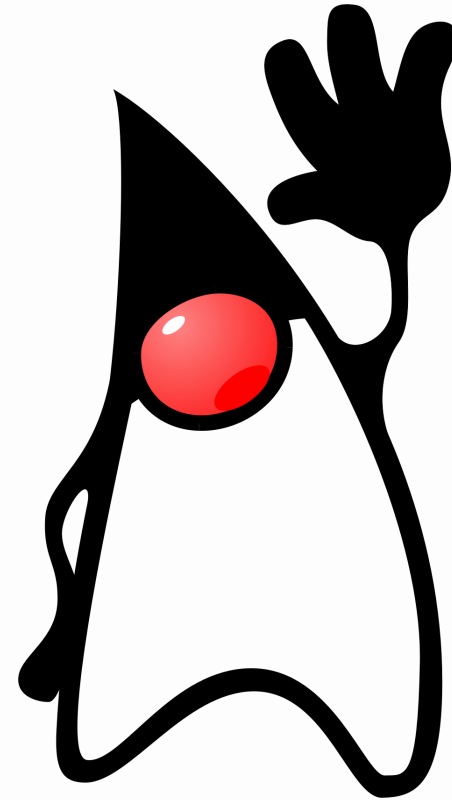


Introduzione a Java

Java



1990: Il Green Project ed il prototipo 7*



Il Linguaggio OAK

James Gosling

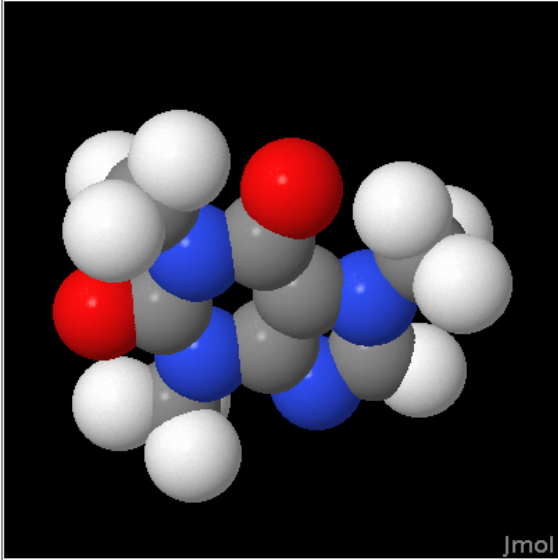
“A simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, high-performance, multithreaded, dynamic language.”

1995: Il browser WebRunner e le applet

jmol applet atoms demo x +

jmol.sourceforge.net/demo/atoms/ Cerca

caffeine



The RasMol scripting language has two synonymous commands to control atom size, **spacefill** and **cpk** (for Corey, Pauling, and Koltun). The commands are aliases and run exactly the same code.

- ☐ spacefill on or cpk on or spacefill 100%
- ☐ spacefill off or cpk off or spacefill 0 or spacefill 0%

As an extension to the RasMol scripting language, Jmol allows you to specify your size as a percentage of the vanderWaals radius.

- ☐ spacefill 20% or cpk 20%
- ☐ spacefill 25% or cpk 25%
- ☐ spacefill 50% or cpk 50%
- ☐ spacefill 75% or cpk 75%

If you want to specify a fixed size then you can do so in Angstroms. Note that the last button specifies "1.0" instead of the integer "1". This is important because integer values specify *RasMol units* (1/250th Angstrom)

- ☐ spacefill 0.25 or cpk 0.25
- ☐ spacefill 0.5 or cpk 0.5
- ☐ spacefill 0.75 or cpk 0.75
- ☒ spacefill 1.0 or cpk 1.0

<http://jmol.sourceforge.net/demo/atoms>

Un po' di storia

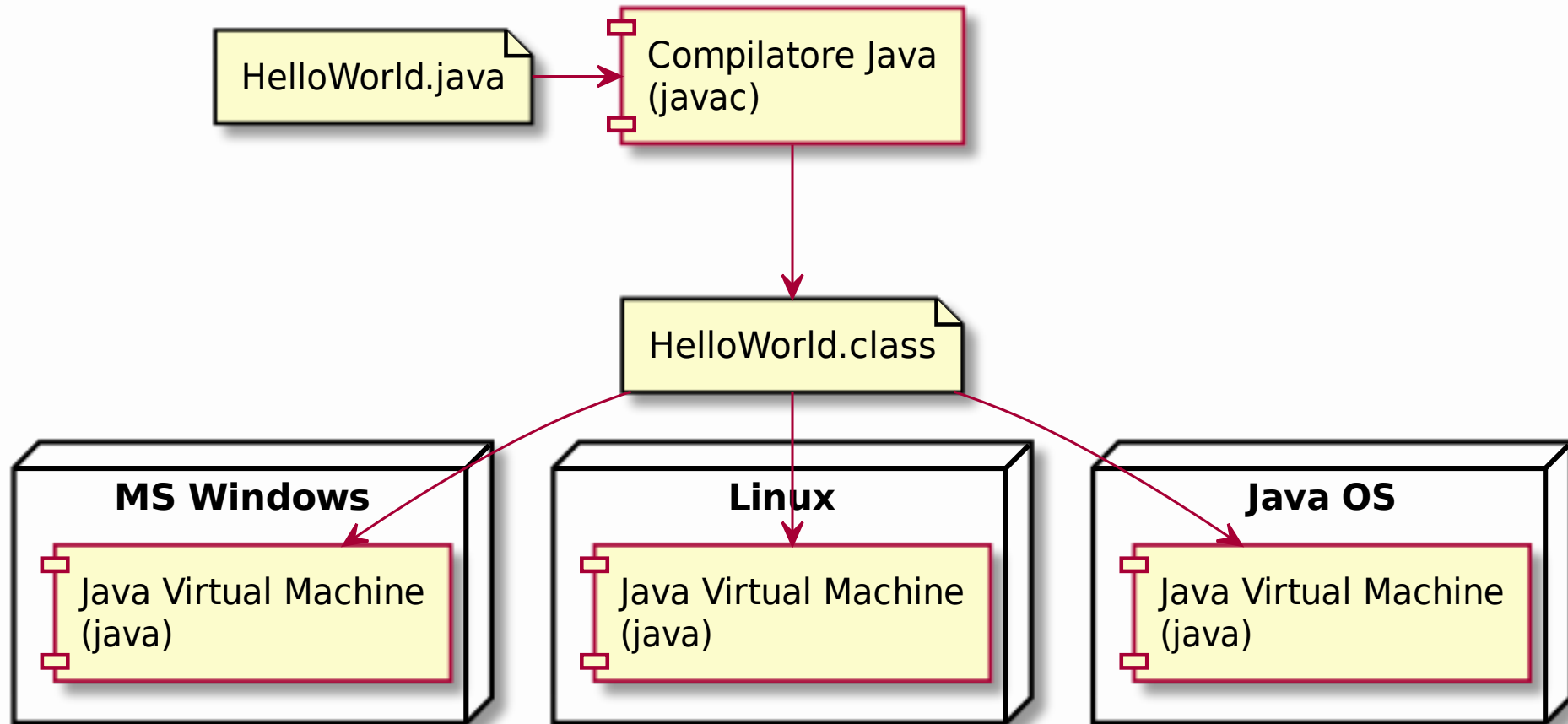
- 1996: Java Development Kit (JDK) 1.0
- 1996: JavaOne Developer Conference (5000 partecipanti)
- 1997: JDK 1.1
- 1998: Java Community Process (JCP)
- 1998: JDK 1.2 (J2SE, Java 2)
- 1999: JavaOne Conference (20.000 partecipanti)
- 1999: Standard Edition, Enterprise Edition, Micro Edition
- 2000: J2SE 1.3
- 2000: J2SE 1.4
- 2004: J2SE 5.0 (Java 5)
- 2005: Java nei lettori Blu-ray
- 2006: Java SE 6
- 2006: Java Opensource
- 2007: (Android)
- 2009: Acquisizione Sun/Oracle per 7.4 miliardi di dollari
- 2011: Java SE 7
- 2014: Java SE 8
- 2017: Java SE 9

Write Once, Run Everywhere

Tecnologia Java

- Linguaggio di programmazione
- Ambiente di sviluppo
- Ambiente applicativo
- Ambiente di installazione e distribuzione

Tecnologia Java



Java Virtual Machine

“La macchina virtuale Java, detta anche Java Virtual Machine o JVM, è il componente della piattaforma Java che esegue i programmi tradotti in bytecode dopo una prima compilazione.”

http://it.wikipedia.org/wiki/java_virtual_machine

Java Virtual Machine

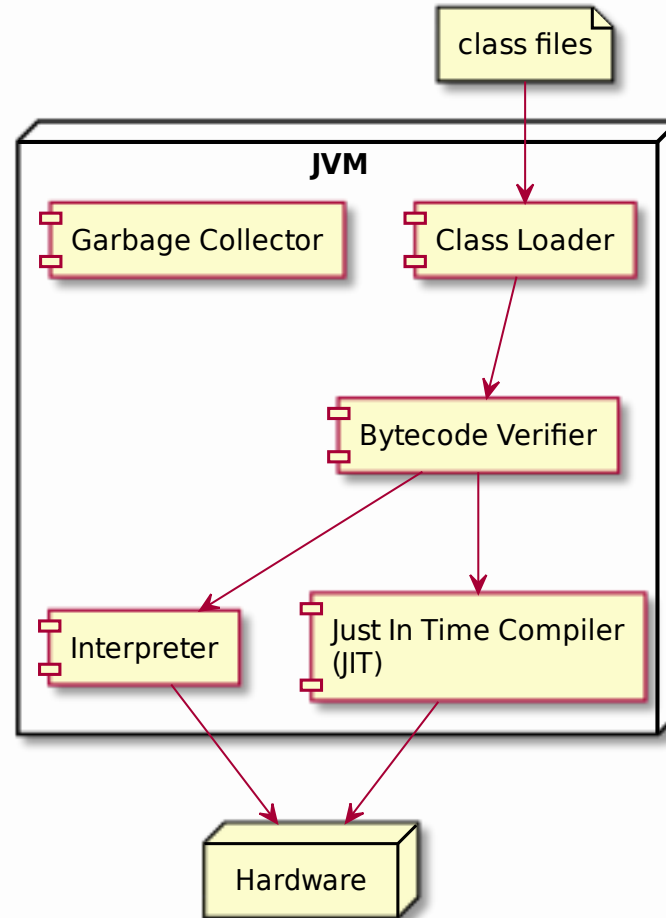
È una specifica hardware usualmente realizzata in software:

- Insieme di istruzioni
- Insieme di registri
- Formato dei file .class
- Stack
- Memoria heap gestita da un garbage collector
- Area di memoria per costanti, metodi, ecc.

Gestione della memoria

- La memoria non più necessaria deve essere liberata.
- Nella maggior parte dei linguaggi è compito del programmatore.
- In Java viene gestita automaticamente da un thread di sistema (Garbage Collector).

Java Virtual Machine



Tool per sviluppatori

- Editor di testi
- Compilatore Java
- Esecutore Java
- (Debugger)
- (Sistema per il versionamento dei sorgenti)

Editor + Compilatore + Esecutore + Debugger + Versionamento + ecc.
=
Integrated Development Environment (IDE)

Quale scegliere?

- NetBeans (www.netbeans.org)
- Eclipse (www.eclipse.org)
- IntelliJ IDEA (<http://www.jetbrains.com/idea/>)

Programmazione Java

Hello World

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

Compilazione

```
javac HelloWorld.java
```

Viene prodotto il file HelloWorld.class, che contiene il bytecode della classe.

Esecuzione

```
java HelloWorld
```

Output:

```
Hello World!
```

Esercizi

- Scrivere, compilare ed eseguire il programma HelloWorld da riga di comando.

Passare parametri al programma

```
public class HelloWorld1 {  
    public static void main(String[] args) {  
        System.out.println("Hello "+args[0]+"!");  
    }  
}
```

Esecuzione:

```
java HelloWorld1 Lucio
```

Output:

```
Hello Lucio
```

Usare delle Properties

```
public class HelloWorld2 {  
    public static void main(String[] args) {  
        String s = System.getProperty("name");  
        System.out.println("Hello "+s+"!");  
    }  
}
```

Esecuzione:

```
java -Dname=Lucio HelloWorld2
```

Output:

```
Hello Lucio
```

Commenti

Mono-riga

```
// Questo è un commento.  
System.out.println("Hello "+name+"!"); // Ed eccone un altro.
```

Multi-riga

```
/*  
    Tutto questo  
    è un commento.  
*/
```


Commenti di documentazione

```
/**
 * Some utilities for manipulating URLs.
 *
 * @author Lucio Benfante
 */
public class UrlHelper {

    /**
     * Append a parameter to an URL.
     *
     * @param sb The StringBuilder containing the URL where to append.
     * @param parameterName The name of the parameter
     * @param parameterValue The value of the parameter. It will be url encoded.
     * @param ifNotNull if true the parameter is appended only if its vale is not null.
     * @return The passed StringBuilder
     * @throws java.io.UnsupportedEncodingException
     */
    public static StringBuilder appendUrlParameter(StringBuilder sb,
        String parameterName, String parameterValue, boolean ifNotNull)
        throws UnsupportedEncodingException {
```

Il tool `javadoc`

Serve per generare la documentazione a partire dai sorgenti:

```
javadoc -d docs HelloWorld.java
```

Aprire il browser con il file `docs/index.html`

Anche la documentazione della "Standard API" è generata con `javadoc`:

<https://docs.oracle.com/javase/9/docs/api>

Le keyword del linguaggio Java

| | | | | |
|-----------------|-----------------|-------------------|------------------|---------------------|
| abstract | continue | for | new | switch |
| assert | default | goto | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

La classe e i suoi membri

È un template per gli oggetti.

```
[ModificatoreAccesso] class NomeClasse {  
    // Attributi  
    [ModificatoreAccesso] definizioneAttributo;  
    ...  
  
    // Costruttori  
    [ModificatoreAccesso] NomeClasse([Parametri]) {}  
    ...  
  
    // Metodi  
    [ModificatoreAccesso] tipoRitorno nomeMetodo([Parametri]) {}  
}
```

L'operatore **new**

Serve per costruire gli oggetti dalle classi.

```
Libro libro = new Libro();
```

L'operatore

Serve per accedere/utilizzare i membri delle classi/oggetti.

```
int p = libro.getPagine();
```

Package (1)

I package sono un modo per organizzare le classi ed evitare le collisioni fra i loro nomi.

Hanno inoltre un ruolo nella accessibilità delle classi e dei loro membri con i modificatori di accesso `protected` e quello di default (senza nessun modificatore di accesso).

```
package com.corsojava.biblioteca;  
  
public class Libro { /* codice della classe Libro */ }
```

```
/src/com/corsojava/biblioteca/Libro.java
```

Package (2)

Ora il nome completo della classe comprende il package di cui fa parte.

```
com.corsojava.biblioteca.Libro libro =  
    new com.corsojava.biblioteca.Libro(120);
```

```
java.lang.String s = "Lucio Benfante"
```


import

```
import com.corsojava.biblioteca.Libro;  
  
...  
  
Libro libro = new Libro(120);
```

Per importare tutte le classi presenti in un package:

```
import com.corsojava.biblioteca.*;  
  
...  
  
Libro libro = new Libro(120);
```

Il package `java.lang.*` viene automaticamente importato.

Modificatori d'accesso

| | private | nessun modificatore(default) | protected | public |
|-----------------------------------|----------------|-----------------------------------------|------------------|---------------|
| Stessa classe | sì | sì | sì | sì |
| Stesso package (sottoclasse) | no | sì | sì | sì |
| Stesso package (non sottoclasse) | no | sì | sì | sì |
| Diverso package (sottoclasse) | no | no | sì | sì |
| Diverso package (non sottoclasse) | no | no | no | sì |

Identificatori

- Sono i nomi che diamo alle classi, alle variabili (anche agli attributi) e ai metodi.
- Si può usare qualunque carattere di qualunque alfabeto.
- Il primo carattere di un identificatore può essere una qualunque lettera, il carattere underscore (_), o il simbolo \$.
- Sono case-sensitive e non hanno limite di lunghezza.

Convenzioni per gli identificatori

- Package: nomi tutti in minuscolo (`java.util`)
- Classi e Interfacce: nomi concatenati con la prima lettera in maiuscolo (`StringBuilder`)
- Metodi: azioni con la prima lettera minuscola, e le altre parole concatenate con la prima in maiuscolo (`setTitle()`)
- Variabili e Attributi: parole concatenate, con la prima lettera minuscola e le altre concatenate con la prima in maiuscolo (`birthdayDate`)
- Costanti: parole tutte in maiuscolo, separate da `_` (`MAX_SIZE`)

I tipi primitivi

| | | |
|----------------|---------------------------------------------|--------------------------------------|
| boolean | true/false | Valori booleani |
| char | es.: 'a', '\n' | Caratteri Unicode a 16-bit |
| byte | - (2 ⁷) a (2 ⁷)-1 | Interi a 8 bit |
| short | - (2 ¹⁵) a (2 ¹⁵)-1 | Interi a 16 bit |
| int | - (2 ³¹) a (2 ³¹)-1 | Interi a 32 bit |
| long | - (2 ⁶³) a (2 ⁶³)-1 | Interi a 64 bit |
| double | es.: 321.5E+306 | Floating-point a 64 bit |
| float | es.: 3.14F | Floating-point a 32 bit |
| (String | es.: "Hello World!" | Stringa di caratteri Unicode 16-bit) |

Variabili locali

- Le variabili locali sono variabili definite all'interno di un metodo, e i parametri di metodi e costruttori.
- Vengono create quando l'esecuzione entra nel metodo che le definisce, e distrutte quando ne esce.
- Devono essere esplicitamente inizializzate.
- Essendo locali al metodo che le definisce, lo stesso nome può essere usato in metodi diversi.
- Nel caso il nome si sovrapponga a quello di una variabile di istanza, si usa il reference `this` per disambiguarlo.

Inizializzazione di variabili

```
int a, b, d, e, f;  
a = 12;  
b = 0b010111; // from Java 7  
d = 0xC4;  
e = 077;  
//      a = f;  // Error: variables must be initialized.
```

Inizializzazione di variabili

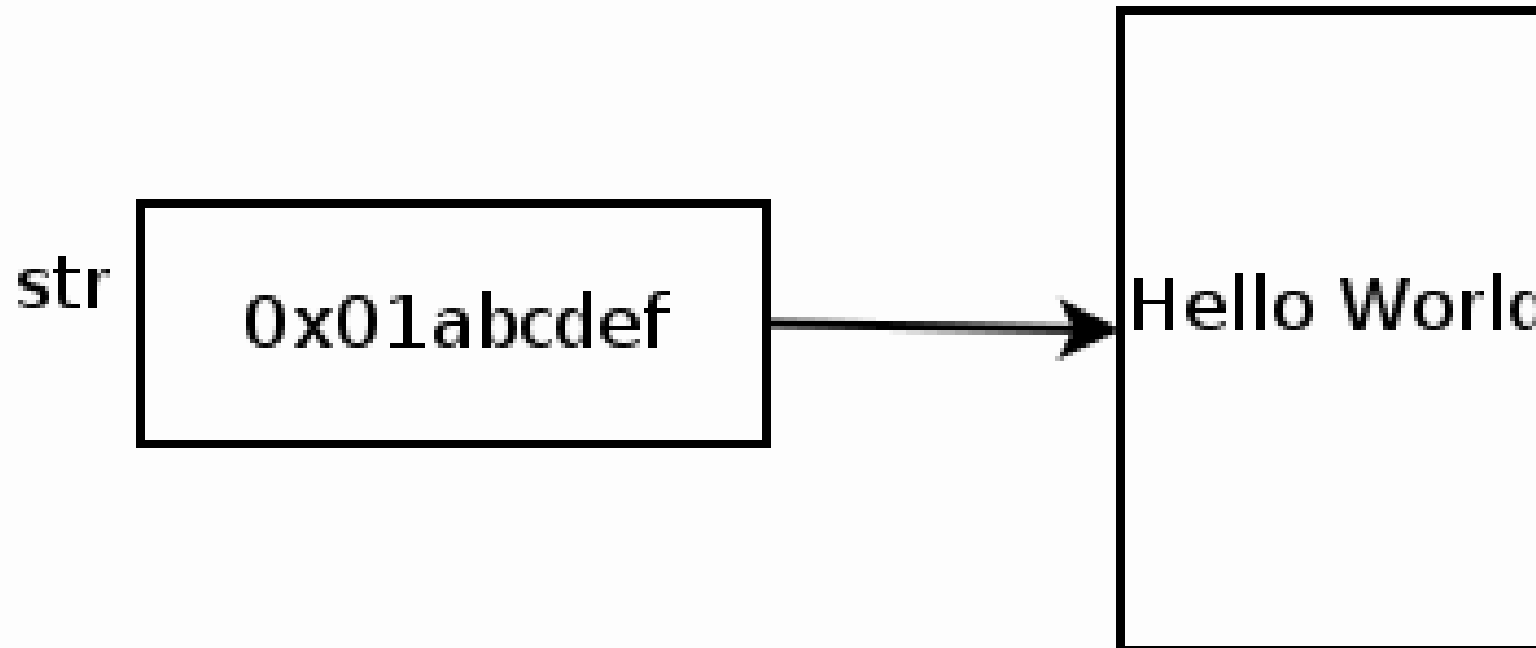
```
long l = 345L;  
long m = 123_456_789L; // from Java 7  
  
float x = 10.254f;  
  
double y = 3.1415;  
  
boolean check = true;  
  
char c;  
String str = "Hello developer!";  
c = 'a';
```


Valori di default per i vari tipi

| Tipo | Valore |
|----------------|----------|
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0F |
| double | 0.0 |
| char | '\u0000' |
| boolean | false |
| Tipi reference | null |

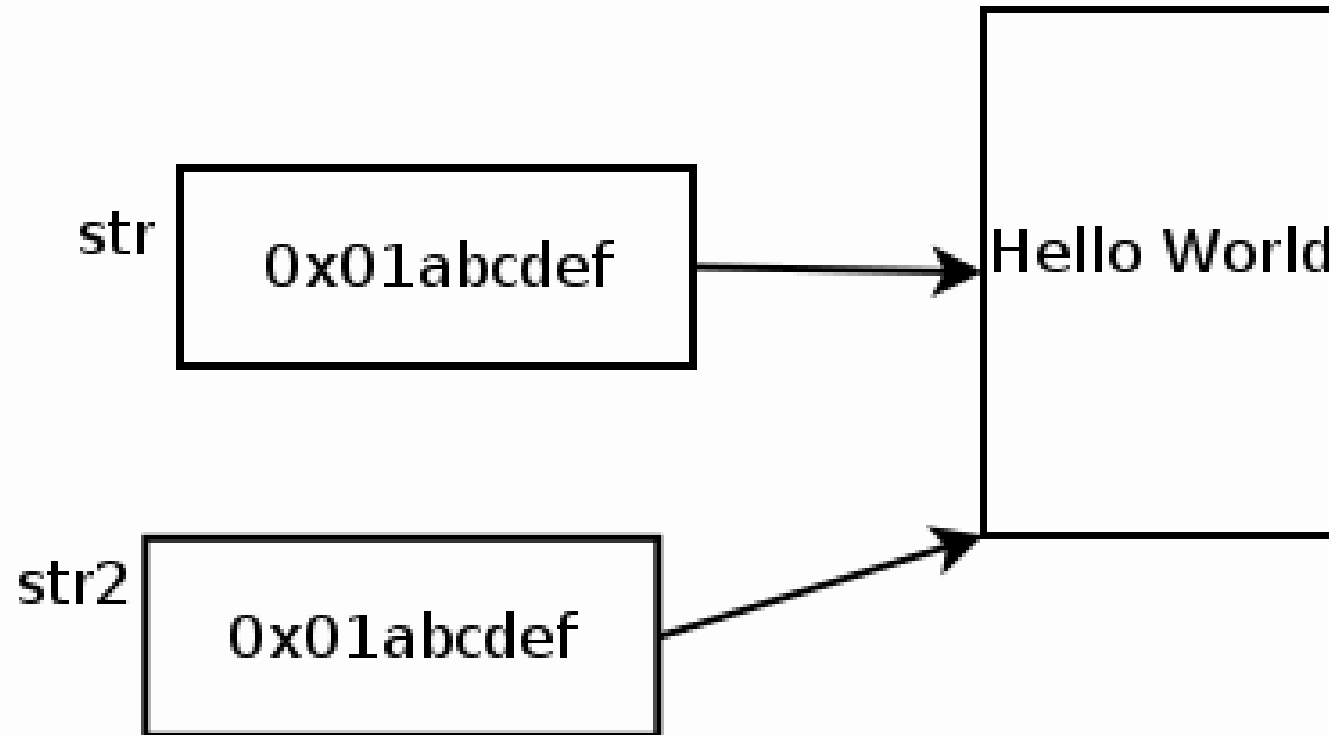
Tipi riferimento (reference)

```
String str = "Hello World";
```



Assegnazioni con tipi riferimento

```
String str2 = str;
```



Passaggio di oggetti

```
public static void printPerson(Person p ,int eta) {  
    System.out.println(p.getFirstName());  
    p.setFirstName("Mario");  
    eta = 25;  
    p = new Person("Mario", "Rossi");  
}
```

```
// ...  
Person person = new Person("Lucio" , "Benfante");  
int e = 50  
printPerson(person, e);
```

Operatori

- Operatori aritmetici: + - * / ++ -
- Operatori di confronto: == > < >= > >> < java EsaminaStringa pippo

Lunghezza: 5 Primo carattere: p Ultimo carattere: o Maiuscolo: PIPPO

Esercizi

Scrivere un programma che assegni valori a variabili di tutti i tipi primitivi
Sperimentare con i membri della classe `java.lang.Math`, ad esempio:

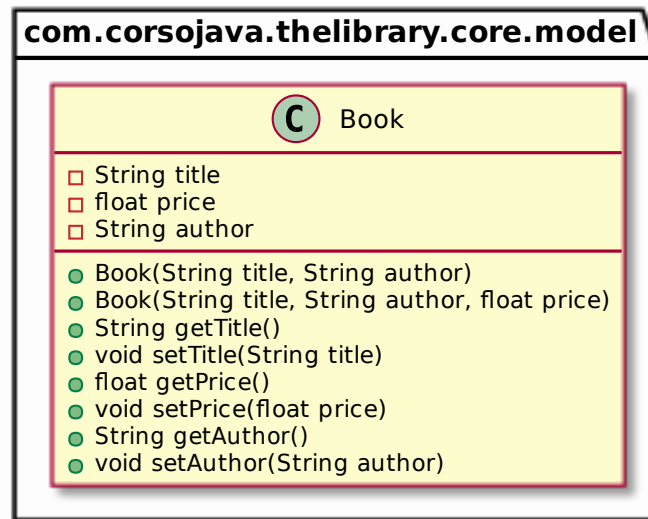
```
* `Math.PI`  
* `Math.max(a, b)`  
* `Math.round(a)`  
* `Math.random()`
```

Maven - Build

```
mvn clean install
```

Esercizio

- Importare in Eclipse il progetto `thelibrary-core`
- Aggiungere il package `com.corsojava.thelibrary.core.model`
- Aggiungere in tale package la classe `Book` con le seguenti caratteristiche:



- Aggiungere la classe `com.corsojava.thelibrary.core.app.Library`, contenente il metodo `main` all'interno del quale si creino alcuni oggetti di tipo `Book`, e se ne stampino a schermo le caratteristiche.

Array

Un array è un gruppo di variabili dello stesso tipo, identificate da un singolo nome, ed accessibili singolarment mediante un indice.

Definizione

```
Tipo nomeVariabile[];  
Tipo[] nomeVariabile;
```

Inizializzazione

```
nomeVariabile = new Tipo[dimensione]
```


Array - Esempio

```
int giorniDelMese[] = new int[12];
```

Uso

```
giorniDelMese[0] = 31;  
giorniDelMese[1] = 28;  
giorniDelMese[2] = 31;  
giorniDelMese[3] = 30;  
...
```

```
System.out.println("Gennaio ha "+giorniDelMese[0]+" giorni.")
```

Array - Inizializzazione compatta

```
int giorniDelMese[] = {31, 28, 31, 30,  
                       31, 30, 31, 31,  
                       30, 31, 30, 31};
```

Array multidimensionali

```
int matrix[][] = new int[4][5];  
  
matrix[0][2] = 20;
```

Anche non rettangolari

```
int notRect[][] = new int[4][];  
notRect[0] = new int[2];  
notRect[1] = new int[5];  
notRect[2] = new int[4];  
notRect[3] = new int[10];
```

Controllo del flusso - if

```
if (condizione) statement1;  
    else statement2;
```

Esempio

```
if (a == 2) {  
    // fai qualcosa se a è uguale a 2  
} else {  
    // fai qualcos'altro altrimenti  
}
```

Attenzione!

```
if (a == 2)
    statement1;
    statement2;
```

Meglio:

```
if (a == 2) {
    statement1;
}
statement2;
```

if nidificati

```
if (condizione1)
    statement1;
else if (condizione2)
    statement2;
else if (condizione3)
    statement3;
else
    statementN;
```

switch

```
switch (espressione) {  
    case valore1:  
        SequenzaDiStatement;  
        break;  
    case valore2:  
        SequenzaDiStatement;  
        break;  
    case valore3:  
        SequenzaDiStatement;  
        break;  
    default:  
        SequenzaDiStatement;  
        break;  
}
```

Switch - Esempio

```
switch (a) {  
    case 1:  
    case 2:  
    case 3:  
        System.out.println("Primo blocco");  
        break;  
    case 4:  
    case 5:  
        System.out.println("Secondo blocco");  
        break;  
    default:  
        System.out.println("Blocco di default");  
}
```


Operatore condizionale ternario

condizione?valorePerTrue:valorePerFalse

```
String s = (a > 0)? "a è positivo": "a non positivo";
```

Cicli iterativi - while

```
while (condizione) {  
    IstruzioniDelLoop  
}
```

Esempio

```
int i = 0;  
while(i < 10) {  
    System.out.println(i);  
    i++;  
}
```

Cicli iterativi - do-while

```
do {  
    IstruzioniDelLoop  
} while (condizione);
```

Cicli iterativi - for

```
for (inizializzazione;condizione;iterazione) {  
    IstruzioniDelLoop  
}
```

Esempio

```
for (int i=0; i< a[i].length) {  
    break outer;  
}  
if (j == (a.length - i)) {  
    break;  
}  
a[i][j] = i*j;  
}  
}
```

return

Permette di uscire da un metodo, restituendo eventualmente un valore:

```
public int getEtà() {  
    return età;  
}
```

Esercizi Cicli (1)

- Usando un ciclo, si stampino a schermo le lettere dell'alfabeto minuscole e maiuscole. Si noti che `'b' = 'a' + 1`, `'c' = 'a' + 2`, e così via.

| | |
|-----------|----------|
| 'a' = 97 | 'A' = 65 |
| 'b' = 98 | 'B' = 66 |
| 'c' = 99 | 'C' = 67 |
| 'd' = 100 | 'D' = 68 |
| 'e' = 101 | 'E' = 69 |
| 'f' = 102 | 'F' = 70 |
| 'g' = 103 | 'G' = 71 |
| 'h' = 104 | 'H' = 72 |
| 'i' = 105 | 'I' = 73 |
| 'j' = 106 | 'J' = 74 |
| 'k' = 107 | 'K' = 75 |
| 'l' = 108 | 'L' = 76 |
| 'm' = 109 | 'M' = 77 |
| 'n' = 110 | 'N' = 78 |
| 'o' = 111 | 'O' = 79 |
| 'p' = 112 | 'P' = 80 |
| 'q' = 113 | 'Q' = 81 |
| 'r' = 114 | 'R' = 82 |
| 's' = 115 | 'S' = 83 |
| 't' = 116 | 'T' = 84 |
| 'u' = 117 | 'U' = 85 |
| 'v' = 118 | 'V' = 86 |
| 'w' = 119 | 'W' = 87 |
| 'x' = 120 | 'X' = 88 |
| 'y' = 121 | 'Y' = 89 |
| 'z' = 122 | 'Z' = 90 |

Esercizi Array e Cicli (1)

- Scrivere un metodo che produca un array bidimensionale di interi, in cui ogni elemento abbia come valore il prodotto dei suoi indici (+1).

```
public int[][] buildTable(int dim)
```

- Scrivere un metodo che stampi la tabella creata dal metodo precedente:

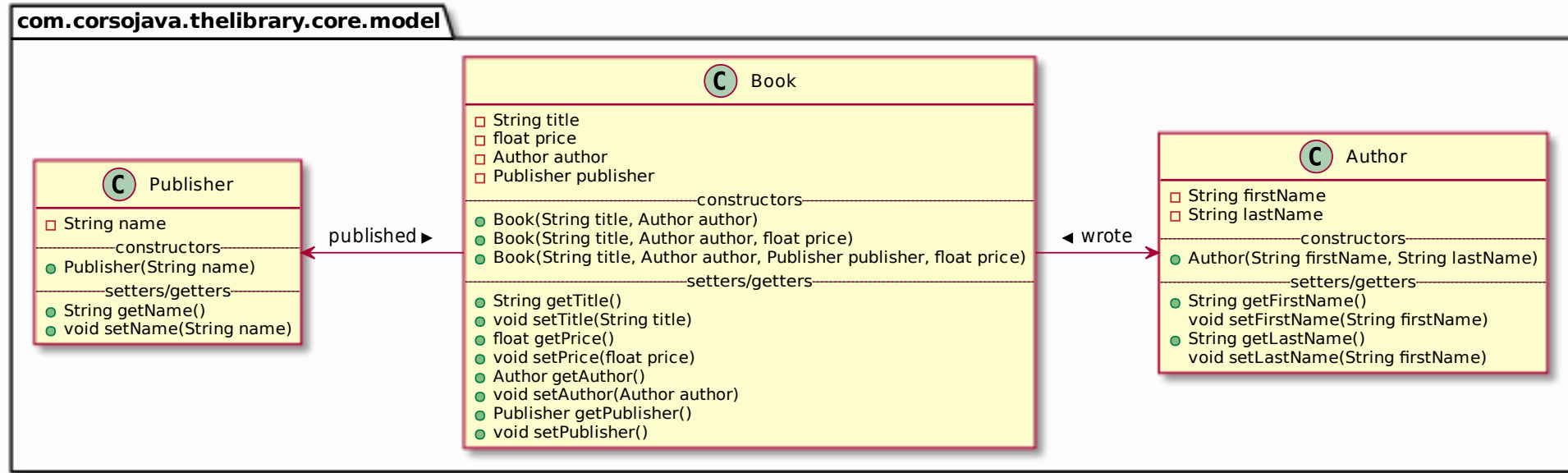
```
public void print(int table[][])
```

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|-----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

Input da tastiera

```
Scanner scanner = new Scanner(System.in);
System.out.print("Nome: ");
String nome = scanner.nextLine();
System.out.print("Cognome: ");
String cognome = scanner.nextLine();
System.out.print("Età: ");
int eta = scanner.nextInt();
scanner.nextLine();
System.out.print("Città: ");
String citta = scanner.nextLine();
scanner.close();
System.out.printf("Nome: %s, Cognome: %s, Età: %d, Città: %s",
    nome, cognome, eta, citta);
```


Es.TheLibrary - Legami fra oggetti



```
Author author = new Author("Agatha", "Christie");
Publisher publisher = new Publisher("Mondadori");
Book book = new Book("Dieci Piccoli Indiani", author, publisher, 10.5f);
```

Array di Oggetti

```
Libro libri[] = new Libro[2];

libri[0] = new Libro(1344,
    "Java: The Complete Reference, Tenth Edition",
    "Herbert Schildt");
libri[1] = new Libro(776,
    "Il linguaggio Java, Manuale Ufficiale",
    "Ken Arnold, James Gosling, David Holmes");
```

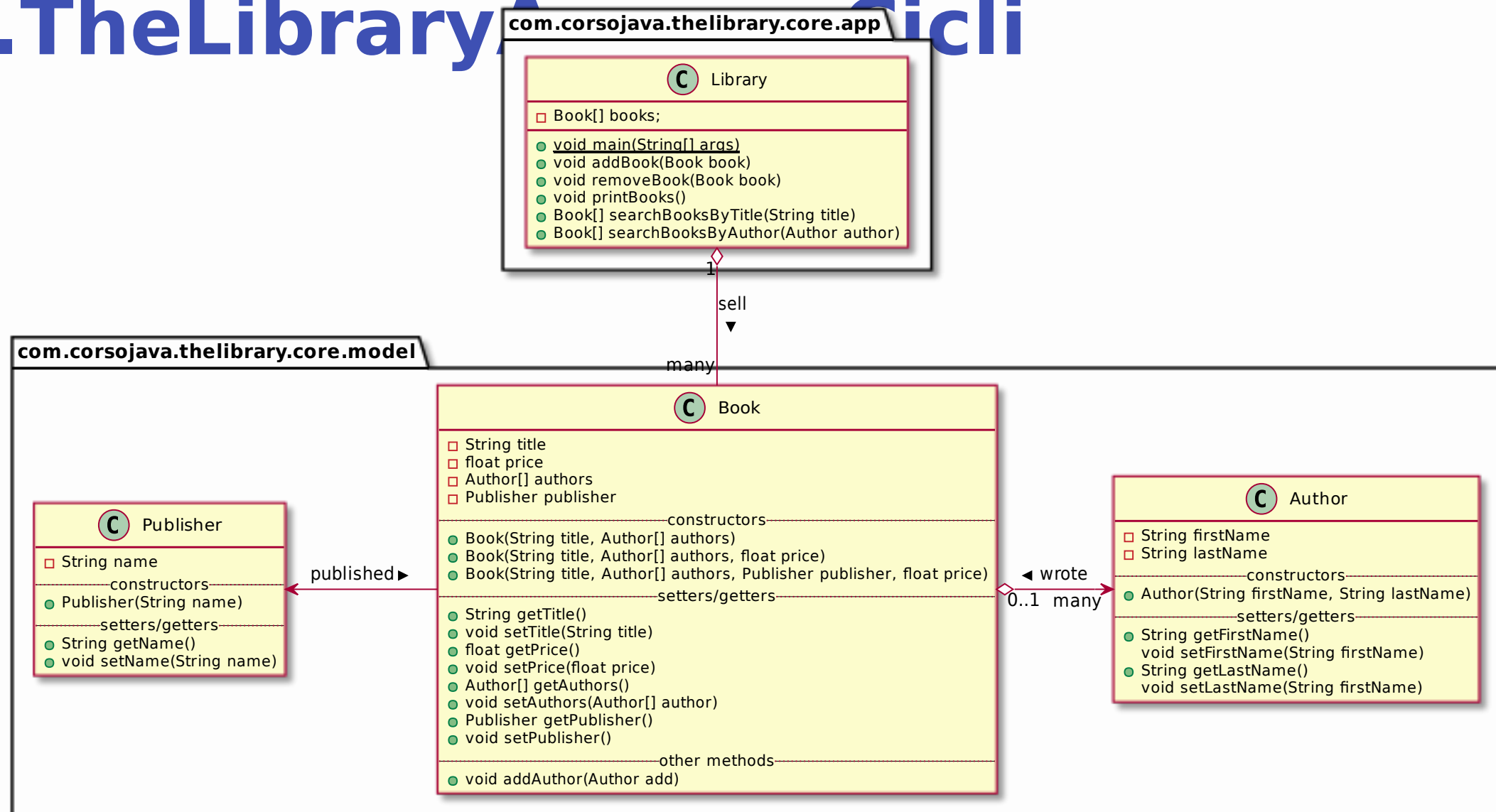
```
Libro libri[] = new Libro[] {new Libro(1344,
    "Java: The Complete Reference, Tenth Edition",
    "Herbert Schildt"),
    new Libro(776, "Il linguaggio Java, Manuale Ufficiale",
    "Ken Arnold, James Gosling, David Holmes")}
```

Es.TheLibrary - Array e Cicli

- Si modifichi la classe `Book` in modo che sia possibile avere più di un autore (si sostituisca l'attributo `author` e si utilizzi un array di `Author` per il nuovo attributo `authors`)
- Oltre ai metodi di accesso (set/get) opportuni, si aggiunga alla classe `Book` un metodo `addAuthor` che aggiunge un autore al libro, ridimensionandone prima l'array che dovrà contenerlo.
- Si aggiunga alla classe `Library` la capacità di memorizzare i libri in un array, con i relativi metodi per:
 - aggiungere un libro
 - rimuovere un libro
 - stampare a schermo l'elenco dei libri
 - cercare i libri per titolo
 - cercare i libri per autore
- Si scriva nel `main` di `Library` il codice per provare le funzionalità aggiunte.

Slide successiva per il diagramma delle classi -->

Es.TheLibrary



Programmazione ad oggetti

Overloading dei metodi

I metodi di una classe possono avere lo stesso nome, ma distinguersi per il tipo ed il numero di parametri. Tipo e numero dei parametri vengono quindi usati per decidere quale metodo overloaded invocare.

```
void test() { /* codice */ }  
void test(int a) { /* codice */ }  
void test(int a, int b) { /* codice */ }  
void test(int a, BigDecimal b) { /* codice */ }  
  
...  
  
test();  
test(10);  
test(10, 20);
```

Overloading dei costruttori

Anche i costruttori possono essere overloaded, quindi è possibile costruire oggetti passando parametri differenti, attivando di conseguenza costruttori differenti a seconda del tipo e numero di parametri passati.

```
public Libro(int numeroPagine) { /* codice */ }  
public Libro(int numeroPagine, String titolo) { /* codice */ }  
public Libro(int numeroPagine, String titolo,  
            String autore) { /* codice */ }  
  
...  
  
Libro l1 = new Libro(120);  
Libro l2 = new Libro(120, "Java: Complete...");  
Libro l3 = new Libro(120, "Java: Complete...", "Herbert...");
```

Uso di **this** per chiamare altri costruttori

```
public Libro(int numeroPagine) {  
    this.numeroPagine = numeroPagine;  
}  
  
public Libro(int numeroPagine, String titolo, String autore) {  
    this(numeroPagine);  
    this.titolo = titolo;  
    this.autore = autore;  
}
```


static

Il modificatore `static` permette di definire membri della classe che non appartengono ad una particolare istanza della classe, ma alla classe stessa.

Non esiste una copia di tali membri per ogni istanza della classe, ma un'unica copia comune a tutte le istanze.

L'accesso a tali membri di solito avviene usando il nome della classe (ma nulla vieta di usare un reference ad un oggetto della classe):

```
Libro.MAX_PAGINE
```

Restrizioni

- i metodi statici posso accedere esclusivamente ad altri membri (metodi o attributi) statici della classe. Non possono usare membri non statici.
- i metodi statici non possono usare i reference `this` o `super`.

Esempio

```
public class ContatoreIstanze {  
    private static int contatore = 0;  
  
    public ContatoreIstanze() {  
        contatore++;  
    }  
  
    public static void main(String[] args) {  
        ContatoreIstanze a = new ContatoreIstanze();  
        ContatoreIstanze b = new ContatoreIstanze();  
        ContatoreIstanze c = new ContatoreIstanze();  
        ContatoreIstanze d = new ContatoreIstanze();  
        System.out.println(ContatoreIstanze.contatore);  
        System.out.println(a.contatore);  
    }  
}
```

Inizializzazione in blocchi **static**

```
public class EsempioBloccoStatic {  
    static int a = 3;  
    static int b;  
  
    static {  
        /* Codice complicato... */  
        b = a * 4;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(b);  
    }  
}
```

final (su variabili)

Una variabile può essere definita usando il modificatore **final**. Una variabile **final** non può essere modificata dopo la sua inizializzazione.

In particolare viene usato per definire delle costanti:

```
public class Math {  
    public static final float PI = 3.142128;  
}  
  
...  
  
System.out.println(Math.PI);  
  
final Libro l = new Libro(...);
```

Ereditarietà e Polimorfismo

Ereditarietà

L'ereditarietà è il meccanismo per creare gerarchie di classi. Lo scopo è quello di definire classi (*sottoclassi*) che ereditano le loro caratteristiche di base da un'altra classe (*superclasse*), e ne aggiungono altre di proprie.

Una sottoclasse eredita dalla propria superclasse tutti gli attributi e i metodi non statici della propria superclasse.

Valgono le regole di accessibilità già viste.

Ereditarietà - sintassi

```
public class SottoClasse extends SuperClasse
```

Esempio:

```
public class A {  
    protected int a,b;  
}  
  
public class B extends A {  
    int c;  
  
    public int somma() { return a + b + c;}  
}
```


Uso dei reference

```
B b = new B(); //ovviamente
```

```
A a = b; /* Posso usare un oggetto di una sottoclasse  
           ovunque sia richiesto un oggetto della superclasse */
```

```
B c = (B)a; /* Solo se sono certo che il reference a stia indicando  
           un oggetto di tipo B. */
```

```
if (a instanceof B) {  
    B c = (B)a;  
}
```

Uso di `super()` nei costruttori

Usando `super(parametri)` nella prima riga di un costruttore si invoca il costruttore della superclasse con i parametri richiesti dal costruttore stesso.

In realtà ciò viene sempre fatto: se all'inizio di un costruttore non è presente `super(parametri)` o `this(parametri)`, viene invocato il costruttore di default (quello senza parametri) della superclasse (`super()`).

Se la superclasse non definisce un costruttore di default, nella sottoclasse è obbligatorio usare `super(parametri)` per indicare quale costruttore della superclasse invocare.

Overriding dei metodi

Una sottoclasse può ridefinire i metodi di una superclasse, cambiandone di fatto il comportamento.

L'overriding avviene solo se la *signature* dei metodi è identica (nome, tipo e numero dei parametri), altrimenti non si tratta di *overriding*, ma di *overloading*.

Overriding dei metodi - Esempio

```
public class A {  
    int a = 10;  
  
    void stampa() {  
        System.out.println("a="+a);  
    }  
}  
  
public class B extends A {  
    int b = 20;  
  
    void stampa() {  
        System.out.println("a="+a)+" , b="+b);  
    }  
}  
  
A[] giornali = new A[] {new B(), new A(), new B()};  
for(A a: giornali) { System.out.println(a.stampa());}
```

Esempio

```
public class ClassA {  
    public void methodOne(int i) { }  
    public void methodTwo(int i) { }  
    public static void methodThree(int i) { }  
    public static void methodFour(int i) { }  
}  
  
public class ClassB extends ClassA {  
    public void methodTwo(int i) { }  
    public void methodThree(int i) { }  
    public static void methodFour(int i) { }  
}
```

Quale metodo fa overriding? Qualcuno fa overloading?

<https://docs.oracle.com/javase/tutorial/java/landl/QandE/inherit-questions.html>

Invocazione dinamica dei metodi e polimorfismo

Dato la possibilità di fare l'overriding dei metodi, e dato che un oggetto può essere assegnato ad un reference di una qualunque delle sue superclassi, la decisione di quale metodo invocare deve avvenire necessariamente dinamicamente a runtime.

Questa capacità delle sottoclassi di compiere azioni diverse a partire da un comportamento comune della superclasse è definito polimorfismo.

Classi astratte

Una classe può essere definita come astratta, e contenere dei metodi anch'essi astratti.

```
public abstract class Figura {  
    public abstract double area();  
}  
  
public class Rettangolo extends Figura {  
    public double area() { return d1*d2;}  
}
```

```
Figura f = disegno.getFigura(0);  
System.out.println(f.area());
```

Esercizio

Usando come base la classe:

```
public abstract class Figura {  
    public abstract double area();  
  
    public String toString() {  
        return "L'area della figura è "+area();  
    }  
}
```

definire le classi Quadrato, Rettangolo, Triangolo e Cerchio.

Il metodo `equals`

`equals` è un altro metodo della classe `Object`. Viene usato per confrontare gli oggetti per uguaglianza del loro stato interno. L'implementazione di `Object` fa semplicemente il confronto dei reference, come l'operatore `==`, quindi ciascuna classe deve farne l'overriding se intende usarlo per il proprio confronto.

```
public boolean equals(Object o) {  
    if (o == null) return false;  
    if (o instanceof Figura) {  
        return this.area() == ((Figura)o).area();  
    }  
    return false;  
}  
  
if (f1.equals(f2)) { /* .. */ }
```

Il metodo `hashCode`

Restituisce un code di hash. Confrontando due oggetti, se i loro codici hash sono differenti, allora i due oggetti sono diversi (nel senso che `equals` restituirà `false`).

```
public int hashCode() {  
    return Double.valueOf(this.area()).hashCode();  
}
```

Se `a.hashCode() != b.hashCode()`, allora `a.equals(b) == false`.

L'annotazione `@Override`

L'annotazione `@Override` segnala al compilatore che l'intenzione è quella di fare l'override di un metodo (non quello di fare un overload, o definire altro metodo)

```
@Override  
public int hashCode() {  
    return Double.valueOf(this.area).hashCode();  
}
```

Esercizio

- Aggiungere il metodo `equals` alle classi "figura" costruite precedentemente.
- Aggiungere il metodo `equals` alla classe `Author` e usarlo per la ricerca per autore.

Ancora **final**

Per evitare che un metodo possa essere overridden:

```
public class A {  
    public final void method() { /* ... */ }  
}
```

Per evitare che una classe possa essere ereditata:

```
public final class A {  
    // ...  
}
```

Wrapper dei tipi primitivi

A volte occorre usare al posto dei tipi primitivi una loro rappresentazione ad oggetti. La Java API fornisce un *type wrapper* per ciascuno dei tipi primitivi.

Per i tipi numerici, la classe (astratta) base è `Number`, che fornisce i metodi di conversione:

```
byte byteValue()  
double doubleValue()  
float floatValue()  
int intValue()  
long longValue()  
short shortValue()
```

`Number` ha implementazioni concrete nelle classi `Double`, `Float`, `Byte`, `Short`, `Integer` e `Long`.

`Character` è il type wrapper per `char`, mentre `Boolean` è quello per boolean.

Uso dei type wrapper

```
// Costruire un oggetto type wrapper  
  
Integer oi1 = Integer.valueOf(10);  
Integer oi2 = Integer.valueOf("10");  
  
// Usare i type wrapper per convertire Stringhe  
// in tipi primitivi  
  
int i = Integer.parseInt("10");  
  
// Convertire a String i tipi primitivi  
Integer.toString(10);
```

Inoltre sono necessari per essere usati con la Collection API.

Autoboxing e Auto-unboxing

L'autoboxing è il processo in cui un tipo primitivo viene automaticamente convertito al suo corrispondente type wrapper:

```
Integer oi = 100;
```

L'auto-unboxing è il processo inverso, da type wrapper a tipo primitivo:

```
int i = oi;
```

In generale tali processi avvengono automaticamente quando occorre un tipo o l'altro: nelle assegnazioni, nelle espressioni e nei passaggi di parametri ai metodi.

Attenzione alle performance!

```
Integer oi = 100;
```

```
oi++;
```

```
Double a = 10.0, Double b = 15.5;
```

```
Double c = a + b;
```

Enumeration

Le enumeration sono un modo per definire un nuovo tipo contenente una serie di costanti, con lo scopo di definire i soli valori validi che possono essere assegnati a tale nuovo tipo.

```
enum Risposta {  
    SI, NO, FORSE, NESSUNA_RISPOSTA  
}
```

```
Risposta r = Risposta.SI;
```

```
if (r == Risposta.SI) { /* ... */ }
```

Switch con enumeration

```
switch(r) {  
    case SI:  
        /* ... */  
        break;  
    case NO:  
        /* ... */  
        break;  
    case FORSE:  
        /* ... */  
        break;  
    case NESSUNA_RISPOSTA:  
        /* ... */  
        break;  
    default:  
        /* ... */  
}
```

Metodi di enum

```
Risposta[] risposte = Risposta.values()

Risposta r = Risposta.valueOf("SI");

int n = r.ordinal();

if (r1.compareTo(r2) > 0) { /* ... */ }
```

Le enum sono classi

```
enum Risposta {  
    SI(10), NO(5), FORSE(7), NESSUNA_RISPOSTA(0)  
  
    private int peso;  
  
    Risposta(int peso) {  
        this.peso = peso;  
    }  
  
    int getPeso() {  
        return this.peso;  
    }  
}  
  
int p = Risposta.SI.getPeso(); // Ritorna 10
```

Esercizio Categorie dei libri

com.corsojava.thelibrary.core.model

C Book

- String title
- float price
- Author[] authors
- Publisher publisher
- BookCategory[] categories

constructors

- Book(String title, Author[] authors)
- Book(String title, Author[] authors, float price)
- Book(String title, Author[] authors, Publisher publisher, float price)

setters/getters

- String getTitle()
- void setTitle(String title)
- float getPrice()
- void setPrice(float price)
- Author[] getAuthors()
- void setAuthors(Author[] author)
- Publisher getPublisher()
- void setPublisher()
- BookCategory getCategories()
- void setCategories(BookCategory[] categories)

other methods

- void addAuthor(Author author)
- void addCategory(BookCategory category)

C «enumeration»
BookCategory

OTHER
ARTS_AND_PHOTOGRAPY
COMPUTERS_AND_TECHNOLOGY
HISTORY
LITERATURE_AND_FICTION

Interfacce

Interfacce

Le interfacce sono un modo di definire **cosa** una classe deve fare, lasciando l'implementazione di **come** farlo alla classe stessa.

Un'interfaccia definisce i metodi che una classe deve obbligatoriamente implementare, ma, in generale, non ne fornisce un'implementazione.

Può essere pensata come la definizione di una classe totalmente astratta.

Una classe può implementare più di un'interfaccia, cioè definire i metodi dettate da esse.

Le interfacce permettono di scollegare dalla gerarchia di ereditarietà la definizione dei metodi che una classe deve implementare.

Interfacce - Definizione

```
public interface Esecutore {  
    void esegui();  
}
```

In un'interfaccia si definiscono principalmente i metodi, ma non se ne fornisce un'implementazione. Possono anche essere definite delle variabili, che sono implicitamente `static` e `final`. Tali metodi e variabili sono implicitamente `public`.

Da JDK 8 è possibile definire in un'interfaccia anche un'implementazione di default per i metodi, e definire metodi statici. Da JDK 9 un'interfaccia può avere dei metodi privati.

Interfacce - implementazione

```
public class Impiegato extends Persona implements Esecutore {  
    ...  
  
    public void esegui() {  
        /* implementazione di esegui per impiegato */  
    }  
}
```

Interfacce - uso

```
Esecutore e = new Impiegato();  
e.esegui();
```

```
class Fabbrica {  
    public void produci(Esecutore[] esecutori) {  
        for(Esecutore e: esecutori) {  
            e.esegui();  
        }  
    }  
}
```

```
if (e instanceof Impiegato)  
    Impiegato i = (Impiegato)e;
```

Implementazione parziale

Una classe che dichiara di implementare un'interfaccia, ma non la implementi (o la implementi solo parzialmente), dovrà essere dichiarata **abstract**.

```
public abstract class Operaio extends Persona
                                implements Esecutore {

}
```

Interfacce multiple

Una classe può implementare più di un'interfaccia:

```
public class Dirigente extends Persona
    implements Esecutore, Comandante {
    /* implementazione sia di Esecutore, che di comandante */
}
```

Ereditarietà delle interfacce

Anche le interfacce possono essere estese:

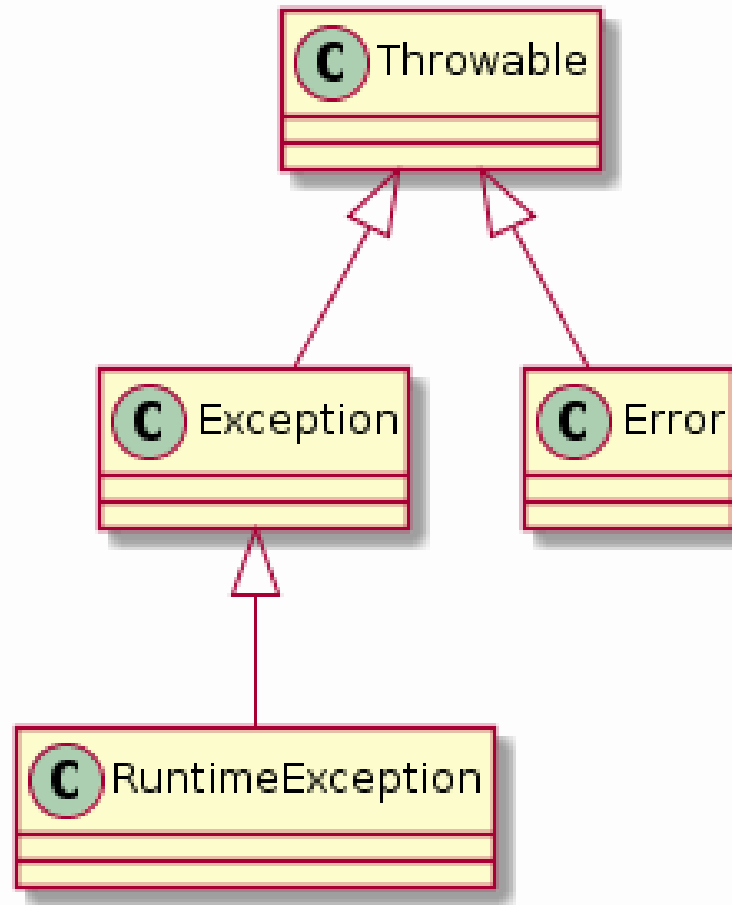
```
interface EsecutoreSpecializzato extends Esecutore {  
    void altroMetodo();  
}
```

Eccezioni

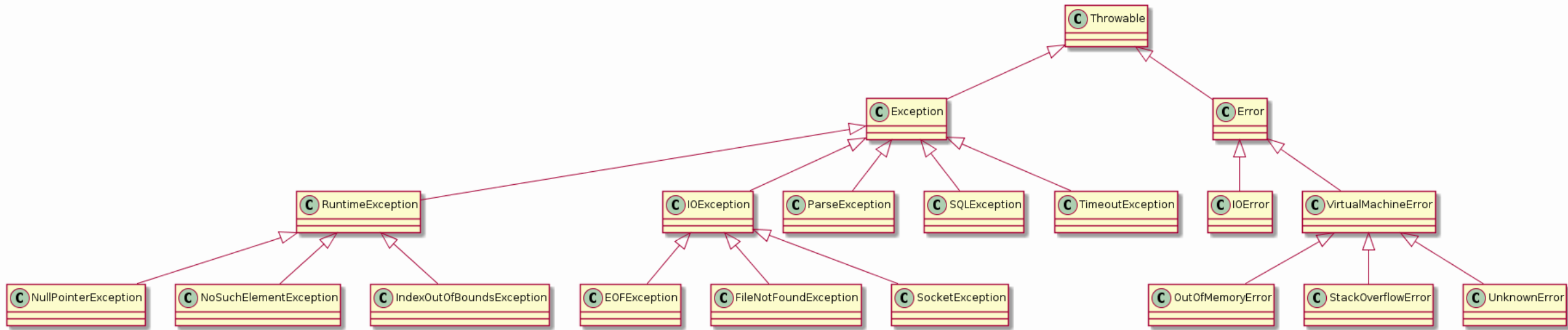
Eccezioni - sintassi

```
try {  
    // blocco in cui può verificarsi un'eccezione  
} catch (TipoEccezione1 e1) {  
    // gestione dell'eccezione 1  
} catch (TipoEccezione2 e2) {  
    // gestione dell'eccezione 2  
}  
// ...  
finally {  
    // blocco da eseguire dopo try/catch  
}
```


Gerarchia delle eccezioni



Gerarchia delle eccezioni



Sollevare eccezioni con **throw**

```
if (divisore == 0) {  
    throw new ArithmeticException("Non puoi dividere per 0");  
}
```

Non gestire le eccezioni con **throws**

```
public double leggi() throws IOException {  
    // codice che può generare una IOException  
}
```

Creare le proprie eccezioni

```
public class LibroStrappatoException extends Exception {  
}
```

Quindi nella mia applicazione potrà esserci:

```
throw new LibroStrappatoException();
```

Multi-catch

```
try {  
    // codice  
} catch ( ArithmeticException | ArrayIndexOutOfBoundsException e) {  
    // gestione di entrambe le eccezioni  
}
```

Chiusura di risorse

```
BufferedReader br = null;
try {
    String line;
    br = new BufferedReader(new FileReader("C:\\aFile.txt"));
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        if (br != null) br.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Try With Resources

```
try (BufferedReader br =  
    new BufferedReader(new FileReader("C:\\aFile.txt"))) {  
    String line;  
    while ((line = br.readLine()) != null) {  
        System.out.println(line);  
    }  
} catch (IOException e) {  
    e.printStackTrace();  
}
```


Reflection

Reflection

È la capacità di un software di analizzare sè stesso, in particolare per scoprire a runtime quali ne sono le caratteristiche utilizzabili.

Ad esempio, mediante la reflection, è possibile scoprire quali sono i costruttori, gli attributi e i metodi di una classe.

Esempio - Costruttori, Attributi e Metodi

```
Class c = (args.length > 0)?Class.forName(args[0]):String.class;
System.out.println(c.getName());
for (Constructor constructor : c.getConstructors()) {
    System.out.print("  ");
    System.out.println(constructor);
}
System.out.println("  -----");
for (Field field : c.getFields()) {
    System.out.print("  ");
    System.out.println(field);
}
System.out.println("  -----");
for (Method method : c.getMethods()) {
    System.out.print("  ");
    System.out.println(method);
}
```

Ottenere un oggetto Class

```
String s = "Hello World";  
Class c1 = s.getClass();  
Class c2 = String.class;  
Class c3 = Class.forName("java.lang.String");
```

Restituiscono tutti lo stesso oggetto (`c1 == c2 == c3`).

Invocare i metodi via reflection

```
Class c = Person.class;  
Constructor constructor =  
    c.getConstructor(String.class, String.class, int.class);  
Person person = constructor.newInstance("Lucio", "Benfante", 50);  
Method m = c.getMethod("describePerson");  
System.out.println(m.invoke(person));
```

"Bypassare" i restrittori di accesso

```
Class c = Person.class;
Constructor constructor =
    c.getConstructor(String.class, String.class, int.class);
Person person = constructor.newInstance("Lucio", "Benfante", 50);
Method m = c.getMethod("describePerson");
System.out.println(m.invoke(person));

Field field = c.getDeclaredField("lastName");
field.setAccessible(true);
field.set(person, "Rossi");
System.out.println(person.describePerson());
```

Proxy dinamici

```
InvocationHandler handler = new MyInvocationHandler();  
MyInterface proxy = (MyInterface) Proxy.newProxyInstance(  
    MyInterface.class.getClassLoader(),  
    new Class[] { MyInterface.class },  
    handler);
```

```
public class MyInvocationHandler implements InvocationHandler{  
    public Object invoke(Object proxy, Method method, Object[] args)  
    throws Throwable {  
        //do something "dynamic"  
    }  
}
```

Deploying delle applicazioni

JAR

manifest.txt

```
Main-Class: com.esempio.MyExampleWithMain
```

```
jar cvfm MyApplication.jar manifest.txt -C bin/ .
```

```
java -cp MyApplication.jar com.esempio.MyExampleWithMain
```

```
java -jar MyApplication.jar
```

Applet

```
import java.applet.*;
import java.awt.*;

public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello World Applet!.", 40, 20);
    }
}
```

Applet nella pagina Web



Java Web Start

Esempio Jnlp
Your Java Company
A Java Web Start Example

Introduzione ai Design Patterns

Design Patterns

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

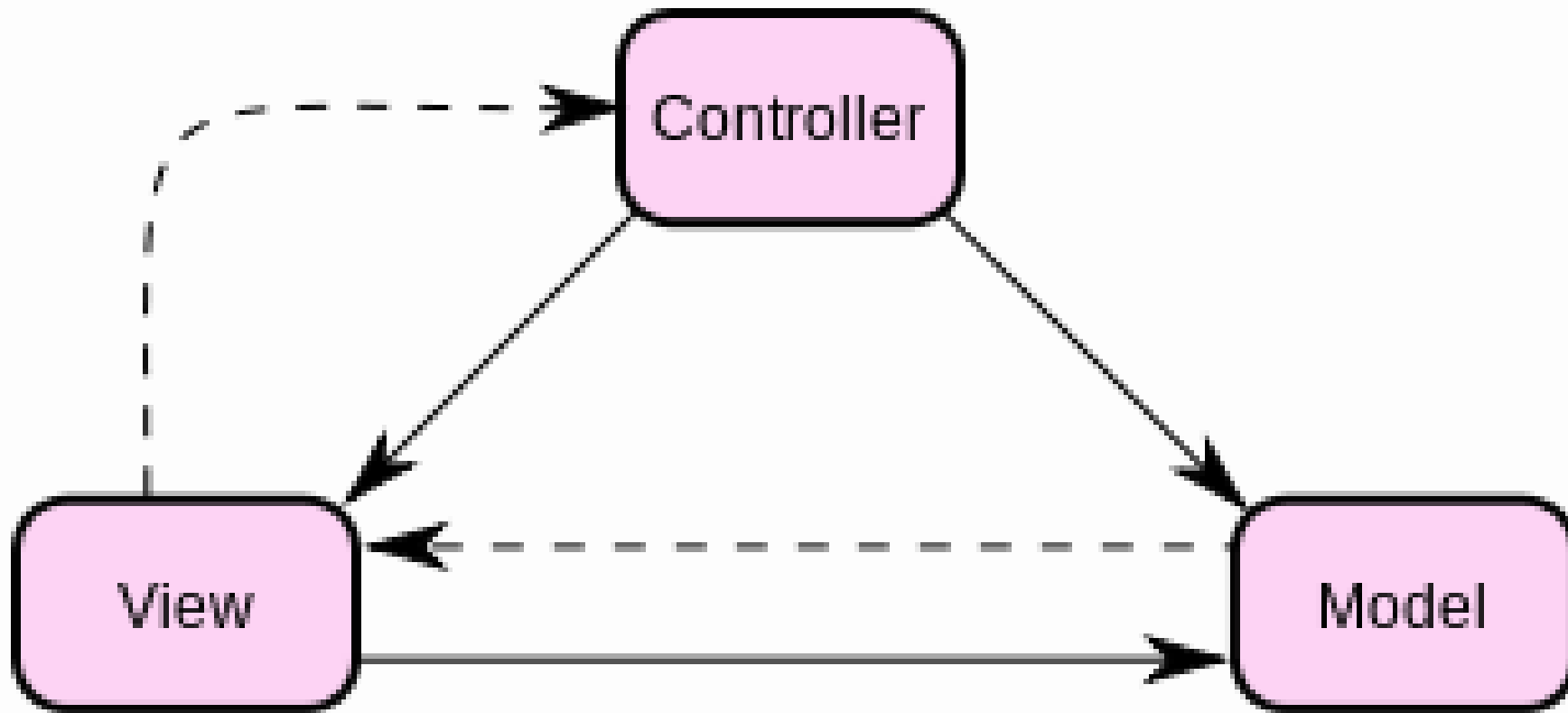
Christopher Alexander, architect

Caratteristiche

1. Nome del pattern
2. Problema
3. Soluzione
4. Conseguenze

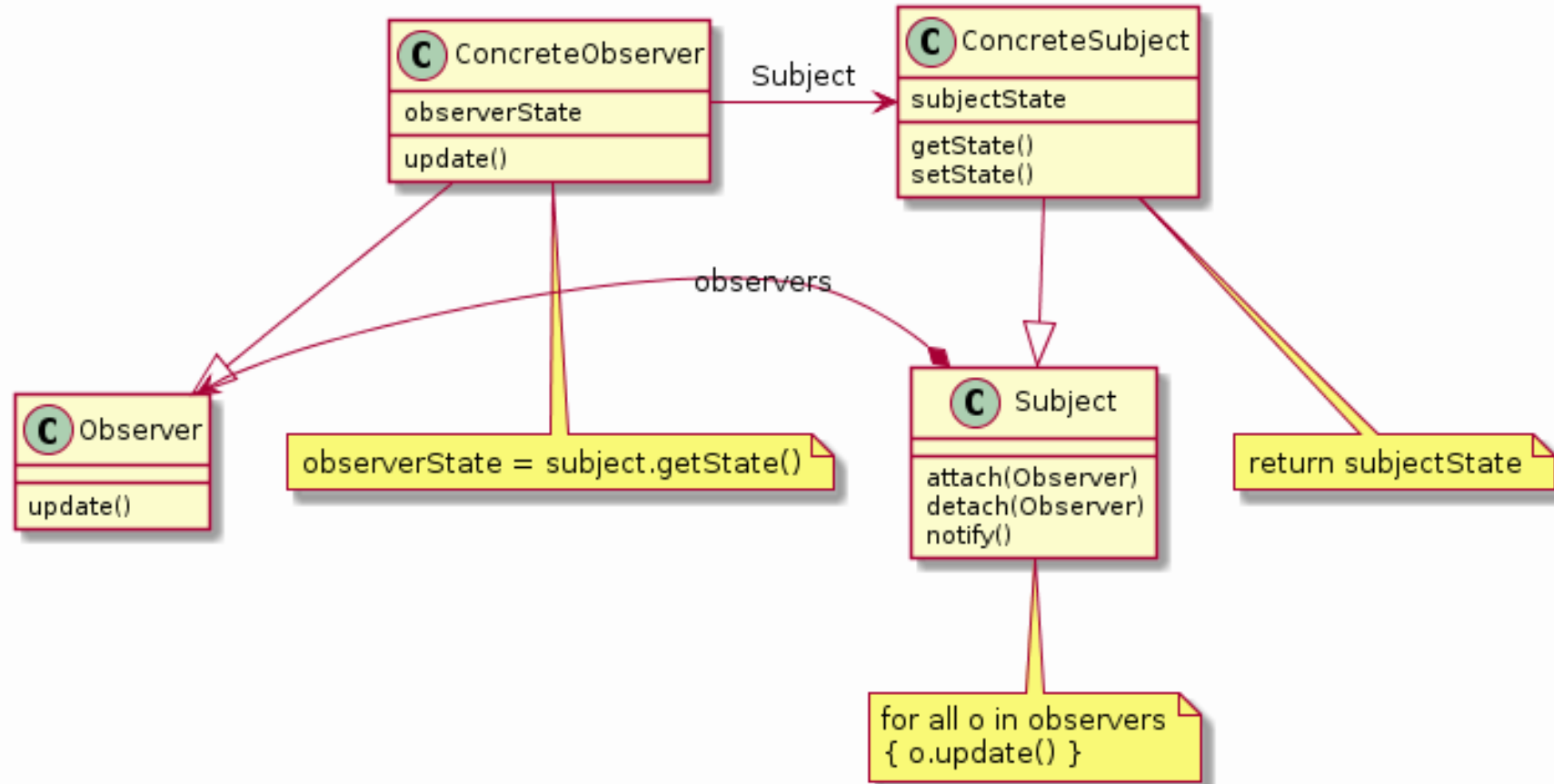
Pattern strutturali

Model View Controller (MVC)

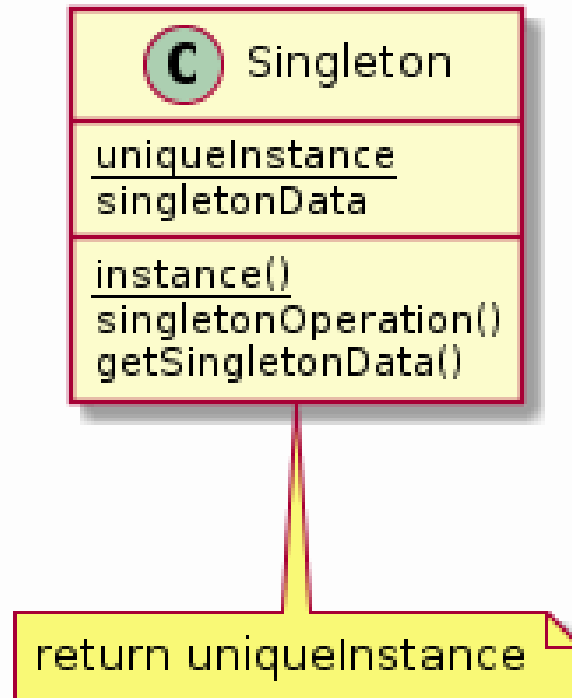


Di File:ModelViewControllerDiagram.svg: Traced by User:Stannered / *opera derivata Davjoh - File:ModelViewControllerDiagram.svg, Pubblico dominio, <https://commons.wikimedia.org/w/index.php?curid=10194593>

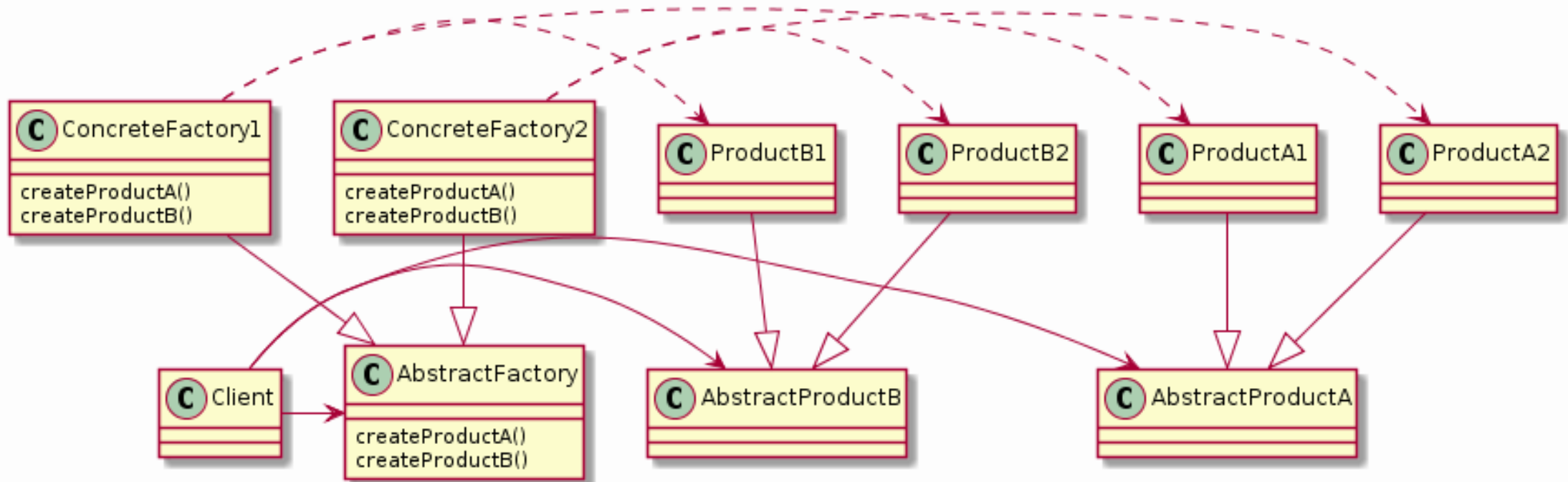
Observer



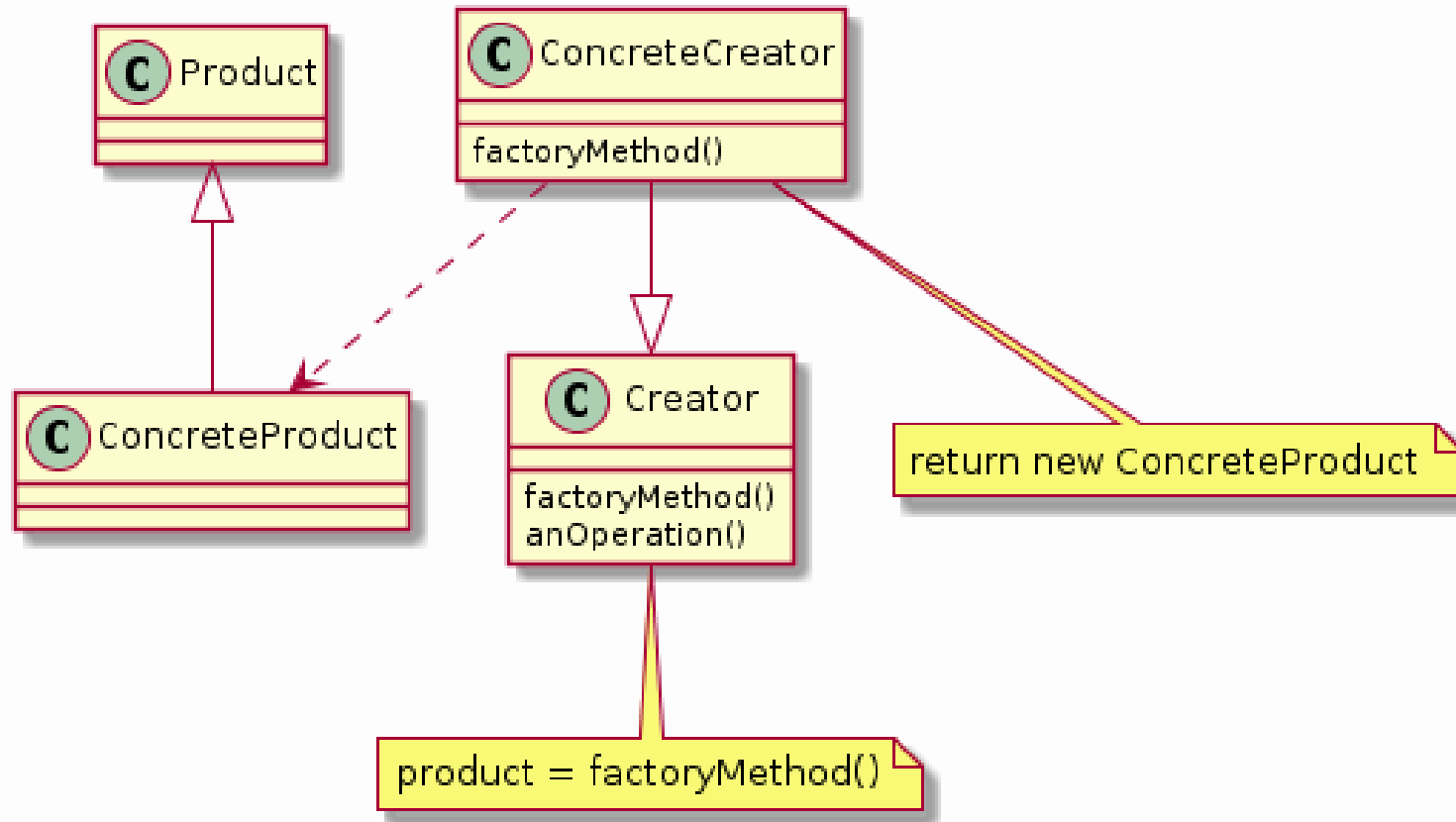
Singleton



Abstract Factory



Factory method



JavaBeans

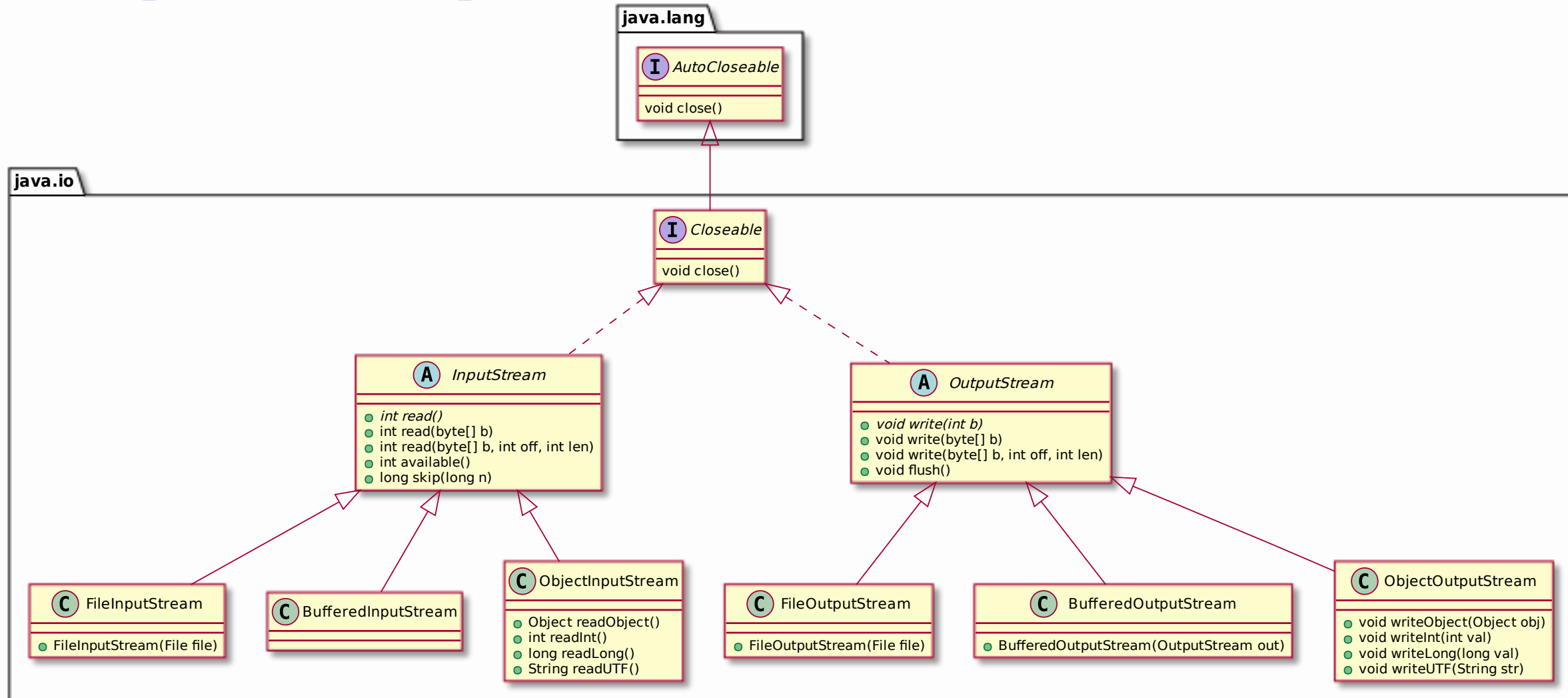
Pattern? Più che altro una specifica per componenti lato client.

Sono sostanzialmente classi con getter e setter.

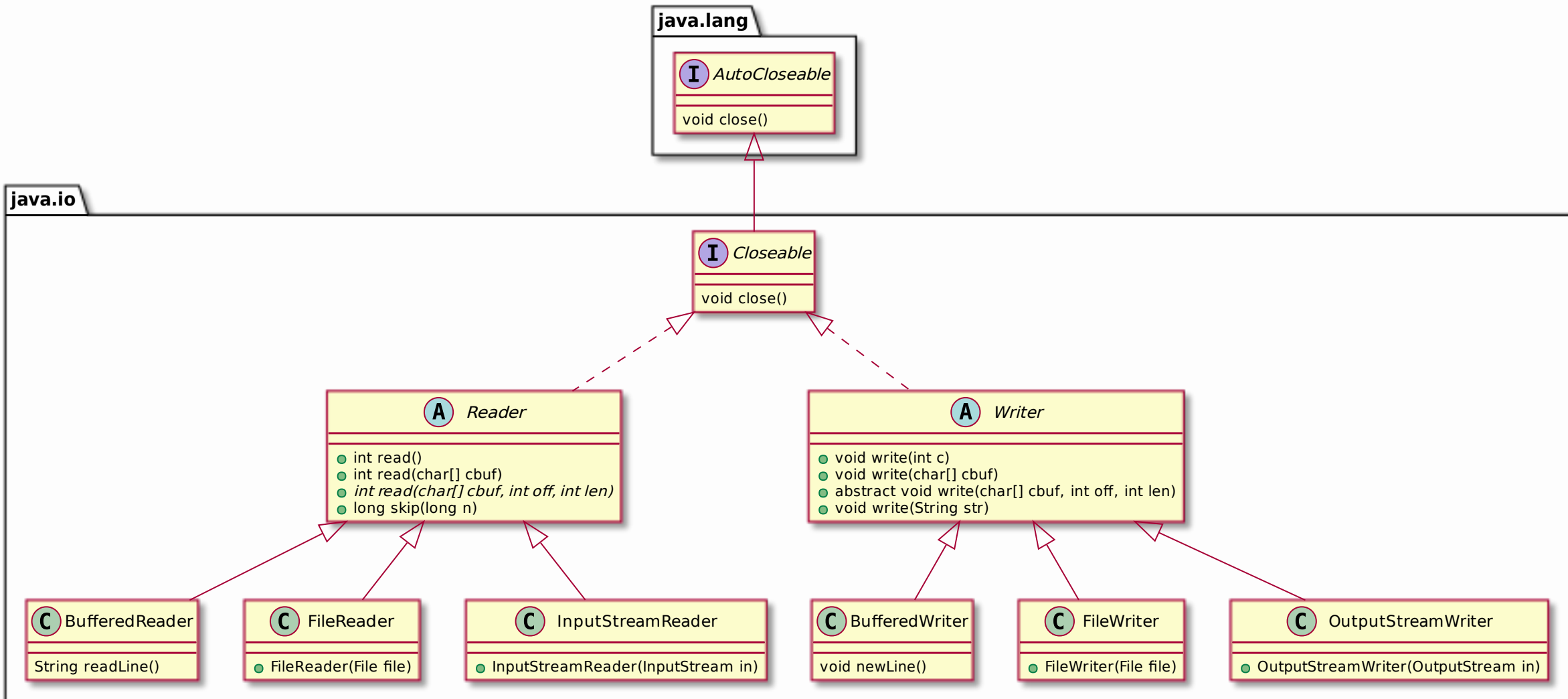
Da non confondere Con gli Enterprise Java Beans (EJB), che sono invece componenti lato server.

Input/Output in Java

Input/OutputStreams



Readers and Writers



Esercizio lettura da tastiera

- Nel main della classe Library, leggere da tastiera i dati di alcuni Libri. Nell'ordine leggere:
 - Titolo del libro (String)
 - prezzo del libro (BigDecimal)
 - Nome dell'autore (String)
 - Cognome dell'autore (String)
 - ripetere l'inserimento dell'autore finchè come nome non viene messo "FINE".
 - Nome dell'editore

Aggiungere il libro nell'archivio di libri di Library (usando addBook).

Ripetere l'inserimento di un libro, finchè come titolo del libro viene scritto "FINE".

La classe **File**

java.io

C File

- File(File parent, String child)
- File(String pathname)
- File[] listFiles()
- boolean createNewFile()
- boolean mkdir()
- boolean mkdirs()
- boolean delete()
- void deleteOnExit()
- boolean renameTo(File dest)
- boolean exists()
- boolean isDirectory()
- boolean isFile()
- boolean isHidden()
- long lastModified()
- long length()
- boolean canRead()
- boolean canWrite()
- boolean canExecute()
- long getFreeSpace()
- long getTotalSpace()
- long getUsableSpace()
- File createTempFile(String prefix, String suffix)
- File createTempFile(String prefix, String suffix, File directory)

Esercizio - Archivio persistente

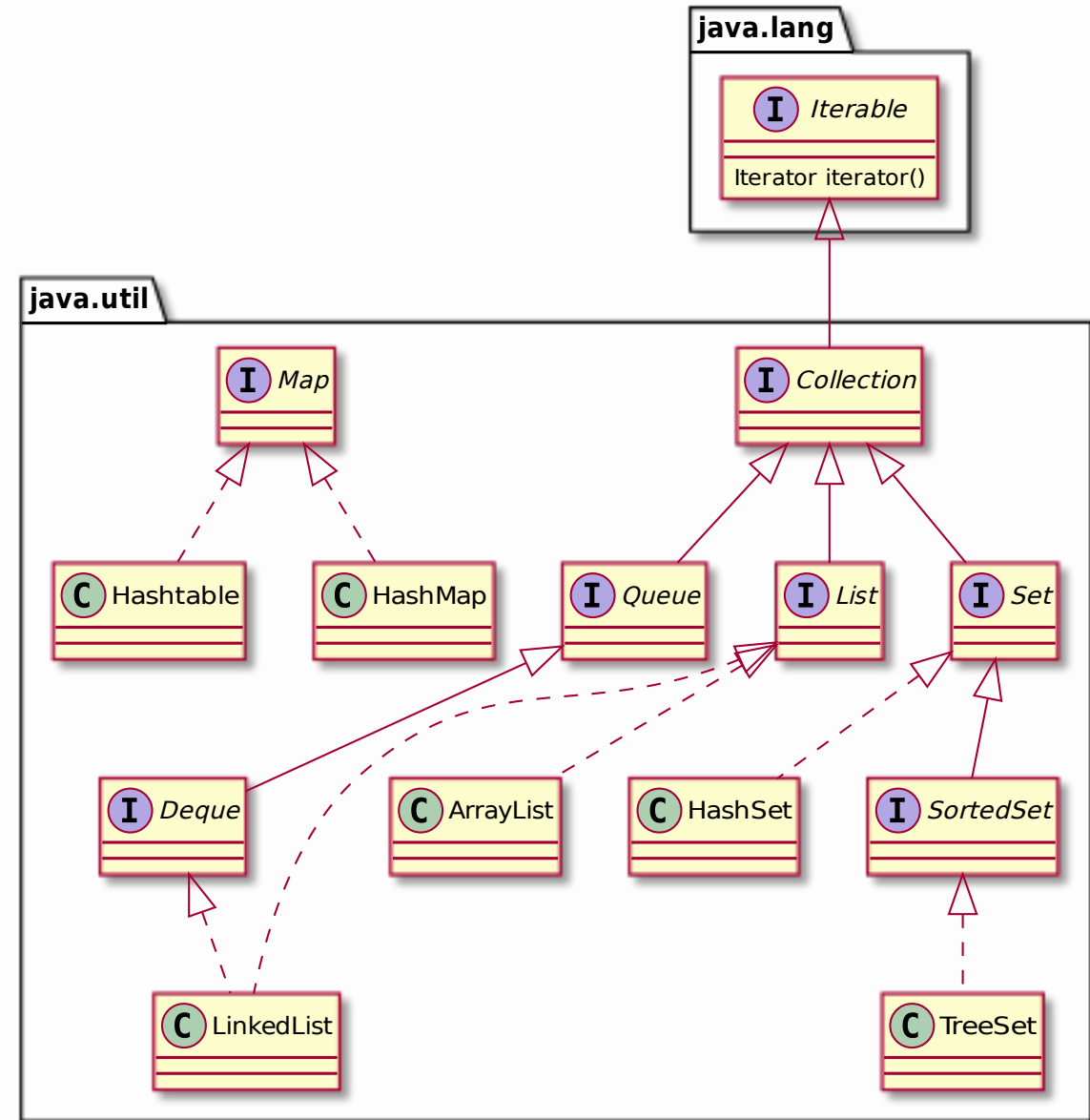
Aggiungere alla classe `library` due metodi:

- il primo, `public void storeArchive()`, salva l'array dei libri in un file `archive.dat` usando la serializzazione (`ObjectOutputStream.writeObject`).
- il secondo, `public void loadArchive()`, carica l'array dei libri dal file `archive.dat` usando la serializzazione (`ObjectInputStream.readObject`).

Invocare il metodo `loadArchive` all'inizio del metodo `main` (ovviamente, dopo aver creato l'oggetto `Library`), e il metodo `storeArchive` alla fine del metodo `main`.

Collection API

Collection API



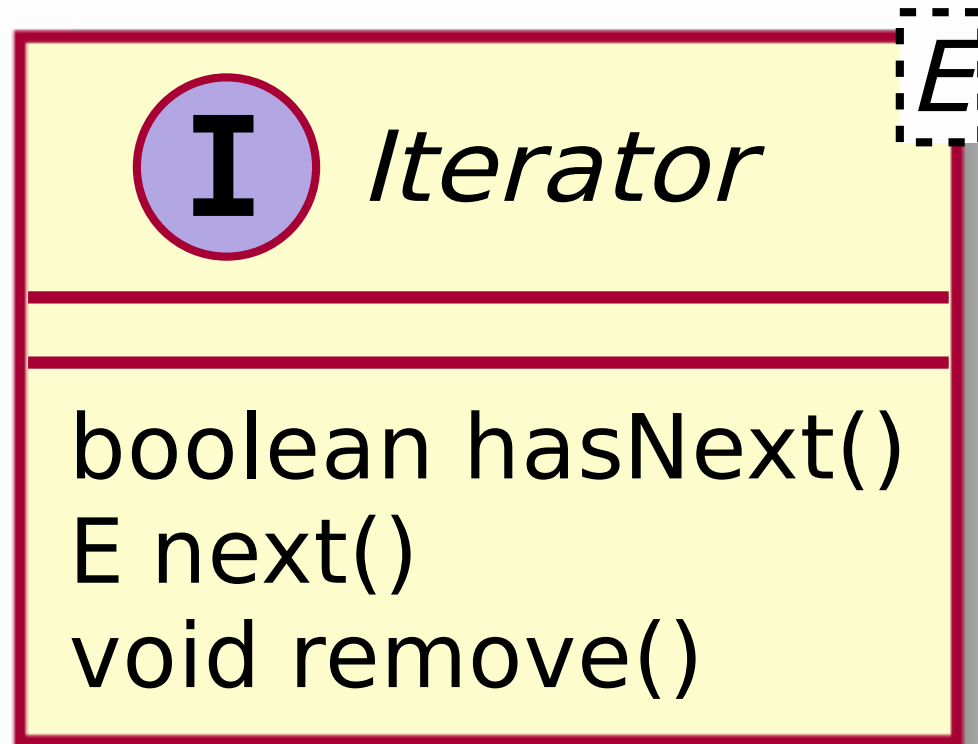
Interfaccia **Collection**

I *Collection*

E

```
boolean add(E e)
boolean addAll(Collection<? extends E> c)
void clear()
boolean contains(Object o)
boolean containsAll(Collection<?> c)
boolean isEmpty()
Iterator<E> iterator()
boolean remove(Object o)
boolean removeAll(Collection<?> c)
int size()
Object[] toArray()
<T> T[] toArray(T[] a)
```

Interfaccia **Iterator**



Interfaccia **List**

I *List*

E

```
void add(int index, E element)
boolean addAll(int index, Collection<? extends E> c)
E get(int index)
int indexOf(Object o)
boolean isEmpty()
int lastIndexOf(Object o)
ListIterator<E> listIterator()
ListIterator<E> listIterator(int index)
E remove(int index)
E set(int index, E element)
void sort(Comparator<? super E> c)
List<E> subList(int fromIndex, int toIndex)
```

Esercizio - Liste

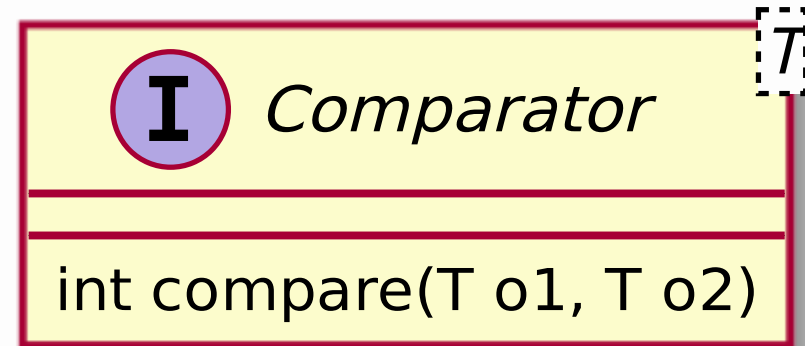
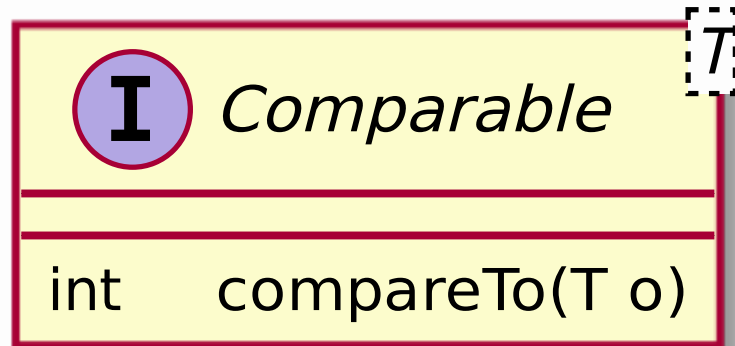
- Nella classe `Book` sostituire l'array dell'elenco degli autori, con un'oggetto di tipo `List`. Si aggiornino di conseguenza i costruttori, e set/get, il metodo di aggiunta di un autore, e, se necessario, tutto il codice che utilizza tale classe.
- Nella classe `Library` sostituire l'array dell'archivio dei libri, con un'oggetto di tipo `Collection` (in creazione dell'oggetto si usi, per il momento, una delle classi che implementano `List`). Si aggiorni il resto del codice in modo che continui a funzionare anche dopo tale modifica.

Rimuovere elementi da una lista

```
for(String animal: animals){  
    if(animal.equals("cat")){  
        // Solleva java.util.ConcurrentModificationException  
        animals.remove(animal);  
    }  
}
```

```
for(Iterator itr = animals.iterator(); itr.hasNext();){  
    String animal = itr.next();  
    if(animal.equals("cat")){  
        // animals.remove(animal); // No!  
        itr.remove();  
    }  
}
```

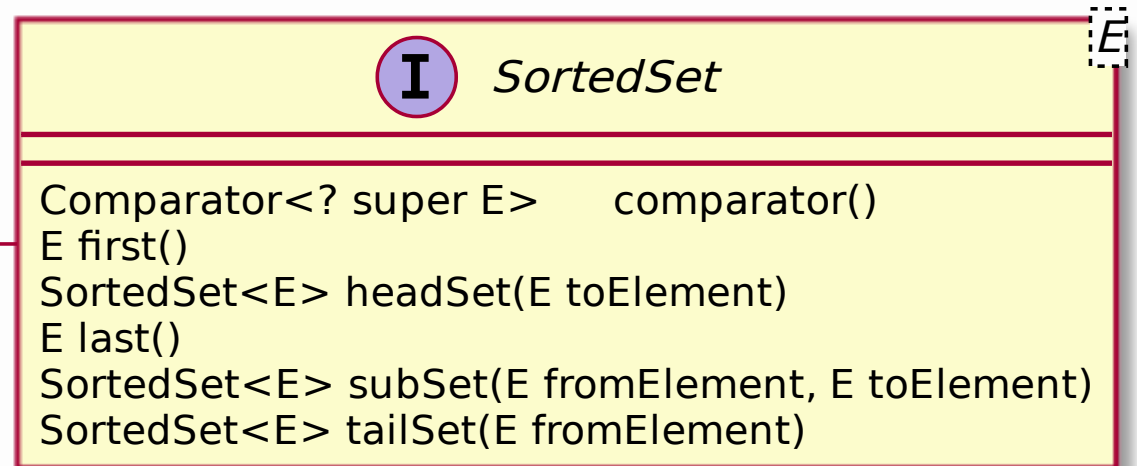
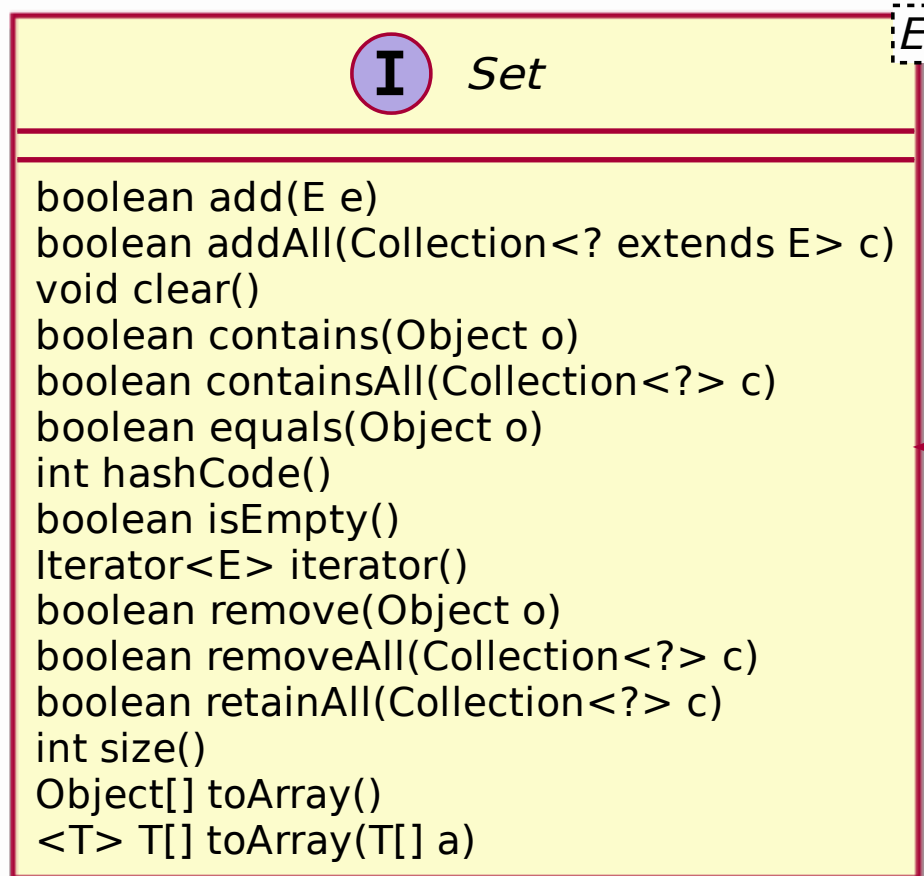
Interfacce Comparable e Comparator



Esercizio - Comparazione di oggetti

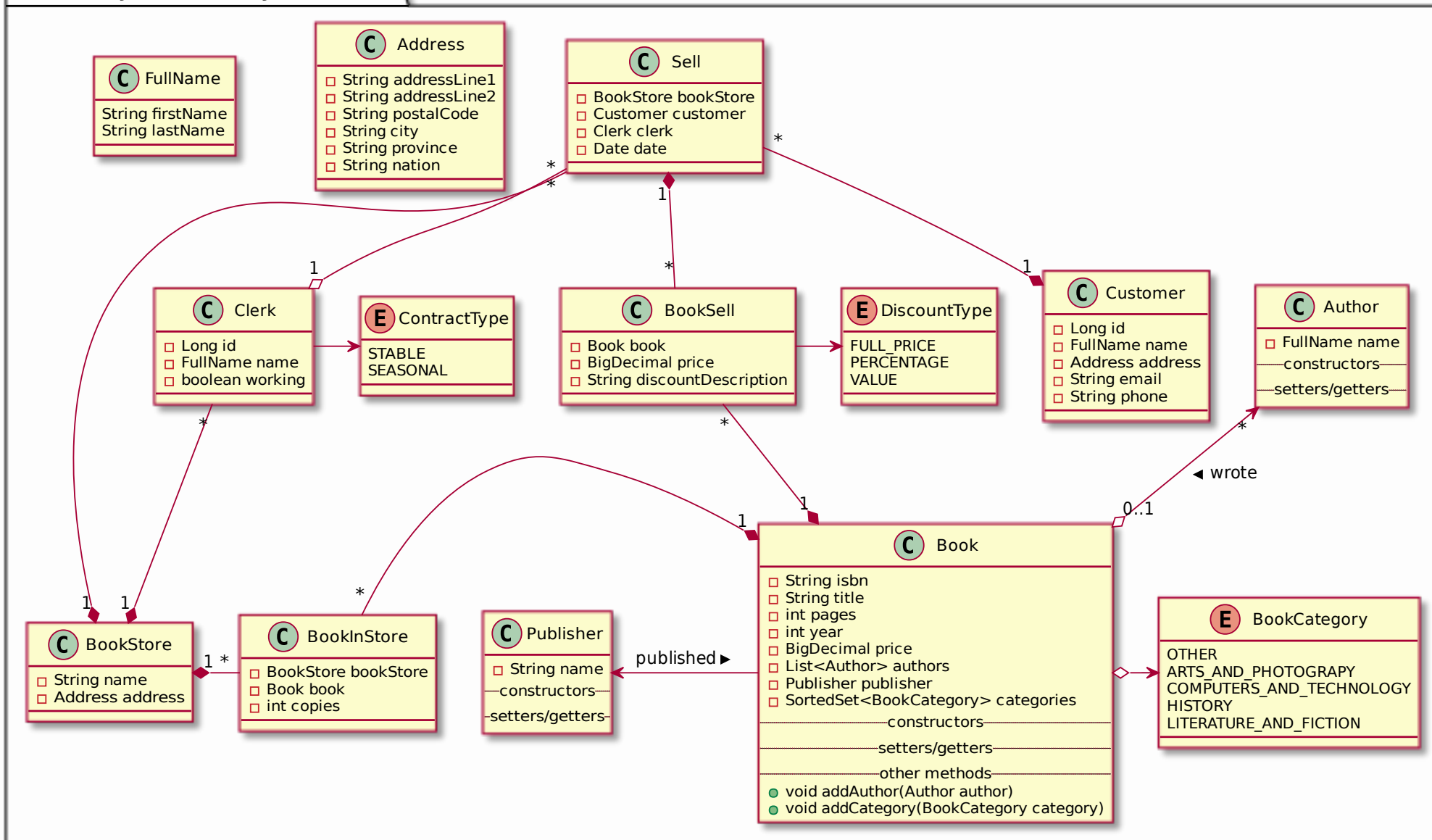
- Aggiungere e implementare l'interfaccia `Comparable` per la classe `Author` in maniera coerente al suo metodo `equals`, e in modo che gli autori siano ordinati in base al loro cognome e nome.
- Aggiungere alla classe `Book` l'attributo `isbn`, che conterrà il codice ISBN del libro, che lo identifica univocamente. Aggiungere i relativi metodi set. Implementare `equals` e `hashCode` usando il codice ISBN come attributo per determinare l'uguaglianza. Aggiungere e implementare l'interfaccia `Comparable` per la classe `Book` in maniera coerente al suo metodo `equals`, e in modo che ordini i libri in base al loro codice ISBN.
- Creare una classe `BookTitleComparator`, che implementa l'interfaccia `Comparator` in modo che i libri risultino ordinati per titolo.

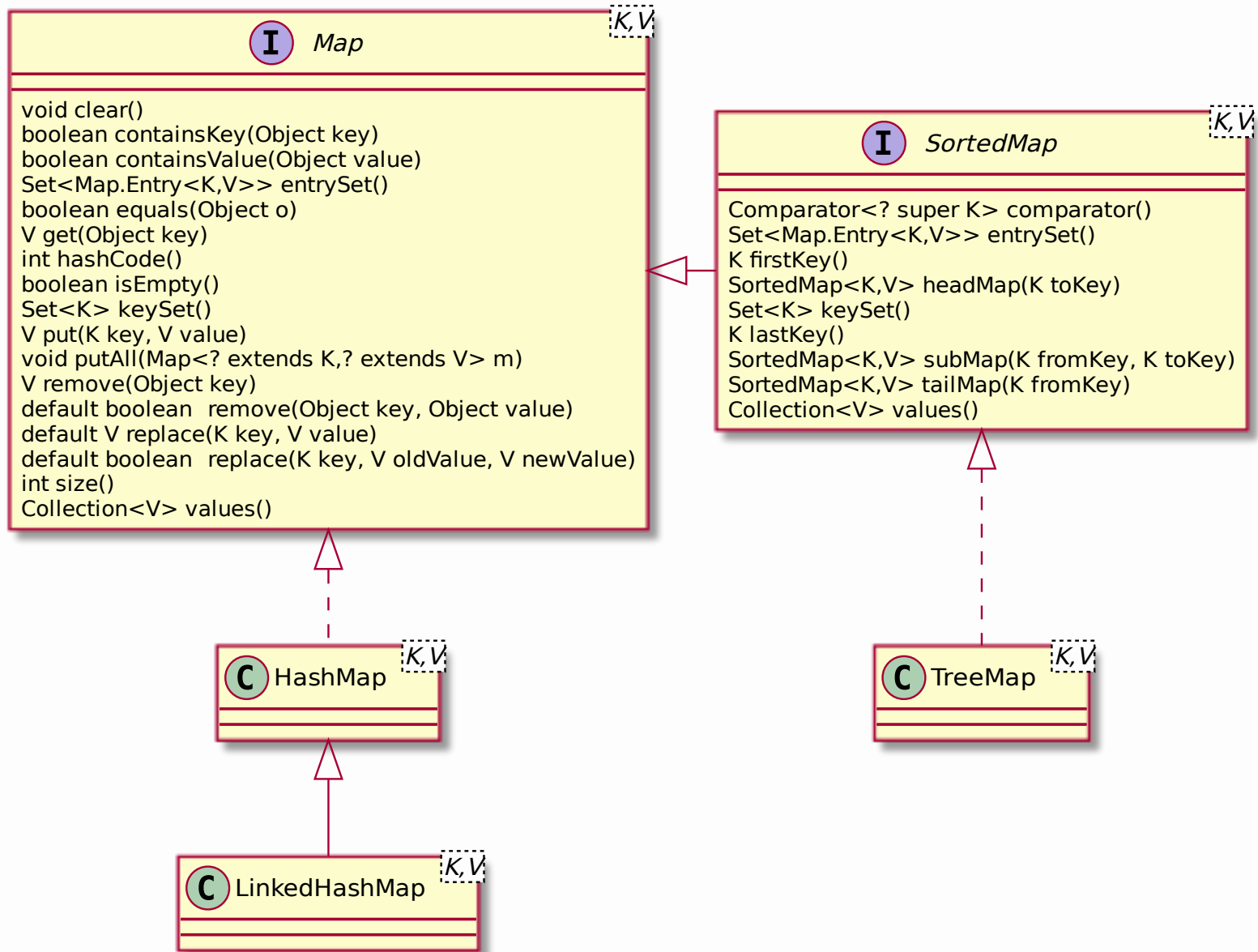
L'interfaccia Set



Esercizi

- Sostituire il contenitore delle categorie di un libro con un oggetto di tipo `SortedSet`, in modo che non ce ne siano di duplicate in un libro, e siano sempre ordinate in base alla loro definizione di enumeration.
- Sostituire il contenitore dell'archivio dei libri in `Library`, in modo che non ci siano libri duplicati.





Esercizi

- Data una collezione di `Person`, si costruisca una `Map` che permetta di trovare le persone in base al loro nome completo. (nome+" "+cognome).
- Data una collezione di `Person`, si costruisca una `Map` che permetta di trovare tutte le persone di una data età.

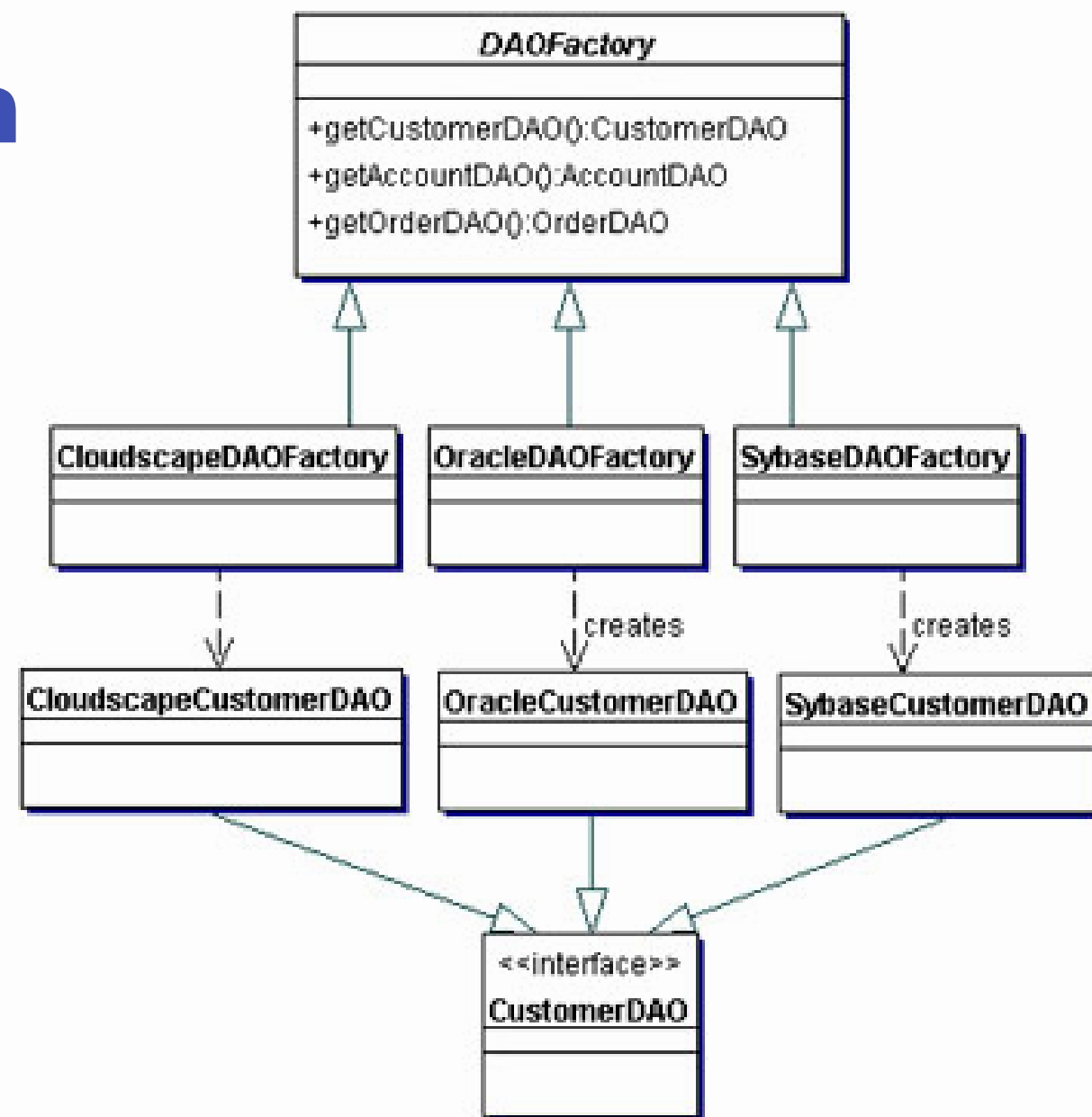
TheLibrary - Esercizi

- Nella classe `Library`, reimplementare i metodi `searchBooksByTitle` e `searchBooksByAuthor` in modo che utilizzino delle `Map` per eseguire la ricerca.

N.B.: significa che le mappe utilizzate per la ricerca devono essere mantenute aggiornate ogni volta che viene modificato l'archivio dei libri.

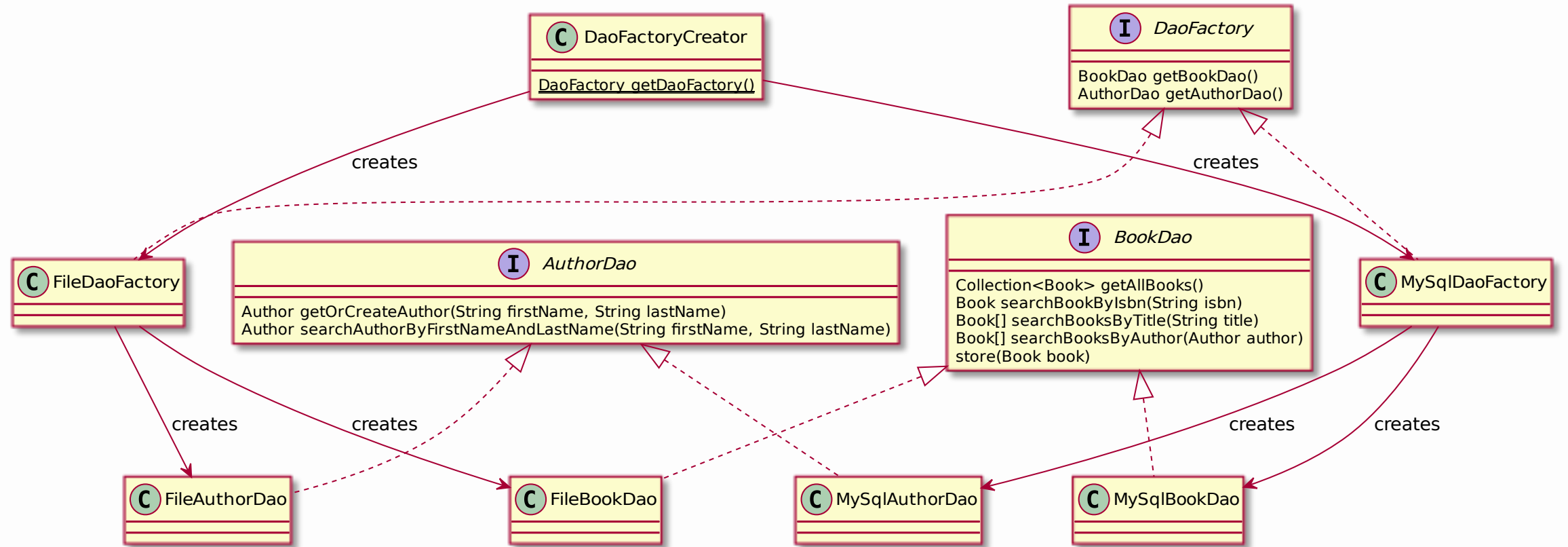
- Approfittare di quanto implementato nell'esercizio precedente per evitare di creare nuovi oggetti di tipo `Author`, nel caso si aggiunga un libro di un autore di cui esistano già dei libri nell'archivio.
- Aggiungere inoltre il metodo `Book searchBookByIsbn(String isbn)`, sempre implementato mediante una mappa.

Il pattern DAO(Da



TheLibrary DAOs

com.corsojava.thelibrary.core.dao



Date

- `java.util.Date`
- `java.util.Calendar`

`DateFormat` funziona solo con `Date`

Joda-time

Finalmente in Java 8

Nel package `java.time`:

- LocalDate
- LocalTime
- LocalDateTime
- Instant,
- Duration
- Period

Esempio

```
LocalDate date = LocalDate.of(2018, 1, 31);  
int year = date.getYear();  
Month month = date.getMonth();  
int day = date.getDayOfMonth();  
DayOfWeek dow = date.getDayOfWeek();  
int len = date.lengthOfMonth();  
boolean leap = date.isLeapYear();
```

```
LocalDate today = LocalDate.now();
```

Durate

```
Period tenDays = Period.between(  
    LocalDate.of(2017, 9, 11),  
    LocalDate.of(2017, 9, 21));
```

```
Duration threeMinutes = Duration.ofMinutes(3);  
Duration threeMinutes = Duration.of(3, ChronoUnit.MINUTES);  
Period tenDays = Period.ofDays(10);  
Period threeWeeks = Period.ofWeeks(3);  
Period twoYearsSixMonthsOneDay = Period.of(2, 6, 1);
```

Manipolazione delle date

```
LocalDate date1 = LocalDate.of(2017, 9, 21);  
LocalDate date2 = date1.plusWeeks(1);  
LocalDate date3 = date2.minusYears(6);  
LocalDate date4 = date3.plus(6, ChronoUnit.MONTHS);
```

```
LocalDate date1 = LocalDate.of(2014, 3, 18);  
LocalDate date2 = date1.with(nextOrSame(DayOfWeek.SUNDAY));  
LocalDate date3 = date2.with(lastDayOfMonth());
```