

Fondamenti di Informatica T2

Lab08 – MyMedia

Corso di Laurea in Ingegneria Informatica

Anno accademico 2021/2022

Prof. ROBERTA CALEGARI

Prof. AMBRA MOLESINI

Dipartimento di Informatica – Scienza e Ingegneria (DISI)



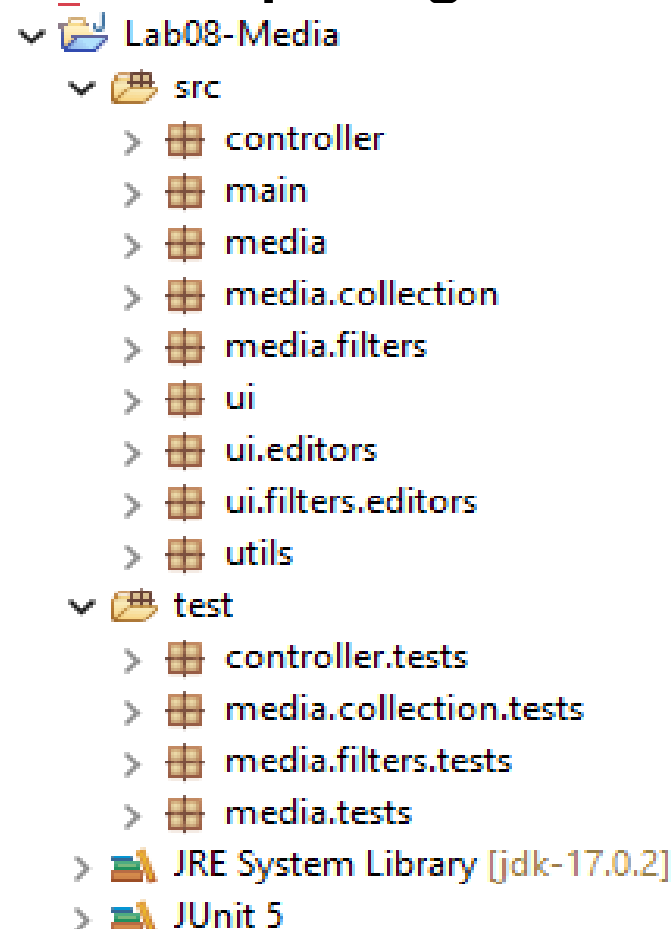
Applicazione MyMedia

- **OBIETTIVO: realizzare una applicazione per la gestione di una libreria di risorse multimediali**
 - canzoni, foto, film, ebook
- IN UNA PRIMA VERSIONE BASE, l'applicazione dovrà permettere di:
 - **Inserire** una nuova risorsa
 - **Cancellare** una risorsa
 - **Mostrare** tutte le risorse
- UNA SUCCESSIVA VERSIONE PIÙ EVOLUTA dovrà poi permettere anche di:
 - **Effettuare ricerche** sulle risorse in base a
 - Tipo della risorsa
 - Durata della risorsa (se disponibile)
 - Genere della risorsa (se disponibile)

Struttura generale

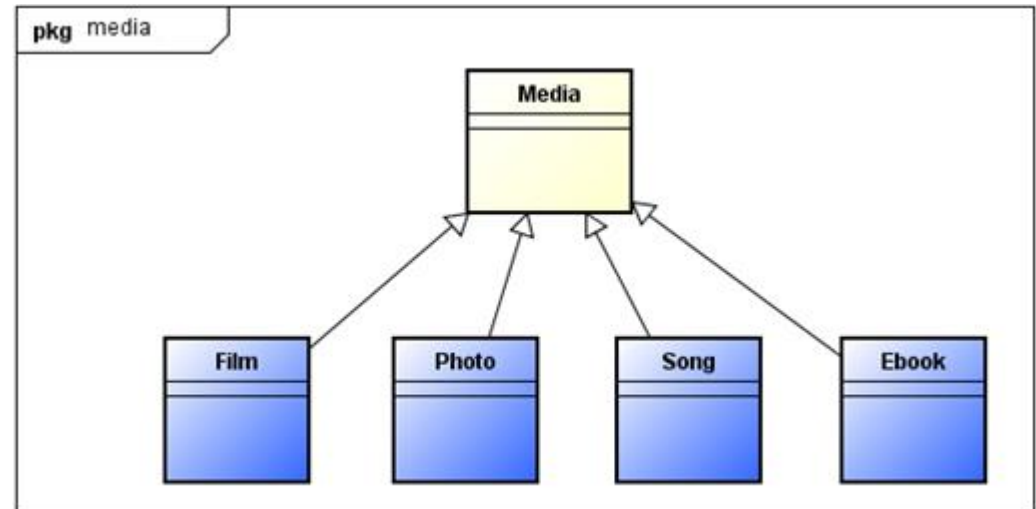
- L'applicazione sarà strutturata su ben *tredici* package

- 9 di logica applicativa + 4 di test
- in buona parte, già pronti 😊
- approccio: evitare mega-package, suddividendo invece per *sotto-obiettivi*
- *Don't Panic! Voi dovreste realizzare solo una minima parte del codice*
[quella tosta... 😊]



MyMedia: Model (1/4)

- Tutte le risorse sono (tipi diversi di) **Media**
 - in effetti, esistono informazioni comuni a tutti i tipi di risorsa
 - quindi, una **gerarchia di classi** modella bene il dominio
- **Media** è la classe base: fattorizza le proprietà comuni
- **Film, Photo, Song e Ebook** sono classi derivate che:
 - aggiungono proprietà o comportamento
 - ridefiniscono comportamento



MyMedia: Model (2/4)

- Tutti i **Media** sono caratterizzati da:

- Titolo e anno di produzione
- Metodo di rappresentazione in stringa
- Metodo di verifica di uguaglianza
- Metodo per recuperare il *tipo* di Media

**Proprietà
della classe
base Media**

- Una **Song** è caratterizzata da:

- Caratteristiche di base Media
- Cantante (o gruppo)
- Durata
- Genere

**Proprietà
specifiche
della classe
derivata
Song**

MyMedia: Model (3/4)

- Un **Film** è caratterizzato da:

- Caratteristiche base di Media
- Regista
- Elenco di attori
- Durata
- Genere

**Proprietà
specifiche
della classe
derivata
Film**

- Una **Photo** è caratterizzata da:

- Caratteristiche base di Media
- Elenco di autori

**Proprietà
specifiche
della classe
derivata
Photo**

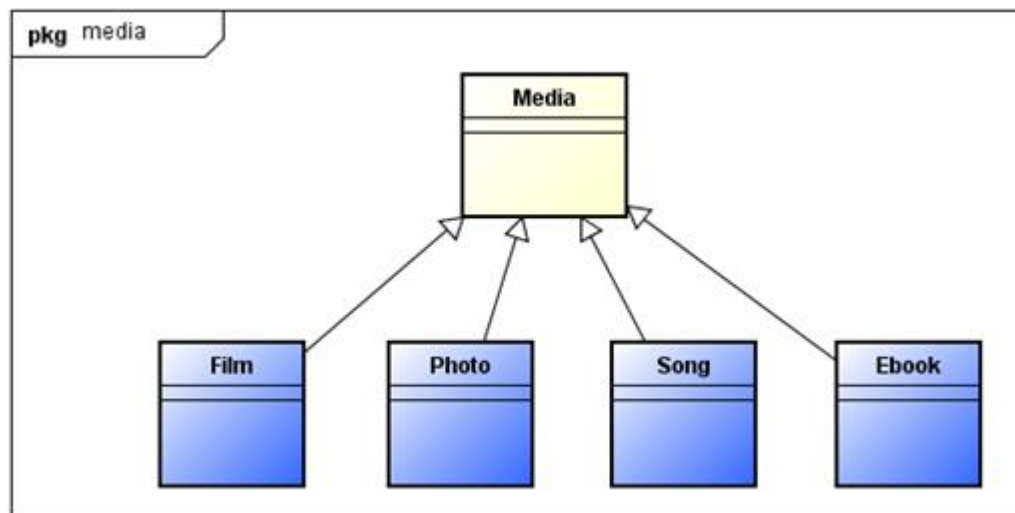
MyMedia: Model (4/4)

- Un **Ebook** è caratterizzato da:

- Caratteristiche base di Media
- Elenco di autori
- Genere

**Proprietà
specifiche
della classe
derivata
Ebook**

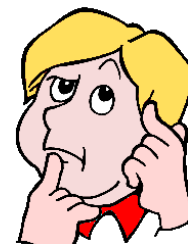
- Package:
tutto il *model* va
nel package
media



Media

- È la classe base della gerarchia
- Il **costruttore** prende in ingresso due parametri che rappresentano il *titolo* e *l'anno di produzione*
- I **metodi accessor** consentono di ottenere/impostare informazioni sul media, in particolare per:
 - ottenere il nome del tipo di media: **getType**
 - ottenere/impostare il titolo: **getTitle**, **setTitle**
 - ottenere/impostare l'anno: **getYear**, **setYear**
- Altri metodi consentono di:
 - ottenere la rappresentazione in stringa (**toString**)
 - stabilire se due **media** sono uguali (**equals**)

...ha senso crearne un'istanza?

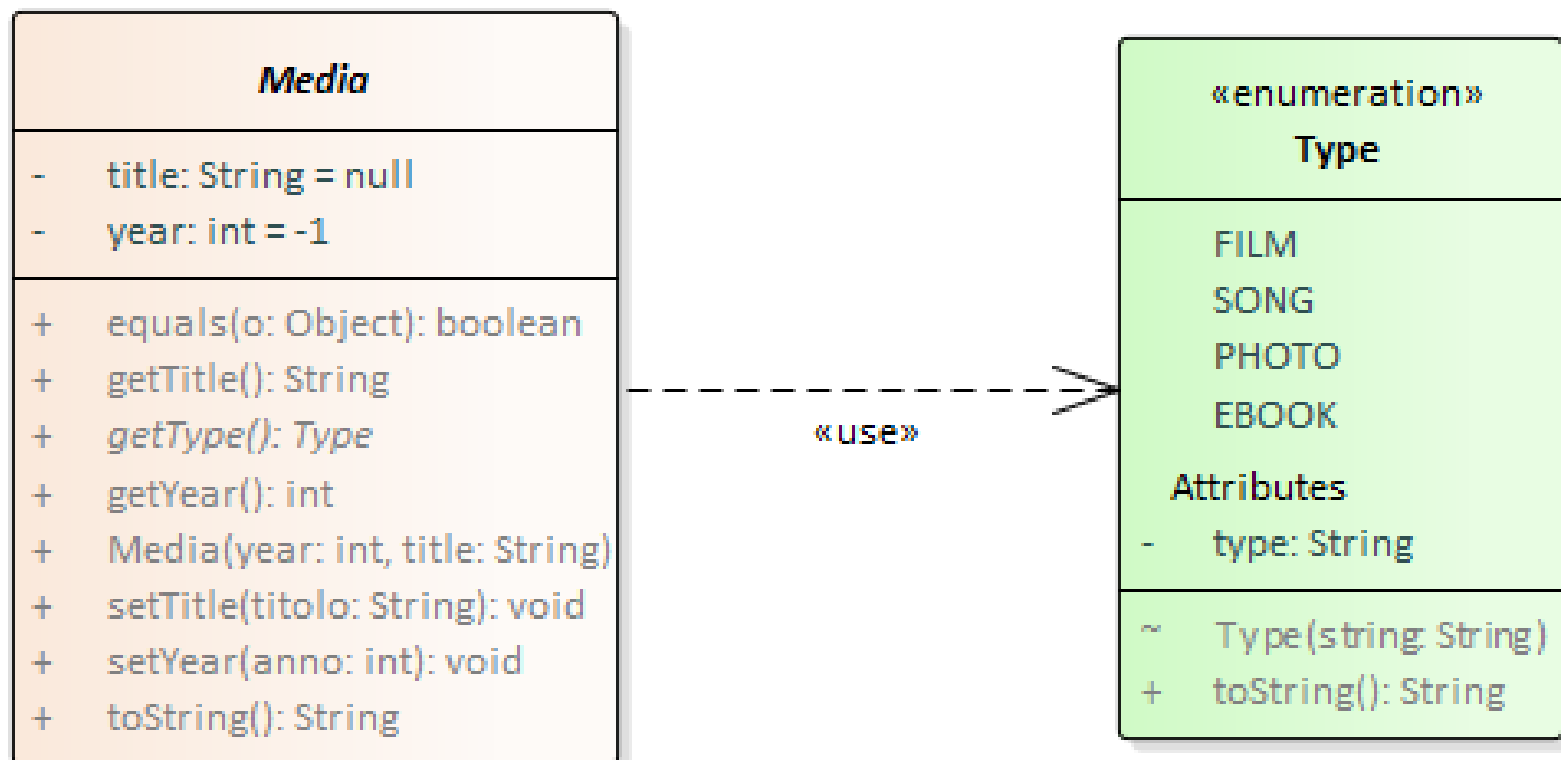




Il tipo del Media

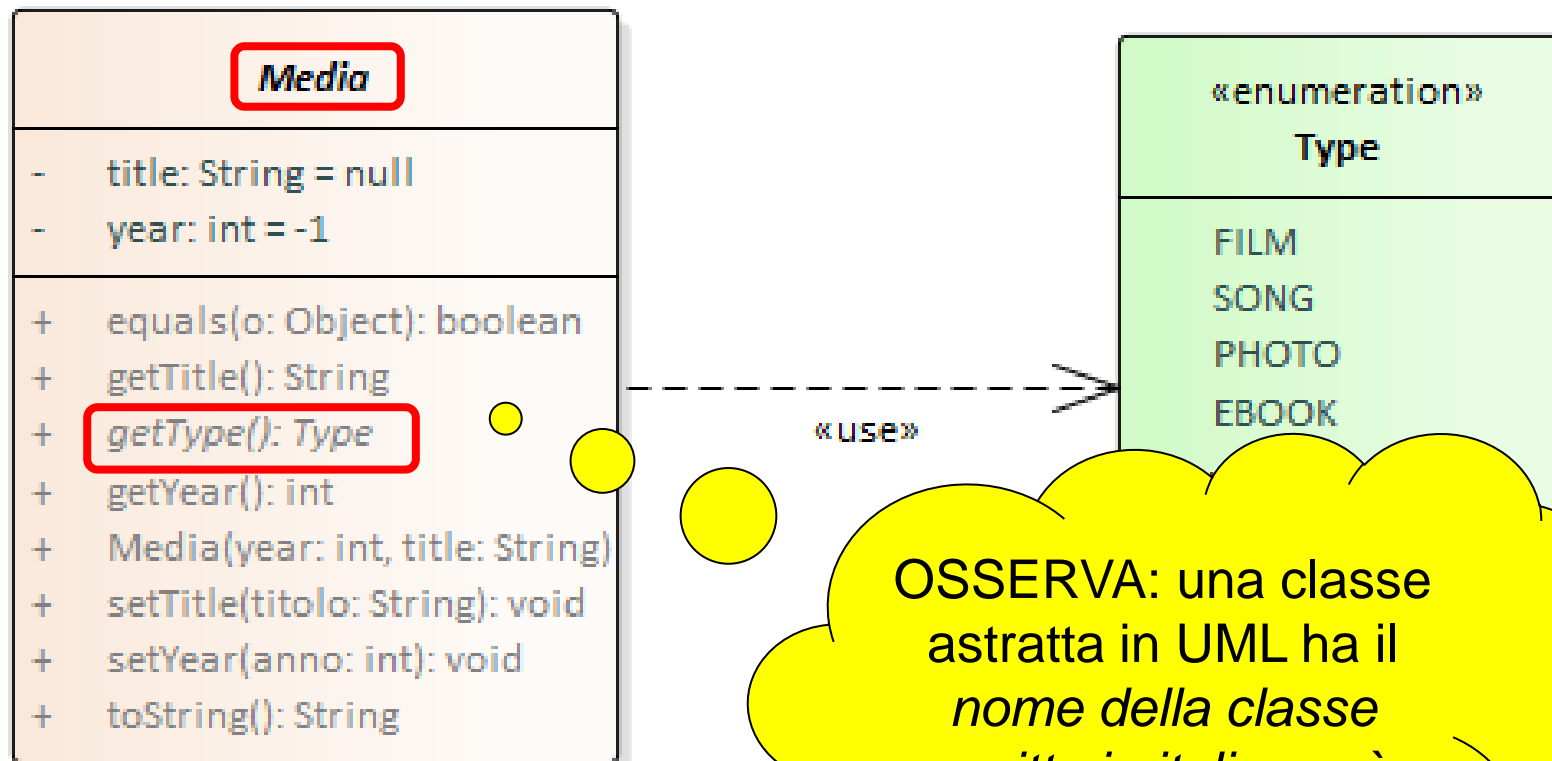
- Per ipotesi, ogni **Media** è caratterizzato da un **tipo**
- Come rappresentarlo?
 - OCCHIO: è una rappresentazione nostra, a livello di modello, non della macchina sottostante! (quindi, non possiamo usare «le classi»..)
- Due possibilità:
 1. una stringa → occorre sceglierne alcune e giurare di usare solo quelle (stando attenti a maiuscole, minuscole, equals...) *e se sbagliamo il compilatore non se ne può accorgere*
 2. un **enumerativo** → evita problemi con maiuscole/minuscole, garantisce check a compile time sull'uso corretto 😊
enum Type: EBOOK, FILM, PHOTO, SONG

Media



Media è una classe astratta: non avrebbe senso crearne istanze, né saremmo in grado di specificare **getType** per un generico **Media**

Media



OSSERVA: una classe astratta in UML ha il *nome della classe* scritto in *italic*, così come i *metodi astratti*

Film

- Il **Costruttore** prende in ingresso sei parametri
 - titolo, anno, regista, durata, elenco attori, genere
 - parte della costruzione è demandata alla classe base
- I **metodi accessor consentono di** ottenere/impostare informazioni sul componente, in particolare per:
 - ottenere/impostare il regista: **getDirector**, **setDirector**
 - ottenere/impostare la durata: **getDuration**, **setDuration**
 - ottenere/impostare la lista attori: **getActors**, **setActors**
 - ottenere/impostare il genere: **getGenre**, **setGenre**
- Questa classe inoltre ridefinisce, specializzandoli, i metodi per:
 - ottenere il tipo di media (**getType**): **restituisce la costante enumerativa FILM**
 - ottenere la rappresentazione in stringa del componente (**toString**)
 - stabilire se due **media** sono uguali (**equals**)

Film

Media

Film

- actors: String ([]) = null
 - director: String = null
 - duration: int = -1
 - genre: String = null
-
- + equals(o: Object): boolean
 - + Film(titolo: String, anno: int, regista: String, duration: int, attori: String[], genre: String)
 - + getActors(): String[]
 - + getDirector(): String
 - + getDuration(): int
 - + getGenre(): String
 - + getType(): Type
 - + setActors(attori: String[]): void
 - + setDirector(director: String): void
 - + setDuration(duration: int): void
 - + setGenre(value: String): void
 - + toString(): String

Song

- Il **Costruttore** prende in ingresso cinque parametri
 - titolo, anno, cantante, durata, genere
 - parte della costruzione è demandata alla classe base
- I **metodi accessor consentono di** ottenere/impostare informazioni sul componente, in particolare per:
 - ottenere/impostare il cantante: **getSinger**, **setSinger**
 - ottenere/impostare la durata: **getDuration**, **setDuration**
 - ottenere/impostare il genere: **getGenre**, **setGenre**
- Questa classe inoltre ridefinisce, specializzandoli, i metodi per:
 - ottenere il tipo di media (**getType**): **restituisce la costante enumerativa SONG**
 - ottenere la rappresentazione in stringa del componente (**toString**)
 - stabilire se due **media** sono uguali (**equals**)

Song

Media

Song

- duration: int = -1
 - genre: String
 - singer: String = null
-
- + equals(o: Object): boolean
 - + getDuration(): int
 - + getGenre(): String
 - + getSinger(): String
 - + getType(): Type
 - + setDuration(duration: int): void
 - + setGenre(value: String): void
 - + setSinger(singer: String): void
 - + Song(titolo: String, anno: int, singer: String, duration: int, genre: String)
 - + toString(): String

Photo

- Il **Costruttore** prende in ingresso quattro parametri
 - titolo, anno, lista di autori
 - parte della costruzione è demandata alla classe base
- I **metodi accessor consentono di** ottenere/impostare informazioni sul componente, in particolare per:
 - ottenere/impostare gli autori: **getAuthors**, **setAuthors**
- Questa classe inoltre ridefinisce, specializzandoli, i metodi per:
 - ottenere il tipo di media (**getType**): **restituisce la costante enumerativa PHOTO**
 - ottenere la rappresentazione in stringa del componente (**toString**)
 - stabilire se due `media` sono uguali (**equals**)

Photo

<i>Media</i>	
Photo	
-	authors: String [] = null
+	equals(o: Object): boolean
+	getAuthors(): String[]
+	getType(): Type
+	Photo(titolo: String, anno: int, authors: String[])
+	setAuthors(authors: String[]): void
+	toString(): String

Ebook

- Il **Costruttore** prende in ingresso quattro parametri
 - titolo, anno, lista autori, genere
 - parte della costruzione è demandata alla classe base
- I **metodi accessor consentono di** ottenere/impostare informazioni sul componente. In particolare per:
 - ottenere/impostare gli autori: **getAuthors, setAuthors**
 - ottenere/impostare il genere: **getGenre, setGenre**
- Questa classe inoltre ridefinisce, specializzandoli, i metodi per:
 - ottenere il tipo di media (**getType**): **restituisce la costante enumerativa EBOOK**
 - ottenere la rappresentazione in stringa del componente (**toString**)
 - stabilire se due `media` sono uguali (**equals**)

Ebook

Media

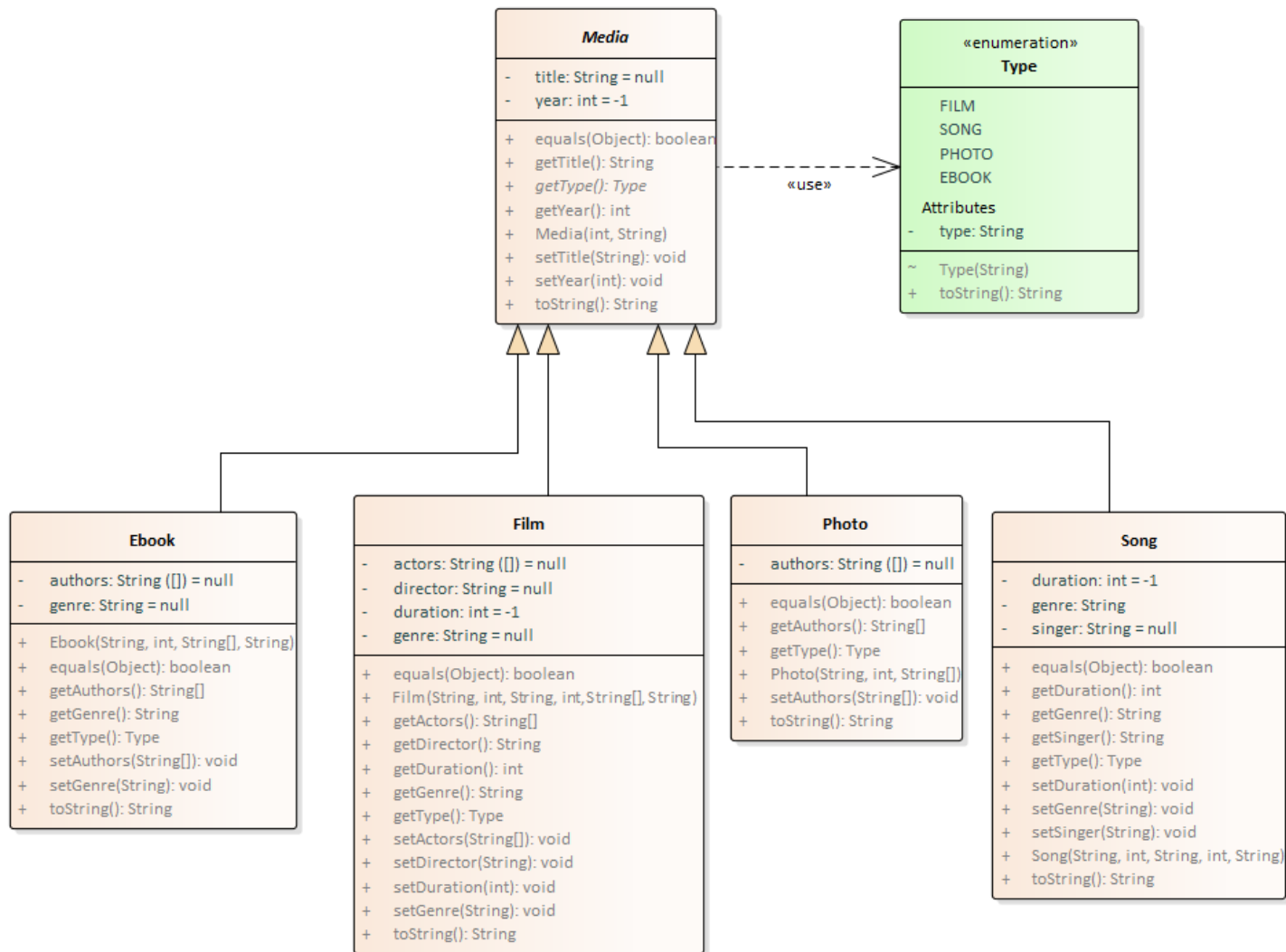
Ebook

- authors: String ([]) = null
- genre: String = null
- + Ebook(title: String, year: int, authors: String[], genre: String)
- + equals(o: Object): boolean
- + getAuthors(): String[]
- + getGenre(): String
- + getType(): Type
- + setAuthors(autori: String[]): void
- + setGenre(genre: String): void
- + toString(): String

Riflessioni

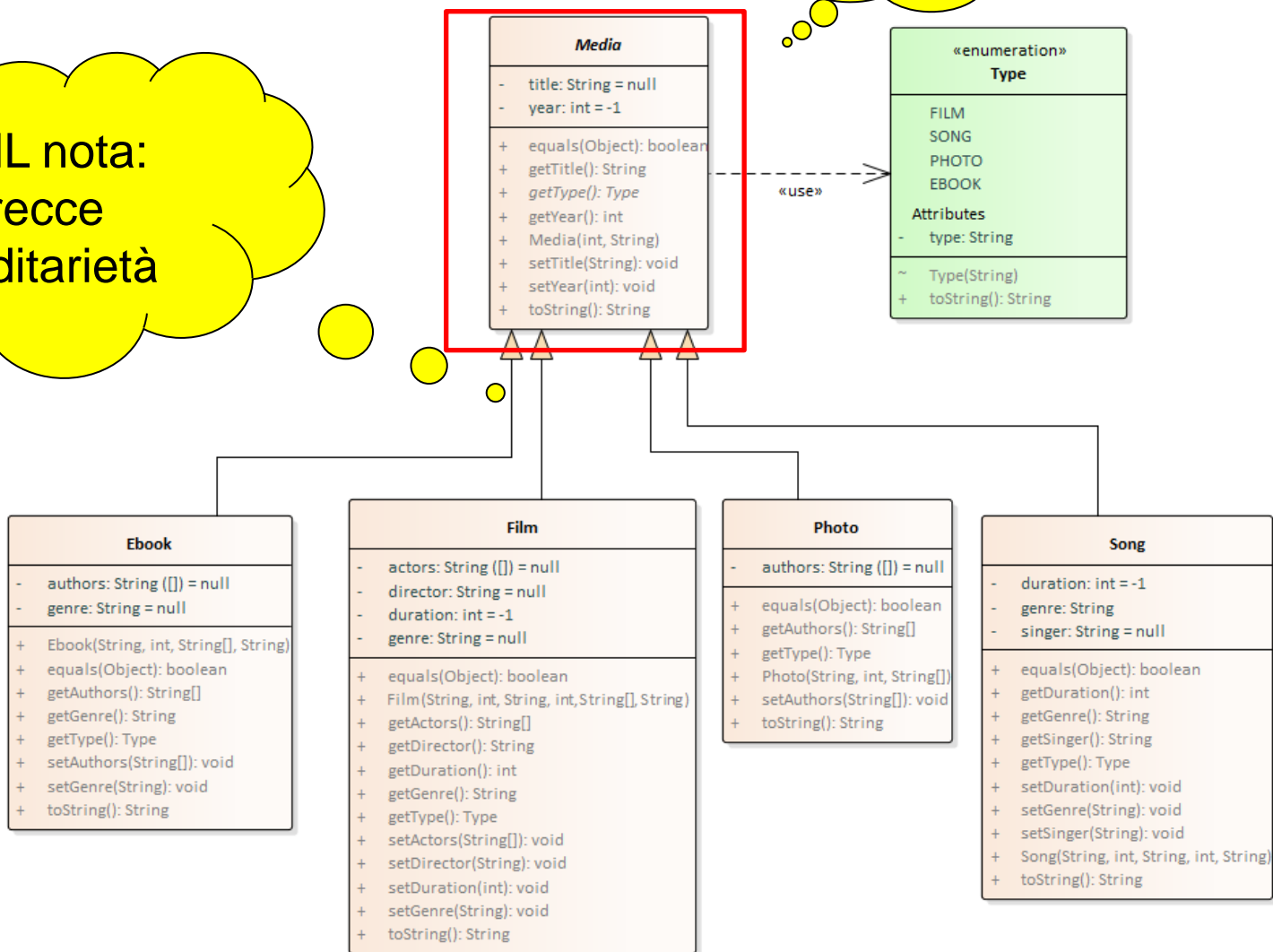
- *Sarebbe bello* che gli oggetti **Media** fossero **immutabili** (oggetti «valore»), MA sarebbe pretendere troppo:
 - è possibile/probabile che l'applicazione debba *modificare nel tempo qualche proprietà* di oggetti **Media** (per correggere sviste ortografiche, errori, ecc.)
 - inoltre, in un'ottica (molto) futura di introduzione di un supporto per la persistenza, sarebbe oneroso dover creare un nuovo oggetto ogni volta che si deve effettuare una modifica
- *Sarebbe bello* che nelle varie *liste di autori, elenco di attori*, etc. si potessero aggiungere/togliere **singoli elementi**
 - in futuro lo faremo (con le *Collection* di Java)
 - per ora useremo **array** e con essi ciò è oneroso
→ le liste di autori/attori verranno lette/assegnate *in blocco*

Model



Model Astratta

UML nota:
frece
ereditarietà





La questione di **equals**

- Poiché **equals** è ereditato dalla classe base **Object**, il metodo **equals** prevede come argomento un generico **Object**
- MA in realtà noi vogliamo confrontare oggetti **omogenei**
- Cosa deve accadere in situazioni come questa?

```
Song s = new Song (...);
```

```
Film f = new Film (...);
```

```
assert s.equals(f);
```

È meglio/giusto che **equals** esploda o restituisca **false**?

- **OVVIAMENTE**, ha senso che restituisca *false*, *senza far esplodere tutta l'applicazione solo per questo!*

Implementazione di **equals**

- Per questo, meglio NON usare un cast brutale...

```
public boolean equals(Object obj) {  
    Song c = (Song) obj;  
    return ...  
}
```

Cast brutale: se `obj` non è una `Song`, esplode

- ...ma preferire un **cast controllato**, dove la conversione di tipo avvenga solo "a colpo sicuro" → **operatore *instanceof***

```
public boolean equals(Object o) {  
    if (o instanceof Song c) {  
        return getSinger().equals(c.getSinger()) &&  
            getDuration() == c.getDuration() &&  
            getGenre() == c.getGenre() &&  
            super.equals(o);  
    }  
    return false;  
}
```

Instanceof esteso: non esplode mai (se non è una `Song`... pazienza).



L'operatore **instanceof**

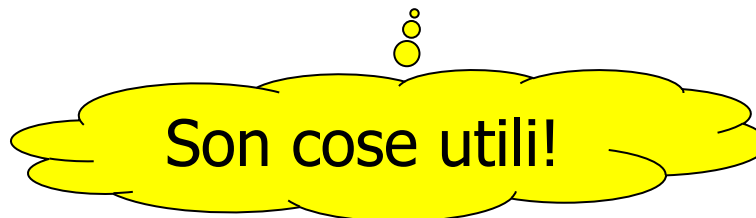
- L'operatore **instanceof** permette di sapere a runtime se un oggetto sia o meno *istanza di una certa classe*
- Wow, ci piace...

... ma non usiamolo a sproposito!

- L'operatore **instanceof** non va usato «solo perché c'è»!
 - in particolare, **non va usato come "sostituto pigro" di una buona soluzione polimorfa** (che quasi sempre esiste, *a pensarci bene*)
 - va usato **solo** quando una soluzione puramente polimorfa darebbe luogo ad una **architettura inutilmente complessa** per il problema - cioè, *raramente*

Un aiutino...

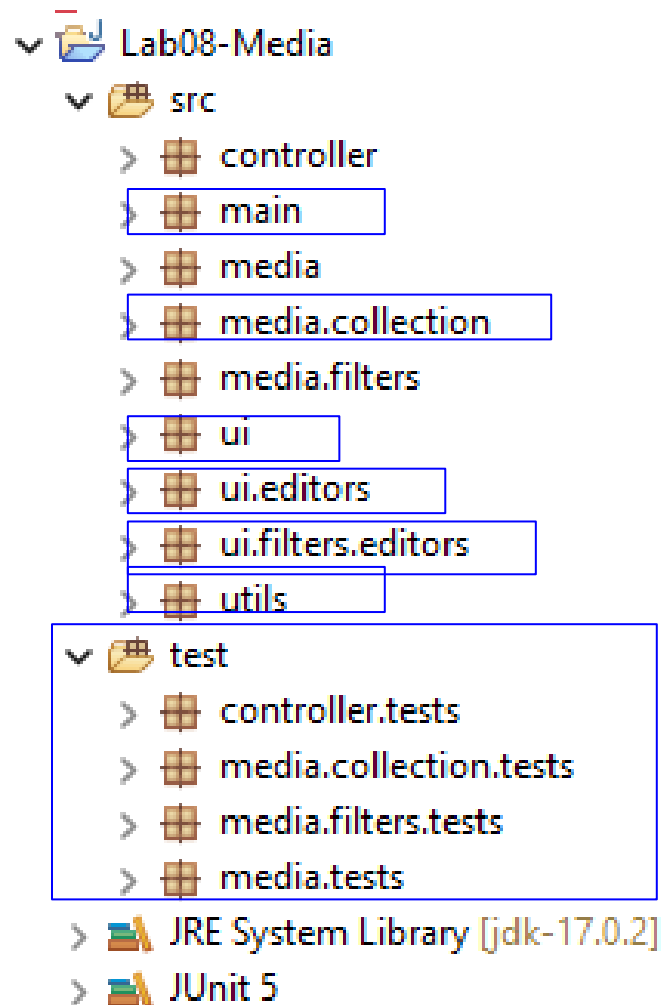
- La classe **StringUtils** del package **utils** contiene metodi per:
 - determinare se due array contengono le stesse stringhe, *non necessariamente nello stesso ordine* (altrimenti basterebbero i metodi standard della classe **Arrays**) → **areEquivalent**
 - cercare una stringa in un array di stringhe → **find**



Struttura applicazione e start kit

Lo start kit contiene svariati package:

- **main**: contiene la classe **Main** che attiva l'applicazione e mostra il menù iniziale
- **media**: destinato al model
- **media.collection**: ovvio..☺
- **controller**: destinato al controller
- **media.filters**: destinato ai filtri
- **ui.***: contengono la user interface
- **utils**: utility varie
 - **StringUtils** come descritta in precedenza
 - **Menu** che realizza il menù di scelta su console
 - **StdInput** che incapsula la lettura da tastiera



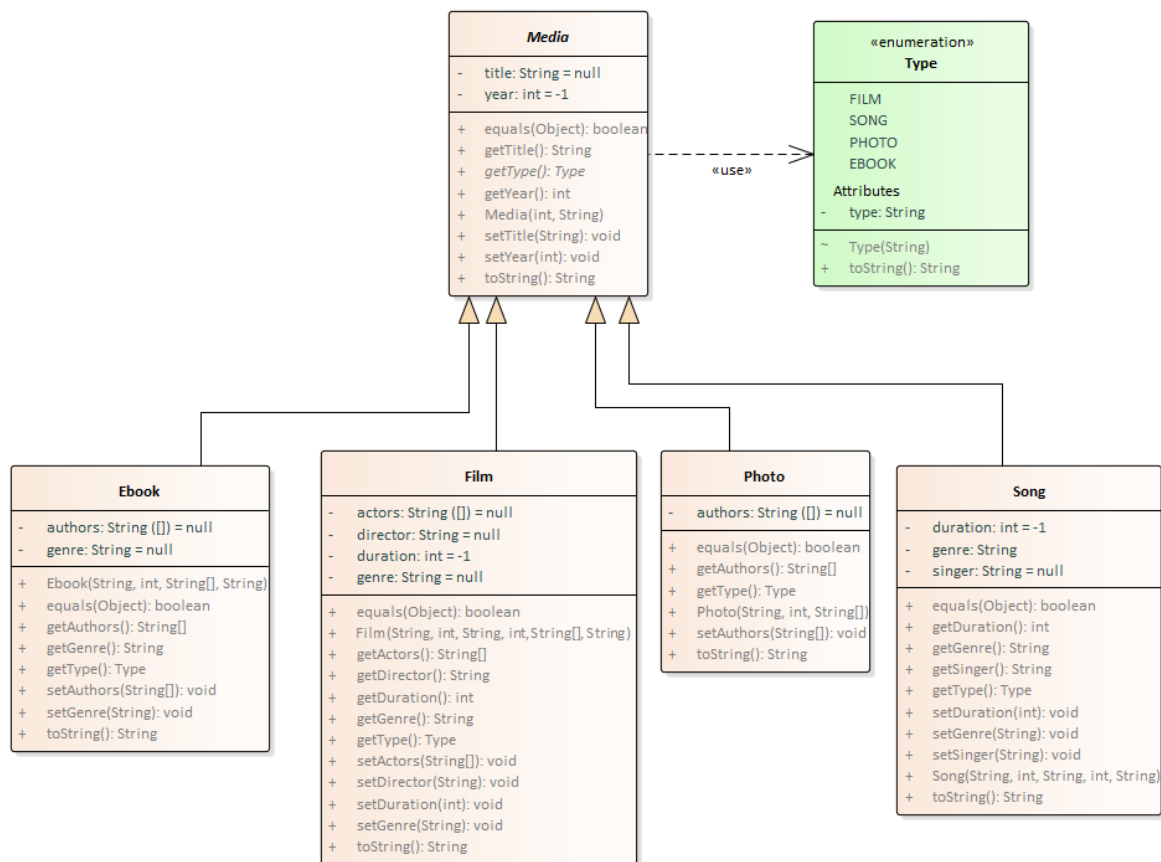
To do

VOI DOVETE REALIZZARE:

- **tutte le classi del modello** (`package media`)
 - `Media`, `Song`, `Ebook`, `Film`, `Photo`
- **Type** lo trovate pronto nello StartKit (`package media`)

NB: come sempre, è essenziale rispettare i nomi di package, classi e metodi.... **altrimenti i test...** ☹️

Model



Tempo a disposizione: 40 minuti



MediaCollection

- L'applicazione deve gestire insiemi di canzoni, foto, ebook..
ossia *collezioni di oggetti Media*
- **MediaCollection** è la classe che rappresenta ciò
 - ospita istanze di (qualsiasi sottoclasse di) Media
 - realizzabile usando come prototipo la analoghe collezioni precedenti
- **Punto chiave: come gestire i diversi tipi di contenuti?**
 - ogni oggetto in **MediaCollection** è formalmente un Media, ma in realtà sarà una **Song**, una **Photo**, etc.
 - **PER INSERIRE** non ci sono problemi: `Media m = new Song(...);`
 - ma... **PER ESTRARRE ?**

MediaCollection

- In estrazione, serve sapere di che tipo esatto è un certo Media?
DIPENDE!
- È vero che alcune operazioni *dipendono dal tipo specifico* dell'oggetto estratto...
 - ad esempio, per estrarre i Media *con una certa durata* si deve conoscere se il tipo dello specifico Media *abbia* una durata (non tutti ce l'hanno)
- ...ma è pur vero che **i vari tipi di Media sono in gerarchia!**
 - quindi, se si fa buon uso del **polimorfismo** ..
 - ...e, nei **pochi** casi in cui è inevitabile, dell'operatore **instanceof** ...
 - ... **si può scrivere codice *invariante*** rispetto al tipo 😊



MediaCollection

- **Costruttori**, quelli necessari...
- **Metodi** per la gestione della collezione:
 - aggiungere/eliminare un media:
 - **add**: prende in ingresso un Media: se non c'è spazio nell'array si raddoppia la dimensione (lo abbiamo già fatto, ricordate?)
 - **remove**: prende in ingresso una posizione e rimuove un Media in quella posizione (attenzione alle rimozioni in «mezzo» all'array..!)
 - ottenere un certo media:
 - **get**: prende in ingresso un indice e restituisce il Media in tale posizione
 - ottenere la dimensione (logica) della collezione:
 - **size**
 - ottenere la posizione di un media nella collezione:
 - **indexOf**: prende in ingresso un Media ne restituisce la posizione nella collezione – se non presente nella collezione, restituisce -1

OCCHIO: **indexOf** usa **equals** per i confronti

MediaCollection

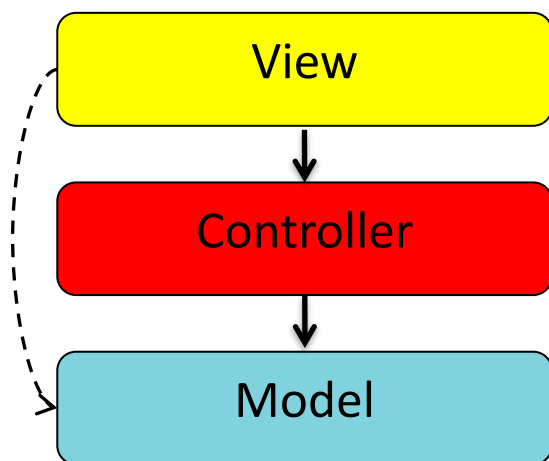
MediaCollection	
-	<u>DEFAULT_GROWTH_FACTOR: int = 2 {readOnly}</u>
-	<u>DEFAULT_PHYSICAL_SIZE: int = 10 {readOnly}</u>
-	innerContainer: Media ([])
-	size: int
+	add(f: Media): void
+	get(index: int): Media
+	indexOf(m: Media): int
+	MediaCollection(collection: Media[])
+	MediaCollection(physicalSize: int)
+	MediaCollection()
+	remove(index: int): void
+	size(): int
+	toString(): String

Già pronta nello start kit

Package `media.collection`

Architettura MVC

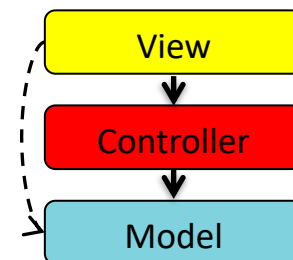
Riusiamo ancora una volta il pattern MVC



- **View:** gestisce la “rappresentazione visuale” dei “dati” e l’interazione con l’utente.
- **Controller:** controlla ed esegue le elaborazioni sui “dati”
- **Model:** costituisce l'insieme dei “dati” dell’applicazione

MediaController

- Ricorda: *solo il controller manipola i dati del model*
 - **MediaController** è il controller dell'applicazione
- Offre **metodi per la gestione della libreria**:
 - aggiungere/eliminare un Media:
 - **add**: prende in ingresso un Media, tenta di aggiungerlo alla collezione e restituisce un **boolean** che indica se ha avuto successo o meno
 - NB: non deve essere possibile inserire due volte lo stesso Media
 - **remove**: prende in ingresso un Media, tenta di eliminarlo dalla collezione e restituisce un **boolean** che indica se ha avuto successo o meno
 - NB: prima si controlla se il Media esiste, poi nel caso si cancella
 - ottenere l'elenco di tutti i Media
 - **getAll**: restituisce una copia della collezione dei Media



To do

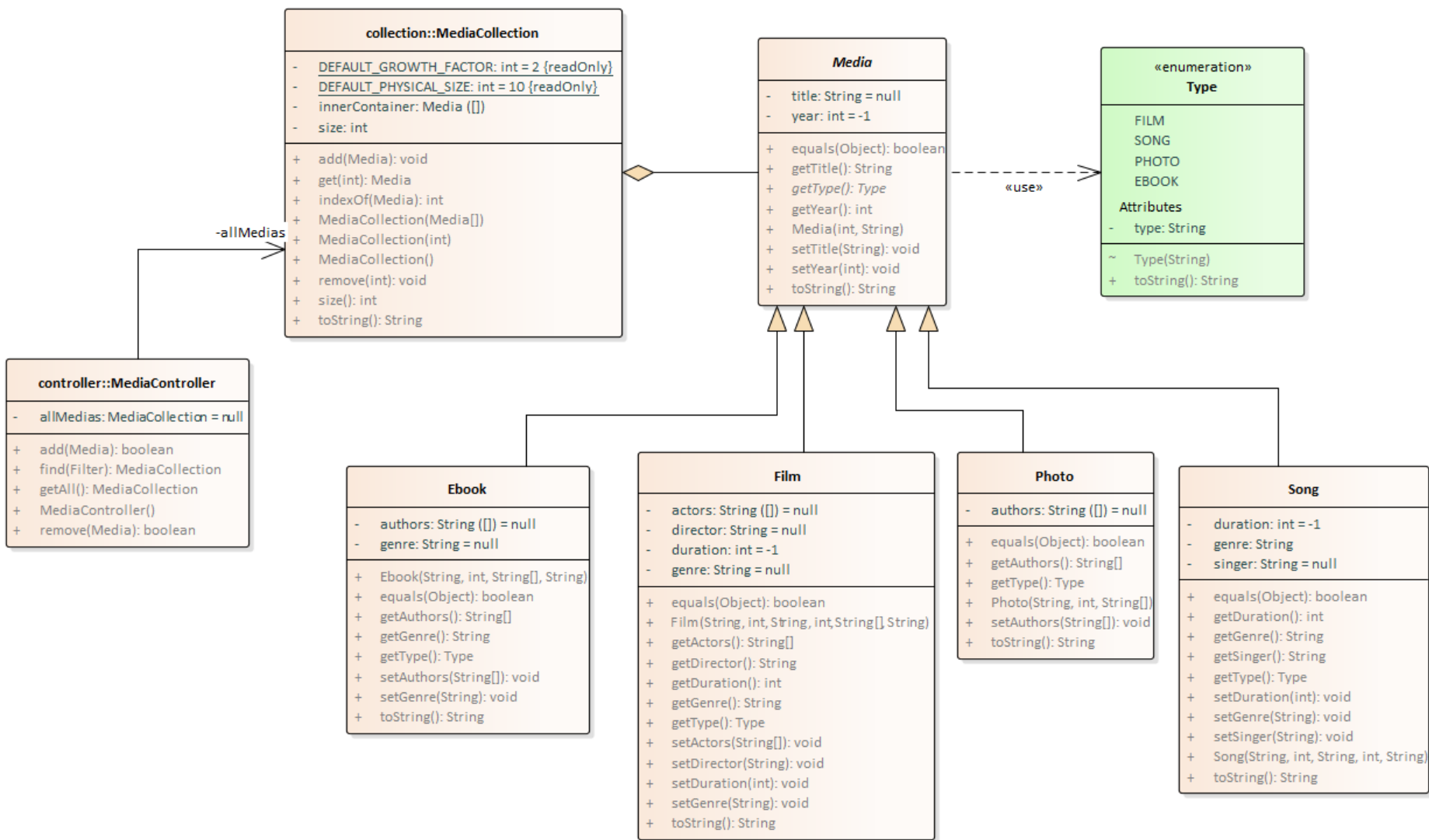
MediaController	
-	<code>allMedias: MediaCollection = null</code>
+	<code>add(m: Media): boolean</code>
+	<code>find(f: Filter): MediaCollection</code>
+	<code>getAll(): MediaCollection</code>
+	<code>MediaController()</code>
+	<code>remove(media: Media): boolean</code>

DA REALIZZARE:

- **il controller** (package **controller**)
- **tranne** il metodo **find** che aggiungeremo dopo

Tempo a disposizione: 20 minuti

Modello quasi completo



Ricerche di Media

- Come impostare le *ricerche* di Media ?
- Un paio di possibilità architetture:
 1. un metodo per ogni ricerca
 - per aggiungere una ricerca, si aggiunge un metodo nel controller
 - quindi, l'interfaccia utente va modificata ogni volta ☹

OPPURE...?

Ricerche di Media

- Osserviamo che **qualunque ricerca** si basa **sempre**, in realtà, sulla medesima **logica di fondo *invariante***:
 - PRIMA, si fa una scansione di tutto l'elenco dei Media
 - POI, si recuperano e si memorizzano (in un nuovo elenco) i soli Media identificati dalla ricerca come **«*interessanti*»**
- Nasce il concetto di ***strategia di ricerca***
 - tutte le ricerche sono fondamentalmente IDENTICHE
 - quello che cambia da una all'altra è solo la STRATEGIA!

Un altro pattern!
Welcome, STRATEGY 😊

Pattern STRATEGY

- **OBIETTIVO:** fare in modo che le strategie siano
 - incapsulate
 - **intercambiabili**
 - **trasparenti all'utilizzatore**
- **IDEA:** modellare ogni singola **strategia di ricerca** in una *opportuna entità*



Una gerarchia di
classi?

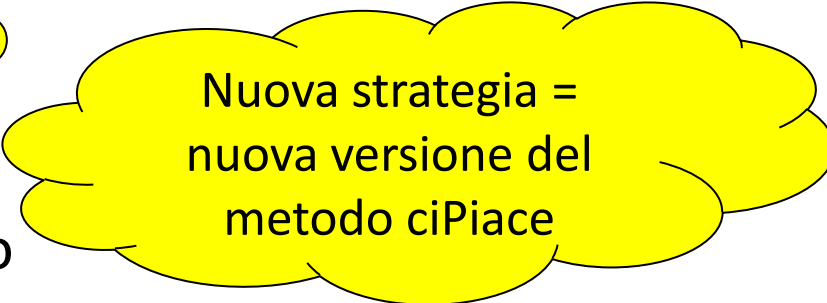
Metodi
polimorfi?

Strategy: un esempio "fai da te"

- Un modo *molto artigianale* per farlo potrebbe essere *incapsulare la strategia in un metodo*, così:

– metodo `ciPiace`:

```
MediaCollection res = new MediaCollection(...);  
for (Media m : allMedias) {  
    if (ciPiace(m)) res.add(m)  
}  
return res;
```



Nuova strategia =
nuova versione del
metodo `ciPiace`

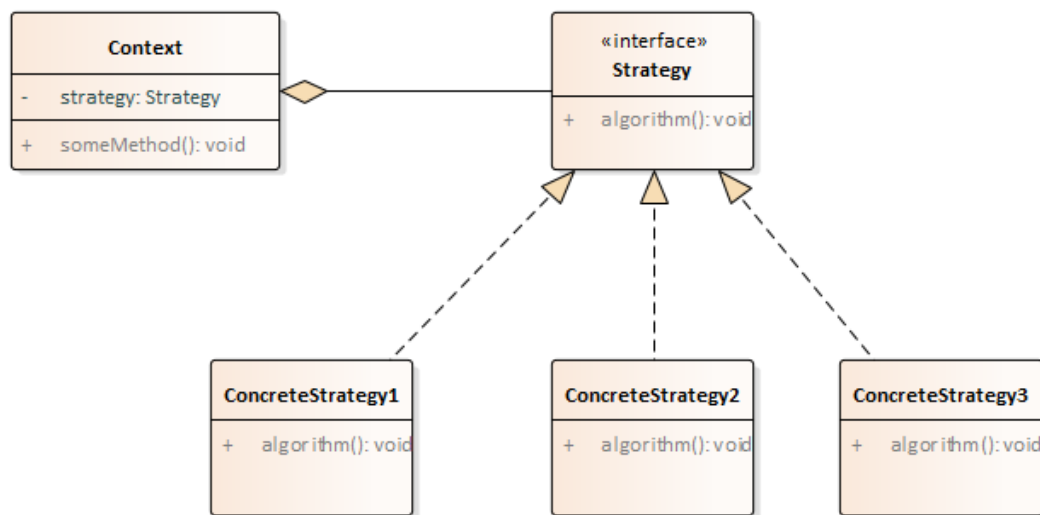
Idea valida, ma ancora allo stadio
un po' *troppo artigianale*...

Strategy

- Il pattern Strategy permette di **definire una famiglia di algoritmi, incapsularli e renderli intercambiabili**
 - INCAPSULARLI = sapere dove sono concettualmente contenuti = sapere dove "mettere le mani" per migliorarli, correggerli, etc.
 - RENDERLI INTERCAMBIABILI = poterli variare *indipendentemente dal contesto*
- Risultato: un meccanismo di *configurazione flessibile*
 - rendere possibile la scelta al momento dell'uso fra i tanti comportamenti disponibili

Strategy: ingredienti di base

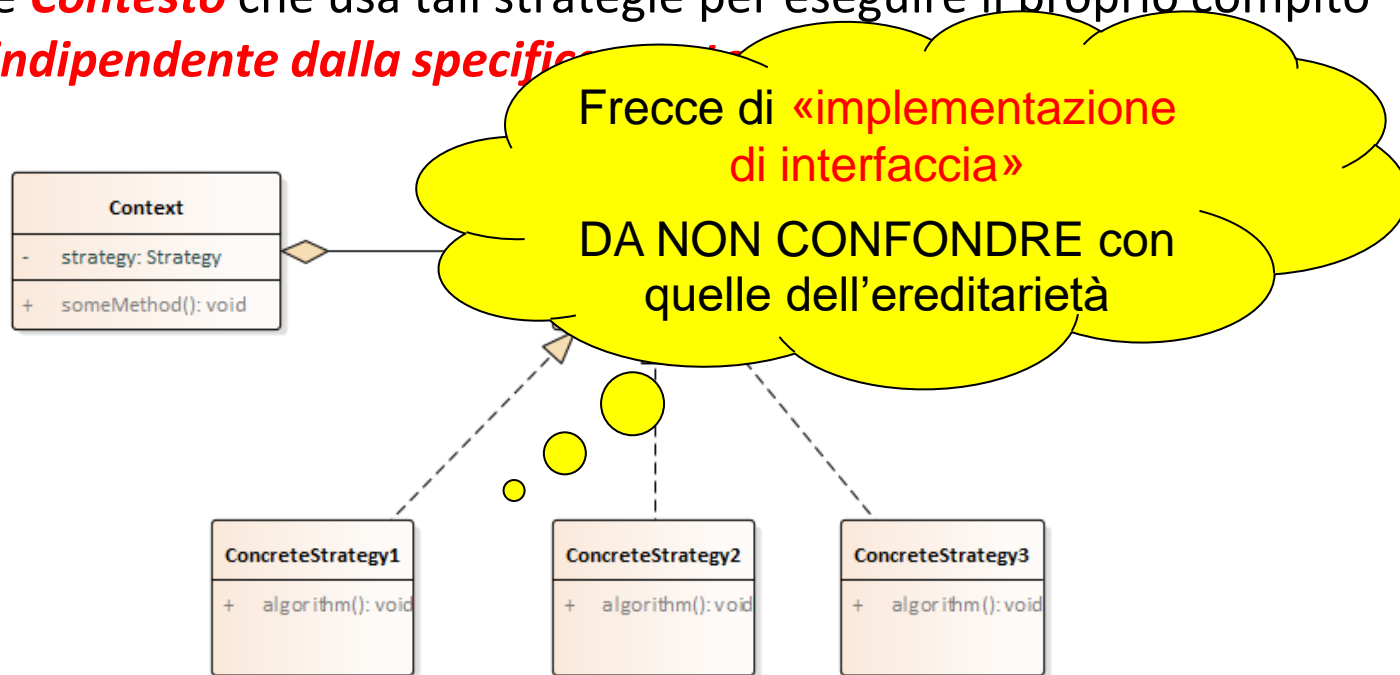
- Il pattern prevede tre ingredienti di base:
 - un'interfaccia **Strategy** che dichiara la *signature* per gli algoritmi concreti
 - un insieme di **classi che implementano le strategie** in modo polimorfo
 - una classe **Contesto** che usa tali strategie per eseguire il proprio compito *in modo indipendente dalla specifica strategia usata*



Strategy: ingredienti di base

- Il pattern prevede tre ingredienti di base:

- un'interfaccia **Strategy** che dichiara la *signature* per gli algoritmi concreti
- un insieme di **classi che implementano le strategie** in modo polimorfo
- una classe **Contesto** che usa tali strategie per eseguire il proprio compito **in modo indipendente dalla specifica della strategia**



Struttura delle strategie

- Le strategie (ex metodo **ciPiace**) devono essere:
 - intercambiabili
 - incapsulate
 - trasparenti all'utilizzatore

- **Come impostare un buon progetto?**

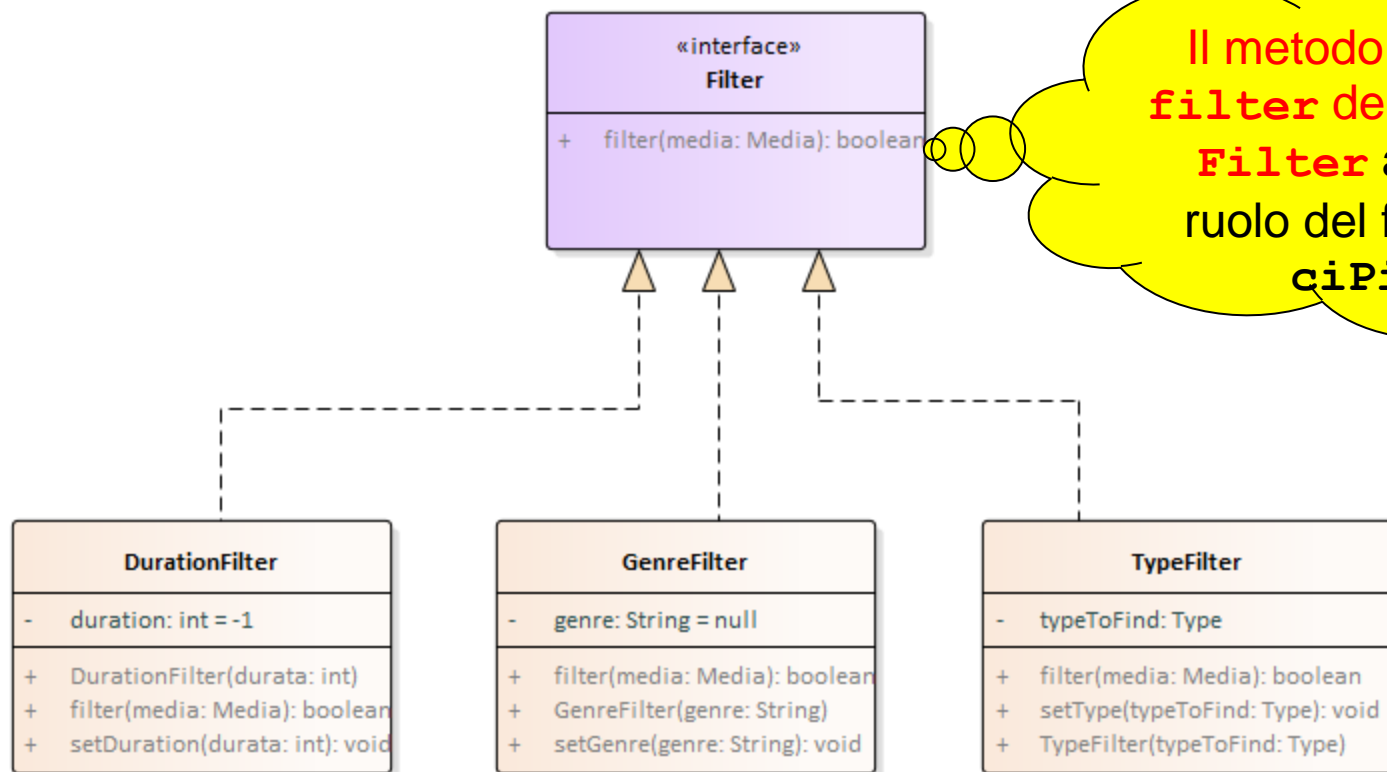
- una ulteriore gerarchia di classi..? . . .
- metodi polimorfi..? • •
- soprattutto, **organizzati come?** • • •

Sì!

Sì (era **cipiacce**)

Una gerarchia
di **Filtri** !

Gerarchia di filtri



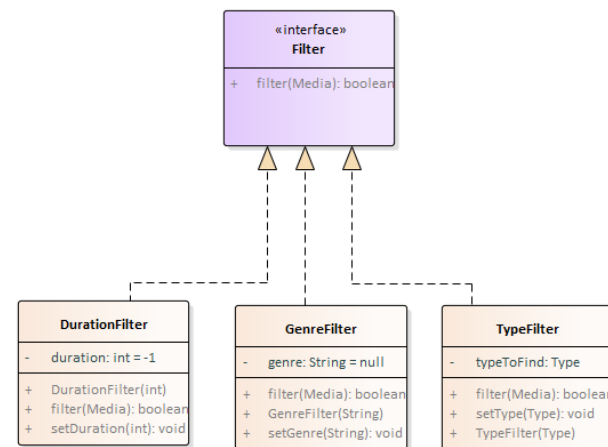
Il metodo polimorfo
filter dell'interfaccia
Filter assume il
ruolo del fu-metodo
ciPiace

Le classi
implementano
diverse strategie
(ossia, diversi filtri)

Gerarchia di filtri

- In particolare:

- in **DurationFilter**,
filter risponde **true** solo per i Media che hanno una *durata minore o uguale* a quella specificata.
Se la durata specificata è 0, risponde *true* per tutti i Media che hanno una (qualsiasi) durata.
- in **GenreFilter**,
filter risponde **true** solo per i Media *il cui genere è quello specificato*.
Se il genere è specificato con “”, risponde *true* per tutti i Media che hanno un (qualunque) genere.
- in **TypeFilter**,
filter risponde **true** solo per i Media il cui tipo è *identico* a quello specificato.



Come progettare i filtri?

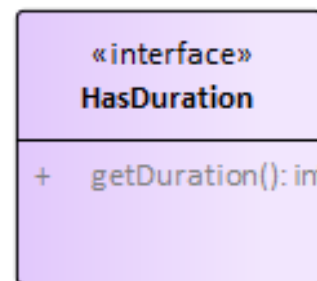
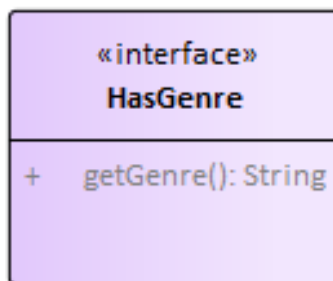
- IL DATO: in ingresso al metodo **filter** arriva un **Media**
- IL PROBLEMA: i filtri per durata o genere sono **costretti** a sapere quali tipi di media abbiano una durata o un genere..?
 - se sì, **allora i filtri diventano degli enormi switch** perché devono conoscere ogni possibile sottoclasse di Media..
 - .. e poi magari farci pure un **cast** per recuperare la proprietà
 - **FA SCHIFO!!** ☹ ☹
- Non sarebbe meglio invece **fattorizzare i concetti «avere una durata», «avere un genere», ecc.?**
 - **EH SÌ, CI PIACEREBBE ASSAI...!** 😊 😊
 - SI PUÒ FARE? E SE SÌ, COME..?

Filters & Features

- Ogni filtro ha le sue *peculiarità* (in inglese, *feature*)
 - chi la durata, chi il genere,...
- **IDEA:**
 - modellare esplicitamente la nozione di *peculiarità di un filtro*
 - implementare ogni filtro dicendo *quali feature gestisce*
- COME? Facile: con opportune *interfacce*!
 - *una interfaccia per ogni feature*
 - interfaccia **HasDuration**: cattura il concetto di *oggetto con durata*
 - interfaccia **HasGenre**: cattura il concetto di *oggetto con genere*
 - interfaccia **HasType**: cattura il concetto di *oggetto con tipo*
 - ... e ne possiamo inventare altre quando vogliamo!

Filters & Features

- Interfacce *feature*



- IPOTESI: i filtri lavorano usando solo questi concetti
 - il **filtro di durata** estrae e lavora *solo sugli oggetti che hanno una durata*, ossia **che implementano l'interfaccia HasDuration**
 - il **filtro di genere** estrae e lavora *solo sugli oggetti che hanno un genere*, ossia **che implementano l'interfaccia HasGenre**

Esempio: filtro di durata

Ricorda:

se durata è 0, deve selezionare tutti i Media che hanno una (qualsiasi) durata

```
1 package media.filters;
2
3 import media.*;
4
5 public class DurationFilter implements Filter {
6     private int duration = -1;
7
8     public DurationFilter(int durata) {
9         setDuration(durata);
10    }
11
12    public void setDuration(int durata) {
13        this.duration = durata;
14    }
15
16
17    @Override
18    public boolean filter(Media media) {
19        if (media instanceof HasDuration m) {
20            return (this.duration==0 || (m.getDuration() <= this.duration));
21        }
22        return false;
23    }
24 }
```

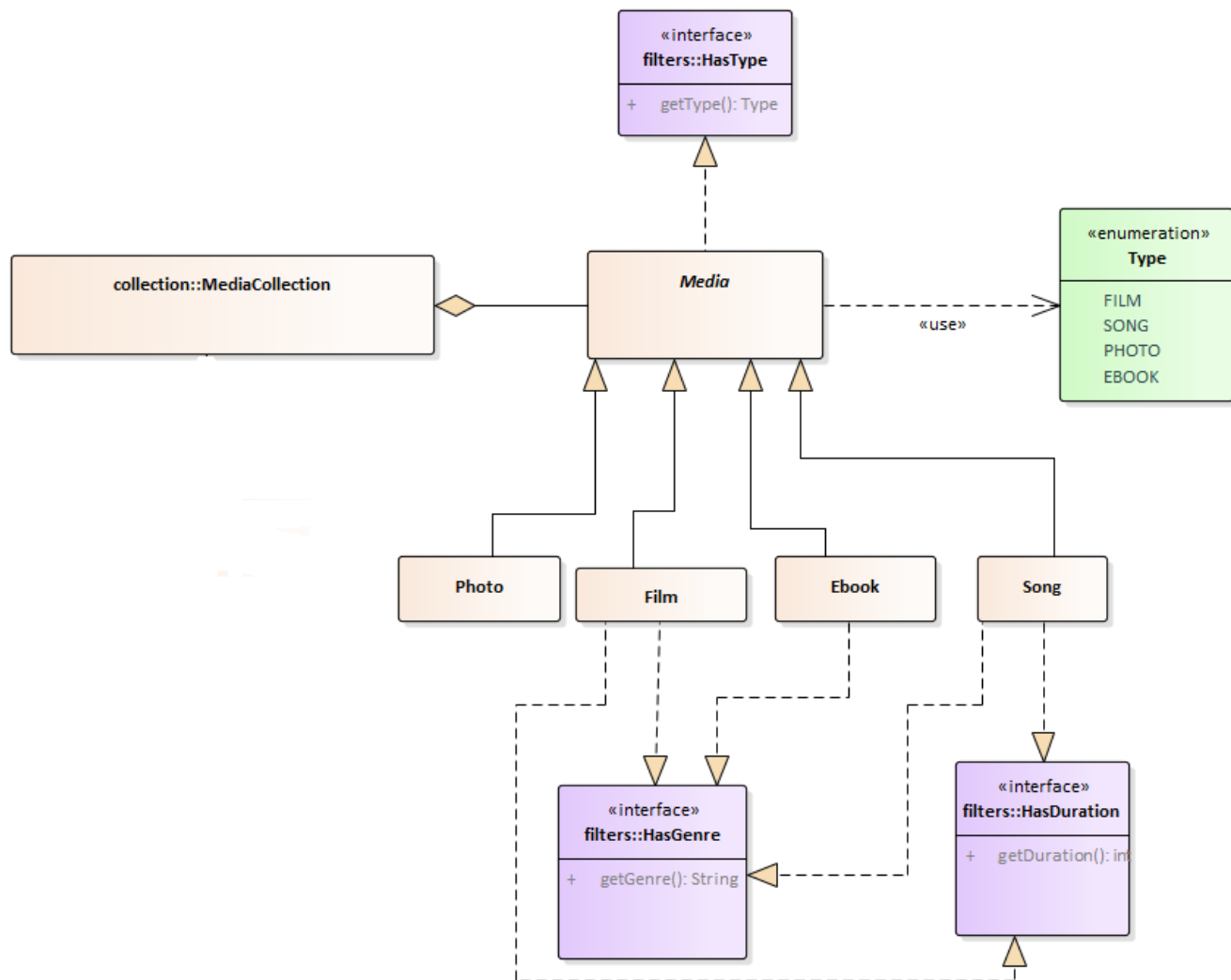
Implementa l'interfaccia
Filter quindi deve
implementare il metodo
filter



Filtro per tipo

- Il tipo è una caratteristica di cui sono dotati tutti i **Media**
 - tale caratteristica è esposta da un metodo polimorfo **getType** che restituisce un valore tra quelli dell'enumerativo **Type**
 - **enum Type:**
EBOOK, FILM, PHOTO, SONG

Il modello rivisto

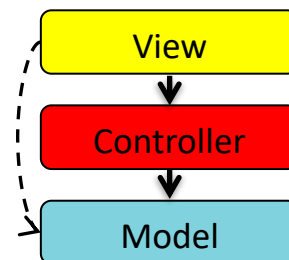


Filtri: vantaggi

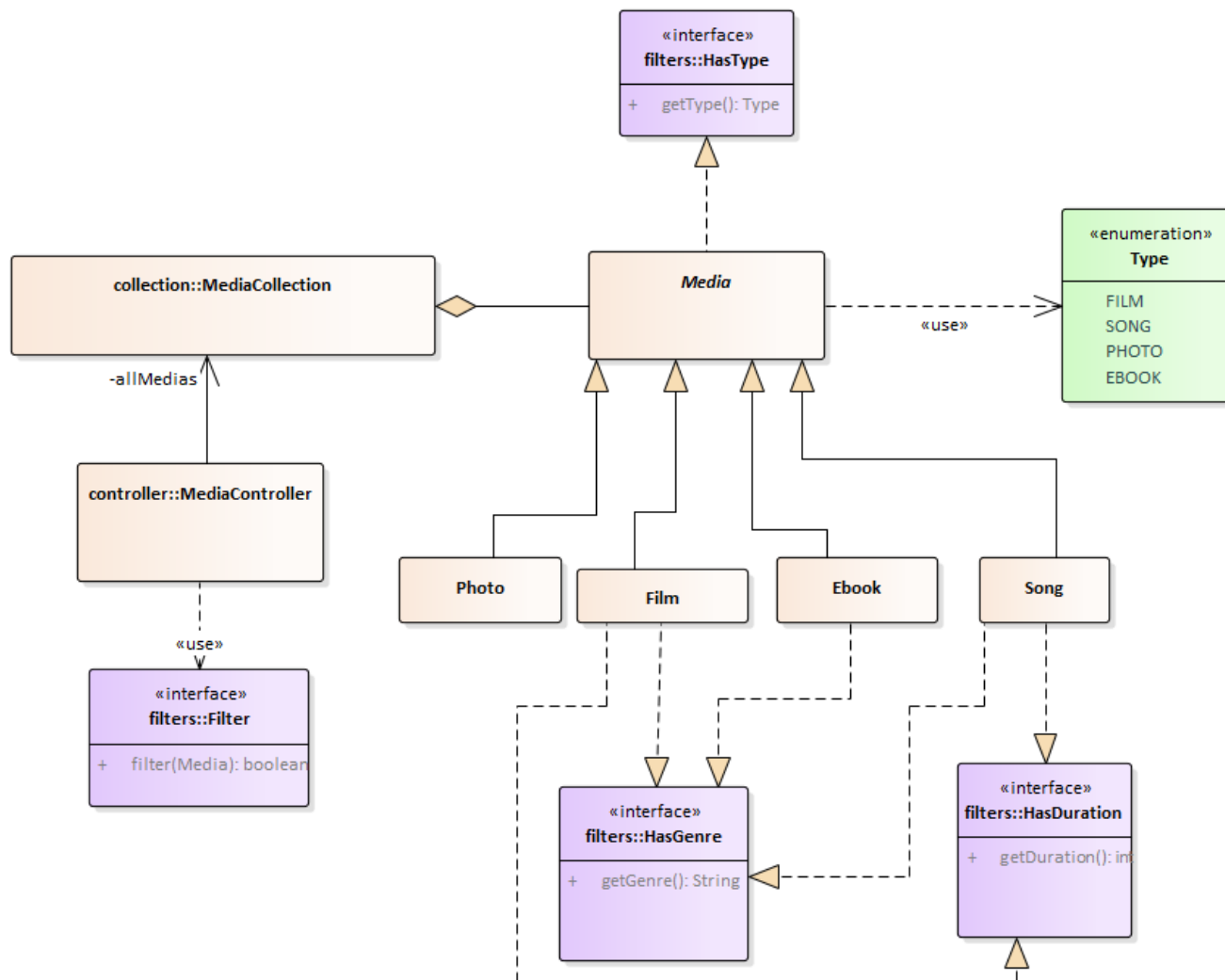
- L'utilizzatore della gerarchia di filtri **conosce solo l'interfaccia **Filter**** e nient'altro! 😊
 - l'utente non sa con quale (tipo concreto di) **Filter** abbia a che fare..
...e non gliene deve importare nulla!! 😊
 - egli si limita a invocare il metodo **filter**:
il polimorfismo si fa carico di attivare la "giusta" implementazione
- Le classi che implementano **Filter** sono **pienamente sostituibili**
 - definendo il proprio metodo **filter**, ogni *classe-filtro concreta* dà il "giusto" comportamento al "suo" specifico filtro

E ora.. finiamo il lavoro!

- Nella nostra app, sfruttiamo ciò nel controller
- Il nostro **MediaController**:
 - riceve nel costruttore la **MediaCollection** su cui opera
 - espone il metodo **find(Filter)** che effettua la ricerca
- A sua volta, tale metodo **find**:
 - riceve un **Filter**
 - crea una nuova **MediaCollection** per il risultato della ricerca
 - poi, per ogni **Media** contenuto nella **MediaCollection** originale:
 - invoca il filtro, ossia **chiama il metodo **filter** del **Filter** ricevuto**
 - se tale metodo risponde true, quel **Media** "ci piace"
→ va inserito nella **MediaCollection** risultato



Modello completo





To do

TROVATE GIÀ PRONTE:

- **tutte le interfacce** (package `media.filters`)
 - `Filter`, `HasDuration`, `HasGenre`, `HasType`

SONO DA FARE:

- **tutte le classi dei filtri** (package `media.filters`)
 - `DurationFilter`, `GenreFilter`, `TypeFilter`
- **il controller, relativamente al metodo `find`**
 - `MediaController`

Hey!



**KEEP
CALM
AND
HAPPY
CODING**