Fondamenti di Informatica T2
Lab10 - Bussy

Corso di Laurea in Ingegneria Informatica

Anno accademico 2021/2022

Prof. ROBERTA CALEGARI

Prof. AMBRA MOLESINI

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

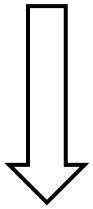
Scenario applicativo

«La compagnia EDMover, operante nella ridente ☺ cittadina di Dentinia, ha richiesto un'app da distribuire ai propri utilizzatori per la ricerca di percorsi di mezzi pubblici (bus, metro..) »

*«La CM Tech, giovane e brillante startup vincitrice dell'appalto, decide di creare l'app **Bussy**, ma senza vincolarla a Dentinia, in modo da poterla rivendere anche ad altre realtà cittadine..»*

Dallo scenario all'Analisi del Dominio

*«La CM Tech, giovane e brillante startup vincitrice dell'appalto, decide di creare l'app **Bussy**, ma senza vincolarla a Dentinia, in modo da poterla rivendere anche ad altre realtà cittadine..»*



Per sviluppare un'applicazione *indipendente dalla specifica città* occorre operare quella che in Ingegneria del Software si chiama «***Analisi del Dominio***»

Analisi del Dominio

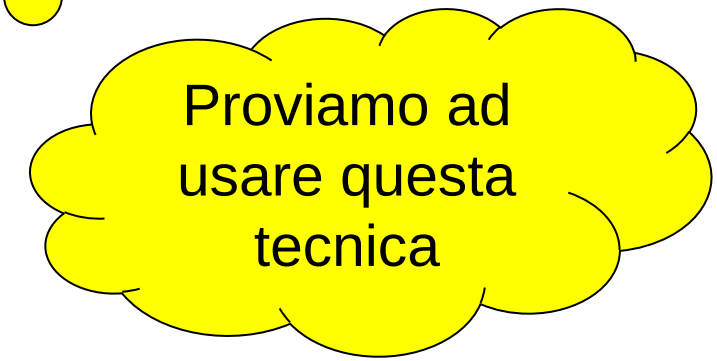
- L'Analisi del Dominio è svolta all'interno della fase di *Analisi dei Requisiti*, che a sua volta costituisce il *primo passo* del processo di ingegnerizzazione di un sistema software
- Obiettivo dell'Analisi del Dominio è definire la porzione del mondo reale rilevante per il sistema
- Tale analisi si focalizza sull'analisi dei *requisiti di dominio,* che:
  - derivano dal dominio di applicazione del sistema
  - solitamente includono una *terminologia propria* del dominio del sistema e/o si *riferiscono ai suoi concetti*
  - vanno *analizzati con grande cura* perché *riflettono i fondamenti* del dominio dell'applicazione

Analisi del Dominio

- L'Analisi del Dominio permette di *gestire la complessità* intrinseca del mondo reale
  - *ignorando gli aspetti non rilevanti* per lo scopo attuale
  - *concentrandosi* su quelli che invece lo sono
- Uno dei risultati è la prima versione del *vocabolario dei concetti*
  - costruito sulla base delle entità che si trovano nei requisiti
  - dando loro un *significato preciso ed univoco*

Analisi del Dominio

- L'Analisi del Dominio può essere effettuata anche considerando un *gruppo di sistemi* afferenti alla *stessa area applicativa*
- Esempi di aree applicative:
  - il controllo del traffico aereo
  - la gestione aziendale
  - le operazioni bancarie
  - …

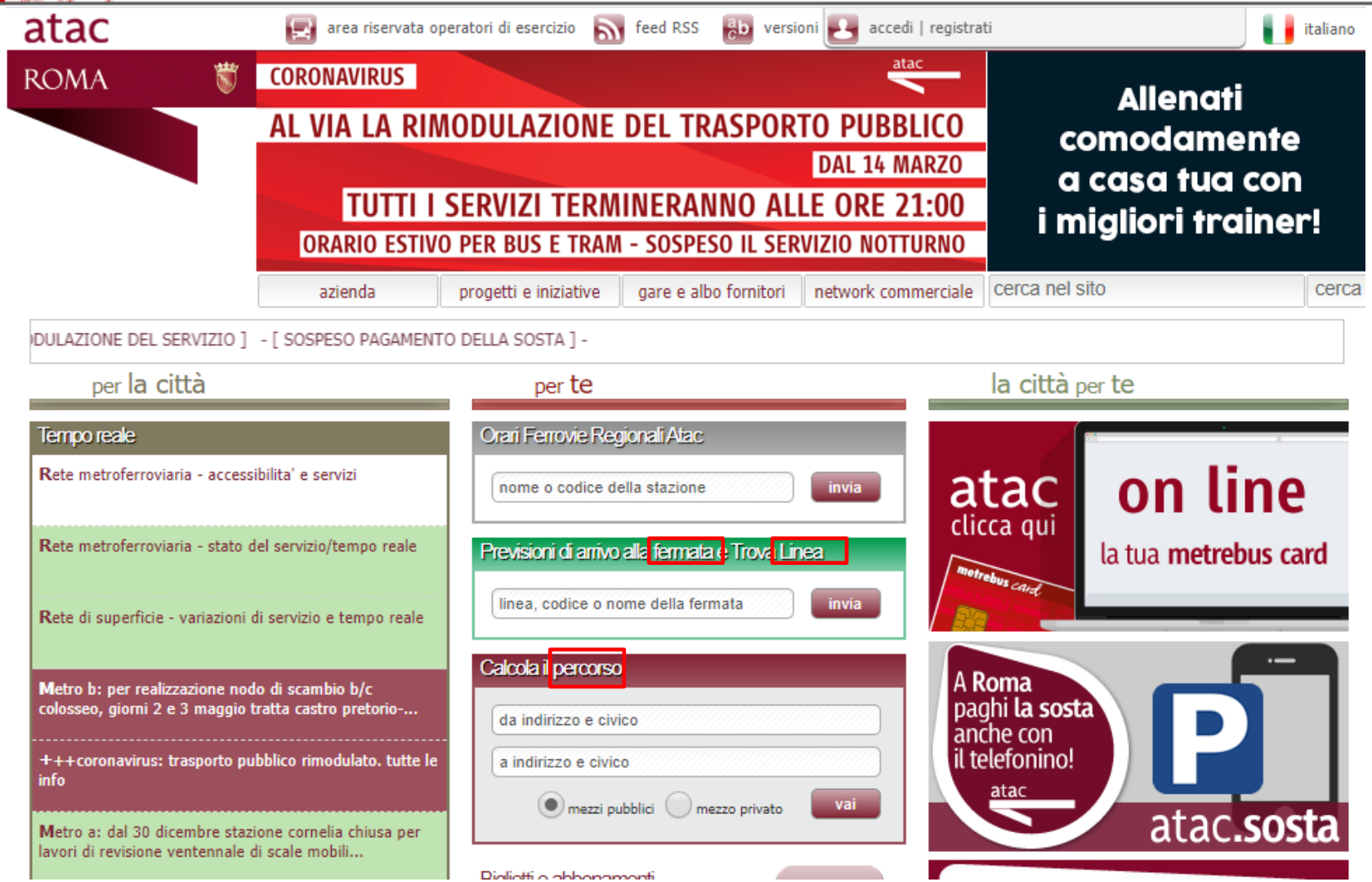


- In tal caso, è possibile
  - identificare entità e comportamenti comuni a tutti i sistemi
  - realizzare schemi di progettazione e componenti software riutilizzabili nei diversi sistemi

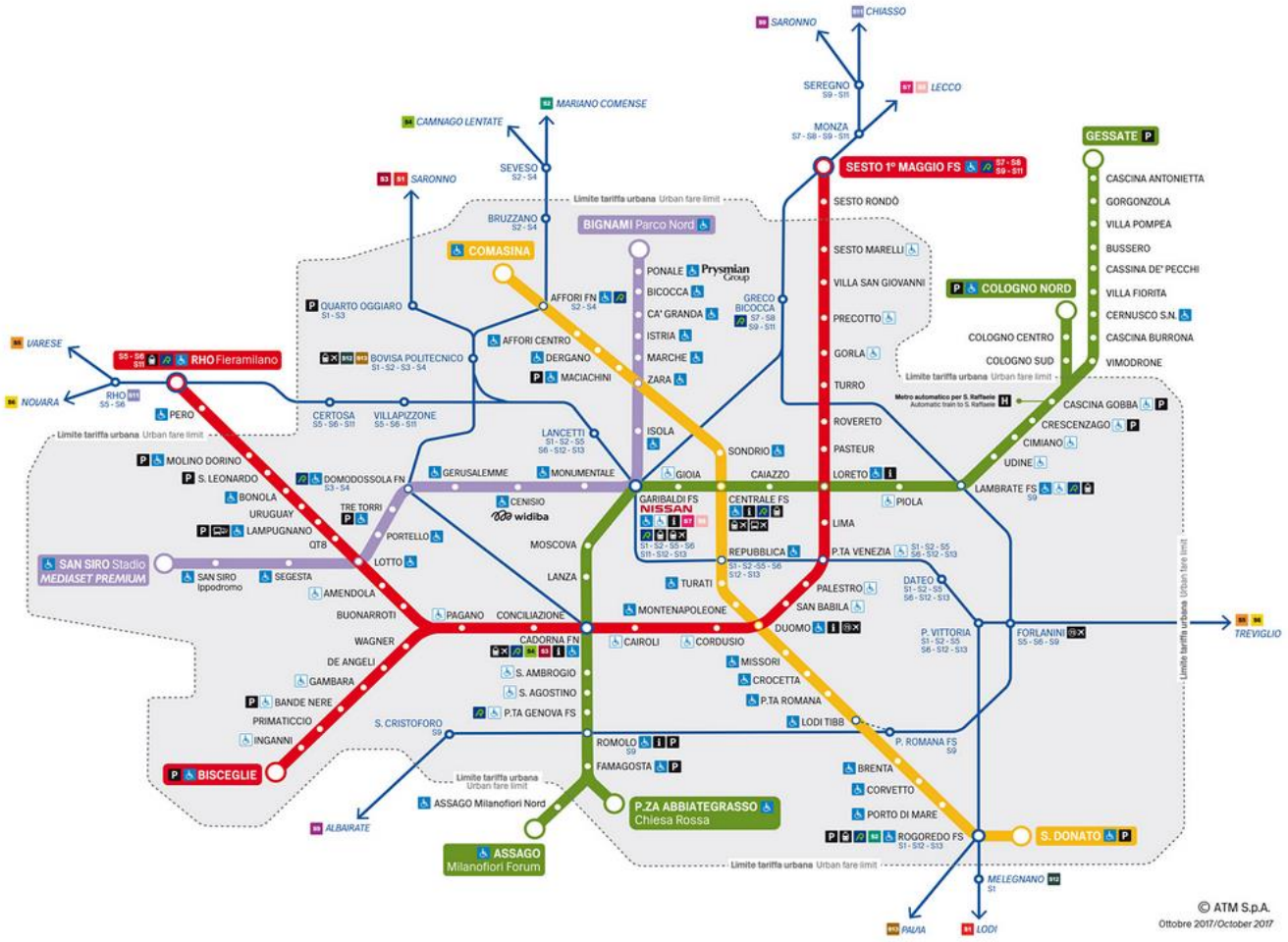
Analisi del dominio

Roma - Autobus

Analisi del dominio

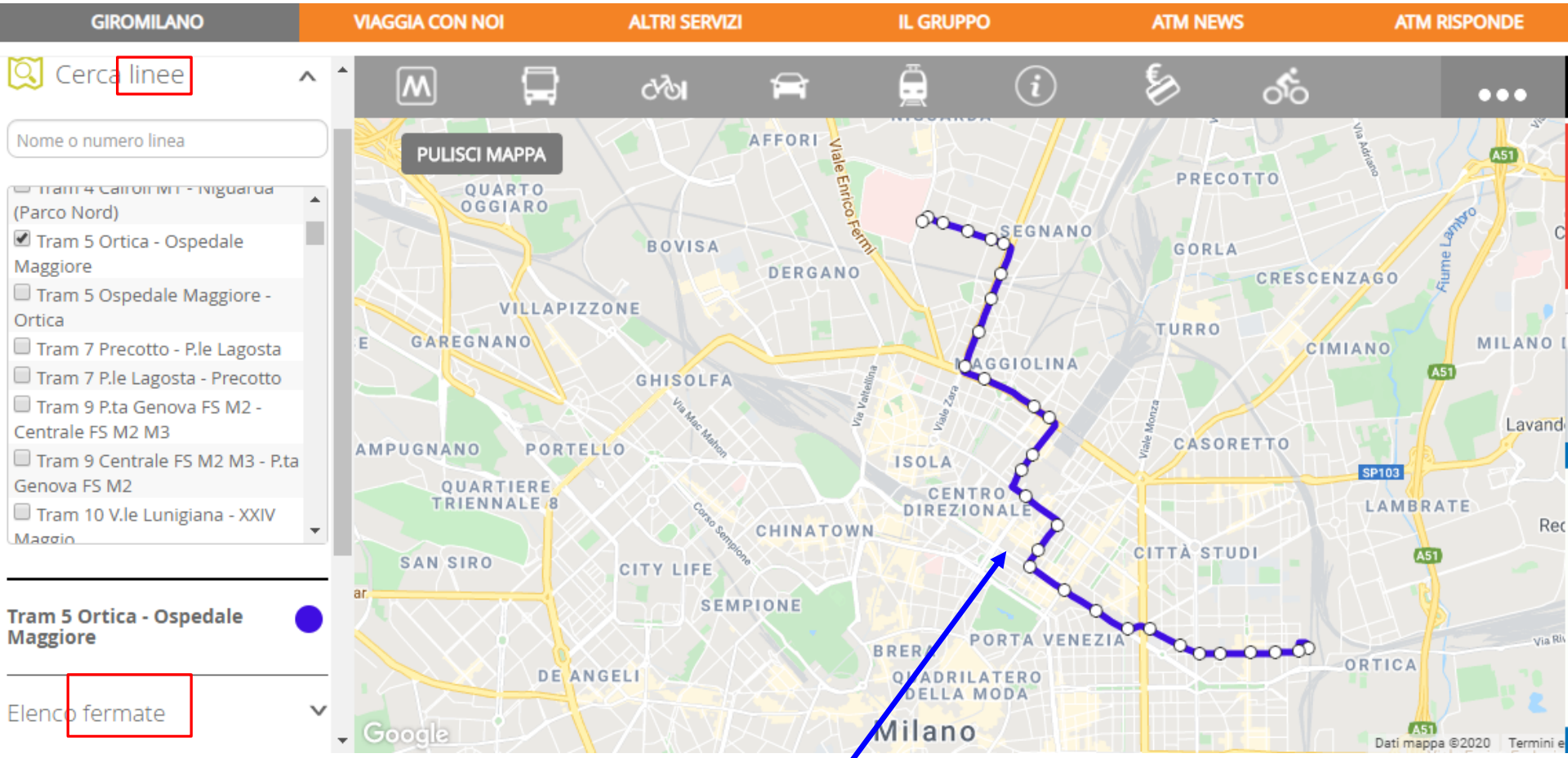


Analisi del dominio



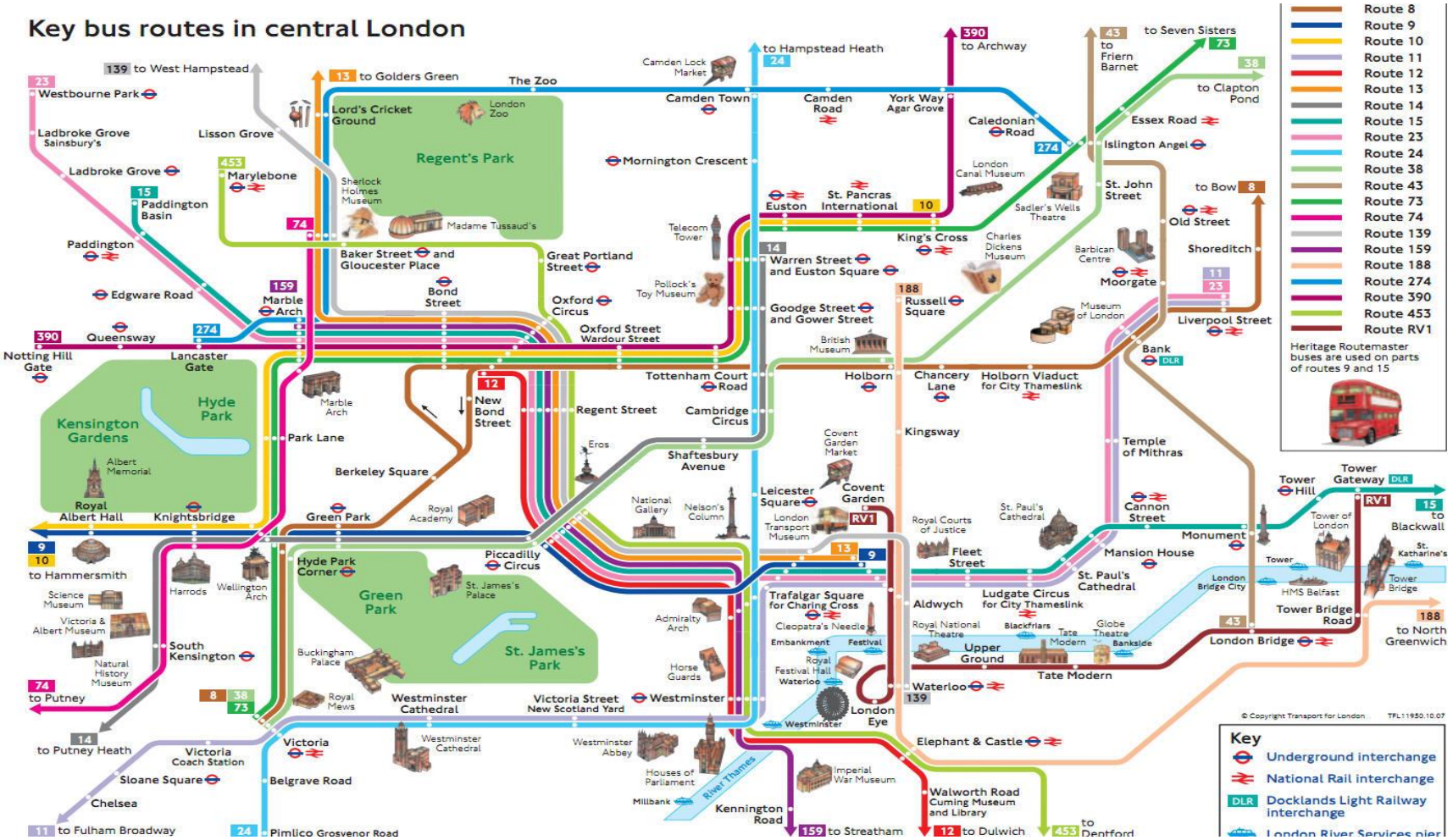
Milano - Metro

Analisi del dominio



Percorso

Analisi del dominio



Londra

Analisi del dominio

Lines | Stations | Add favourites

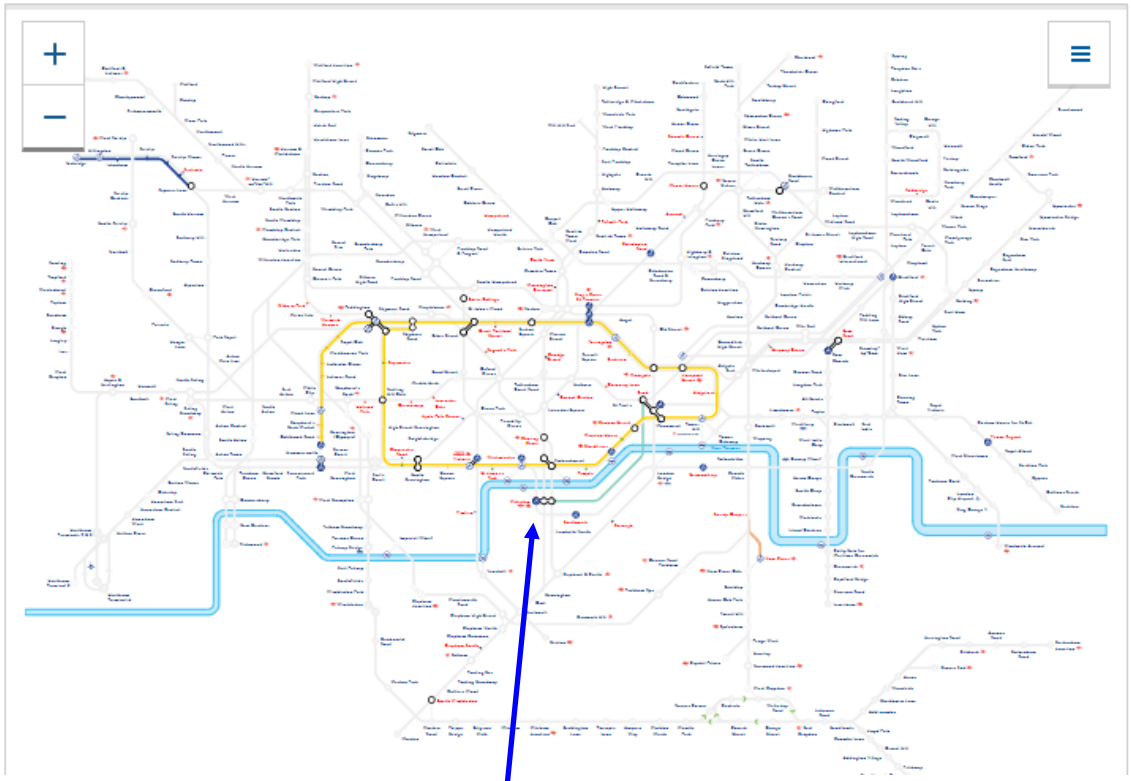
Bakerloo	Special service

Bakerloo Line: A 10 minute service is operating between Elephant & Castle and Queen's Park. A 30 minute service is operating between Queen's Park and Harrow & Wealdstone due to operational restrictions. Public transport should only be used for essential journeys.

Replan your journey

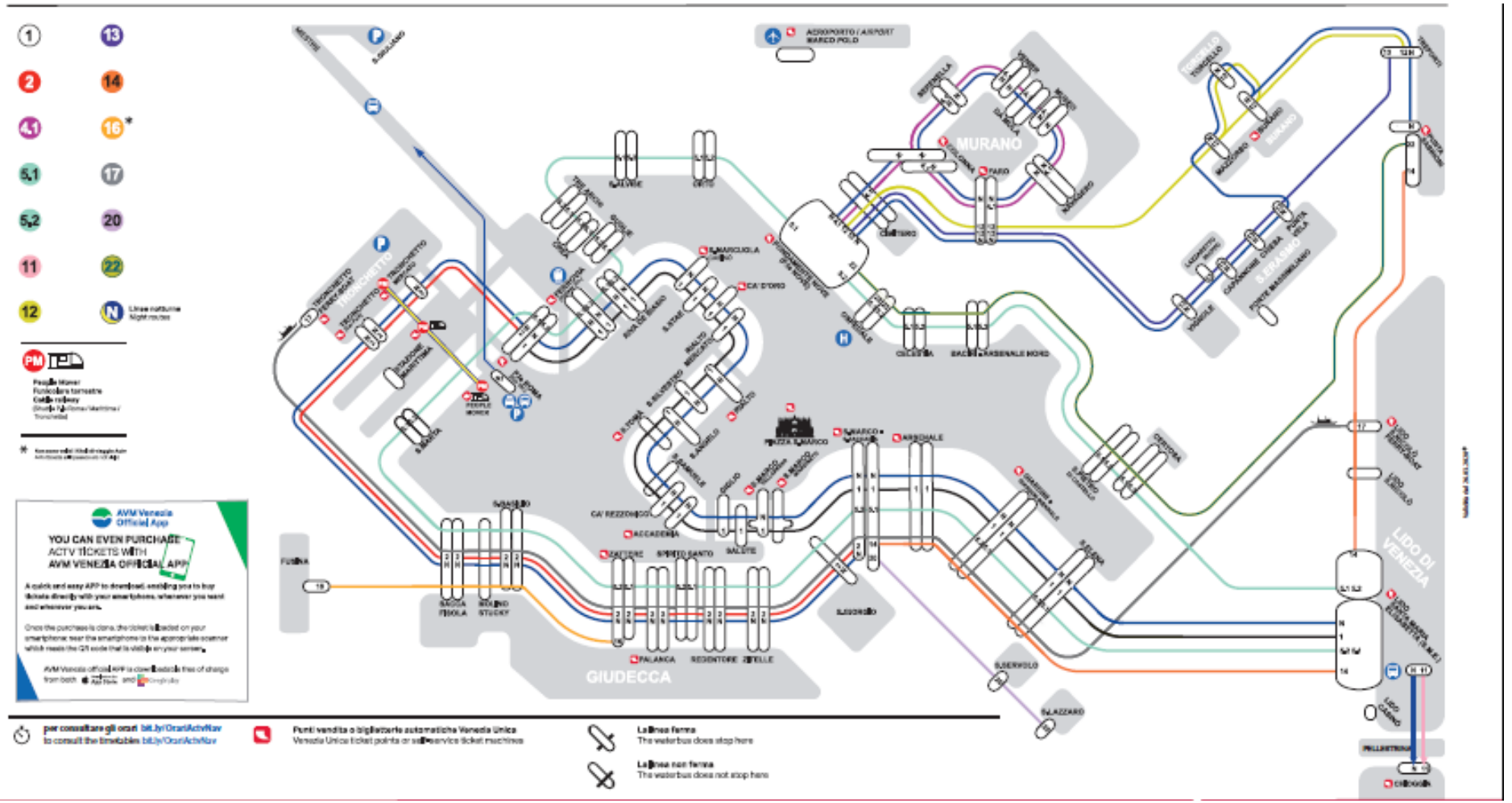
Close status

Central	Special service
Circle	Planned closure Service closed



Percorso

Analisi del dominio



Venezia - Traghetti

Analisi del dominio

Orari del servizio di navigazione / Waterborne transport service timetable

Linea	Percorso	Tabella orari	variazioni
1	P.LE ROMA - FERROVIA - RIALTO - S.MARCO - LIDO S.M.E	APRI	
2	S. MARCO / S. ZACCARIA - GIUDECCA - TRONCHETTO - P.LE ROMA		
4.1	NUOVO PERCORSO: FONDAMENTE NOVE - MURANO- FONDAMENTE NOVE		

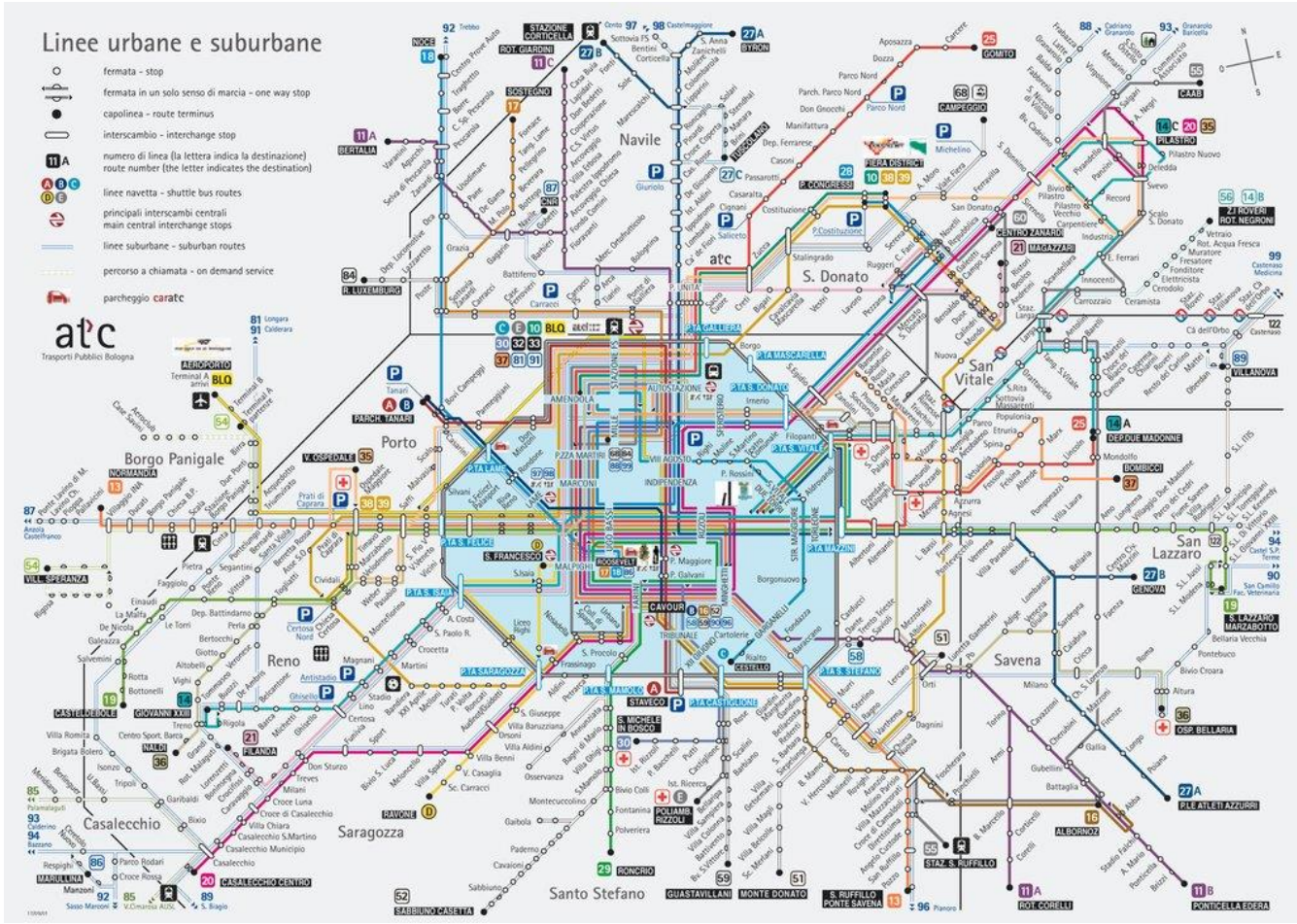
LINEA 1 P.LE ROMA - FERROVIA - RIALTO - S.MARCO - LIDO S.M.E.

Corse provenienti da Giudecca/Tronchetto come linea 2

IN VIGORE DAL 23 MARZO 2020					dalle		ai minuti		alle
P.LE ROMA "F"	**05:01**	**05:21**	**05:41**	**06:01**	**06:21**	**1**	**21**	**41**	**23:41**
FERROVIA "B"	05:04	05:24	05:44	06:04	06:24	4	24	44	23:44
RIVA DE BIASIO "B"	05:07	05:27	05:47	06:07	06:27	7	27	47	23:47
S.MARCUOLA "B"	05:10	05:30	05:50	06:10	06:30	10	30	50	23:50
S.STAE	05:12	05:32	05:52	06:12	06:32	12	32	52	23:52
CA' D'ORO	05:14	05:34	05:54	06:14	06:34	14	34	54	23:54
RIALTO MERCATO	05:17	05:37	05:57	06:17	06:37	17	37	57	23:57
RIALTO "D"	05:20	05:40	06:00	06:20	06:40	20	40	0	00:00
S.SILVESTRO	05:23	05:43	06:03	06:23	06:43	23	43	3	00:03
S.ANGELO	05:25	05:45	06:05	06:25	06:45	25	45	5	00:05
S.TOMÀ SOSPESA	-	-	-	-	-	-	-	-	-
CA' REZZONICO	05:30	05:50	06:10	06:30	06:50	30	50	10	00:10
ACCADEMIA "A"	05:32	05:52	06:12	06:32	06:52	32	52	12	00:12
GIGLIO	05:35	05:55	06:15	06:35	06:55	35	55	15	00:15
SALUTE	05:37	05:57	06:17	06:37	06:57	37	57	17	00:17
S.MARCO Vallaresso "B"	05:40	06:00	06:20	06:40	07:00	40	0	20	00:20
S.MARCO - S.ZACCARIA "D"	05:34	05:44	06:04	06:24	06:44	07:04	44	4	24	00:24
ARSENALE "B"	05:37	05:47	06:07	06:27	06:47	07:07	47	7	27	00:27
GIARDINI "B"	05:40	05:50	06:10	06:30	06:50	07:10	50	10	30	00:30
S.ELENA "C"	05:44	05:54	06:14	06:34	06:54	07:14	54	14	34	00:34
LIDO S.M.E. "D"	**05:49**	**05:59**	**06:19**	**06:39**	**06:59**	**07:19**	**59**	**19**	**39**	**00:39**

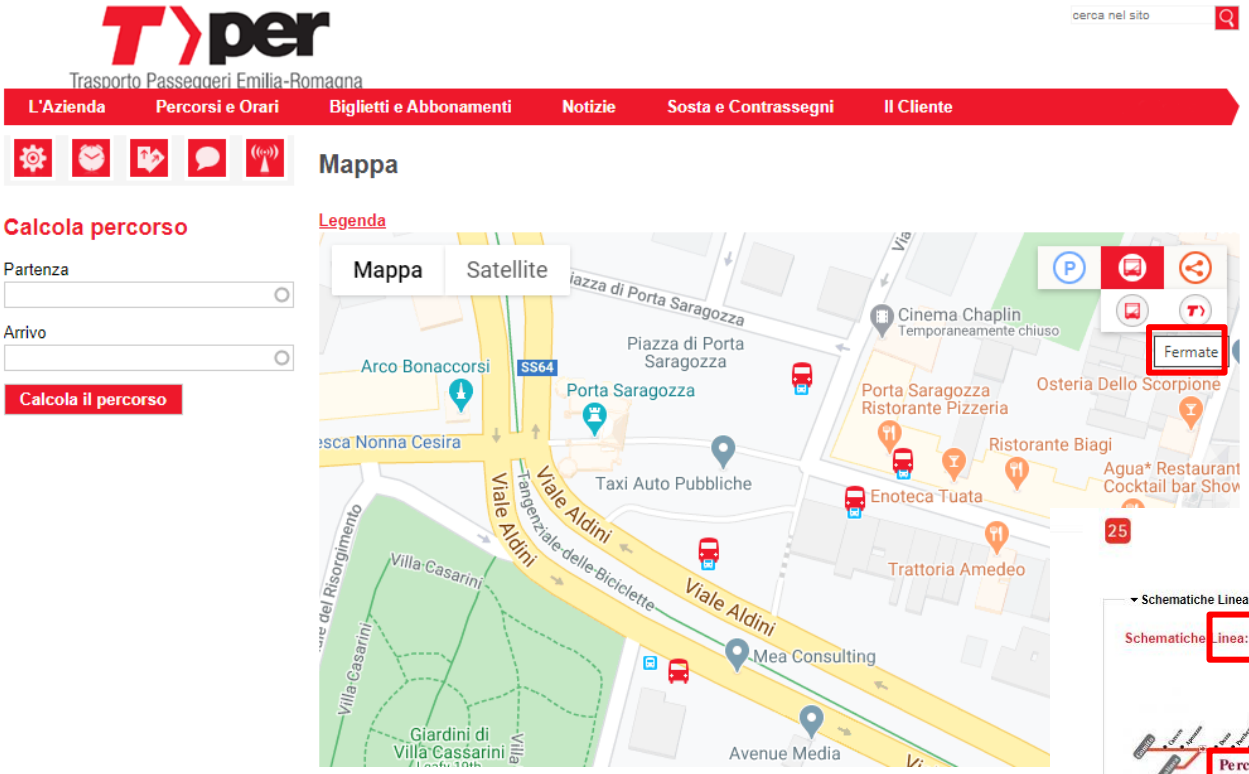
Fermate

Analisi del dominio



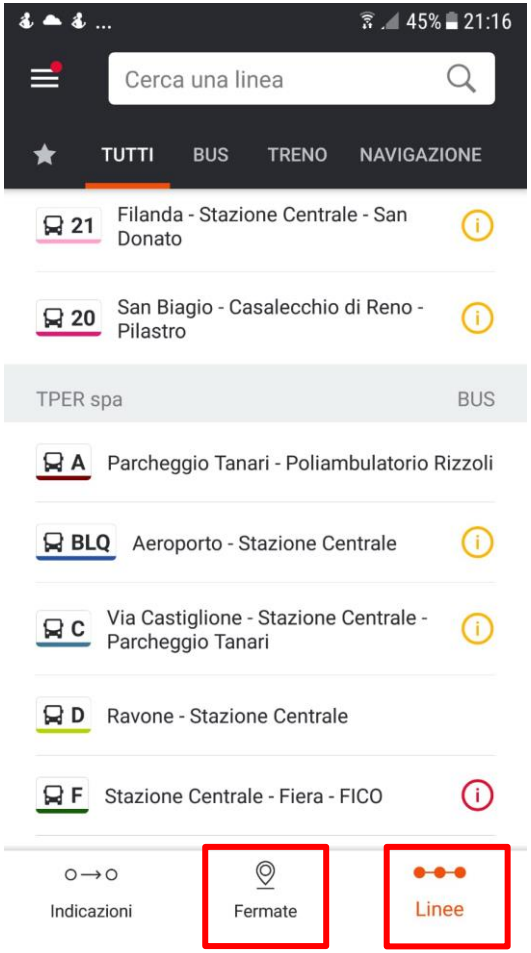
Bologna - Autobus

Analisi del dominio

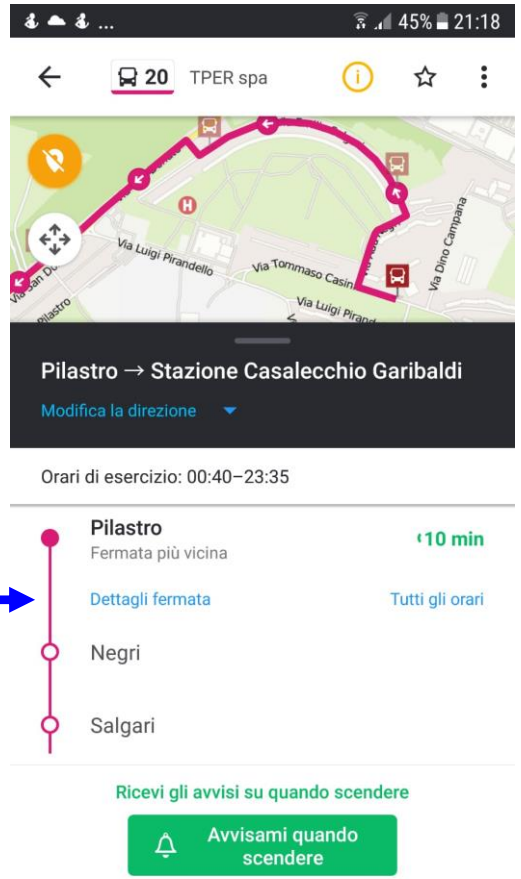


Percorso

Analisi del dominio



Percorso

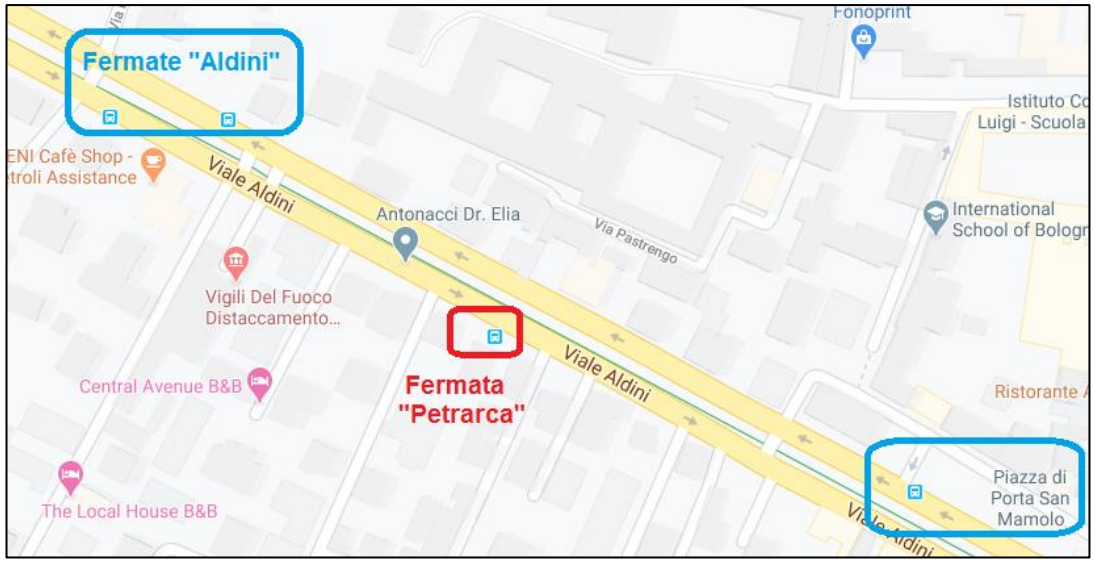


Moovit - Bologna

Esempio 1 (caso a)

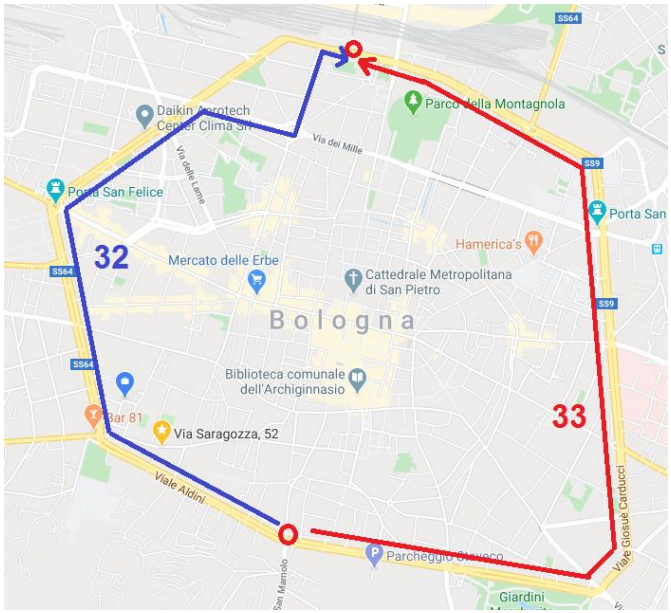
- Bologna, linea 33 (circolare sinistra)
  - il percorso da Porta Saragozza a via Petrarca è svolto nel senso diretto e dura solo 3 minuti
  - il percorso inverso, da via Petrarca a Porta Saragozza, viene coperto dalla stessa linea in ben 35 minuti, facendo tutto il giro passando dalla stazione

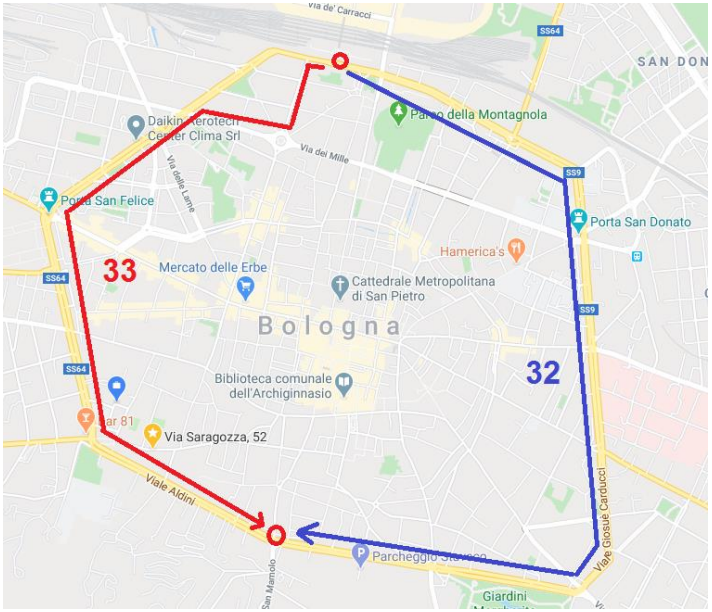
NOTA: non esiste un analogo percorso con la linea 32 (circolare destra), in quanto essa *non ha alcuna fermata di nome "Petrarca".*



Esempio 1 (caso b)

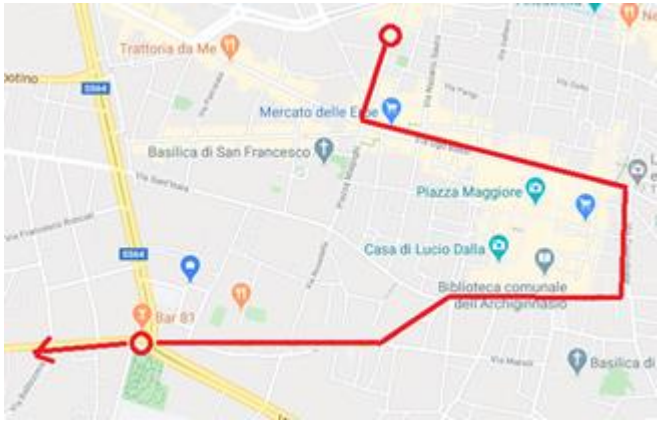
- Bologna, linee 33 (circolare sinistra) & 32 (circolare destra)
  - il percorso da Porta San Mamolo alla Stazione è fattibile invece *sia con la linea 33 sia con la linea 32*, con *tempi di percorrenza diversi*
  - chiaramente, nel percorso inverso da Stazione a Porta San Mamolo i percorsi e i tempi delle due linee si scambiano.

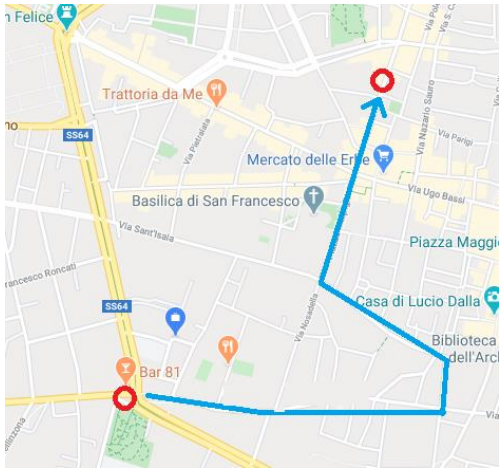




Esempio 2

- Bologna, linea 20 … ?
- Sono ***due linee***, da punto a punto!
  - in un senso, il «20 direzione Casalecchio»
  - nell'altro, il «20 direzione Pilastro»
- Sono due linee diverse, con *diverso instradamento e fermate distinte!*
- Ogni linea copre *una singola direzione*
  - da via Marconi a P.ta Saragozza si va <u>solo</u> con il «20 dir. Casalecchio», non con l'altra linea
  - analogamente il percorso opposto non è possibile in *questa* linea, ma solo *nell'altra* e oltretutto *con diverso instradamento*



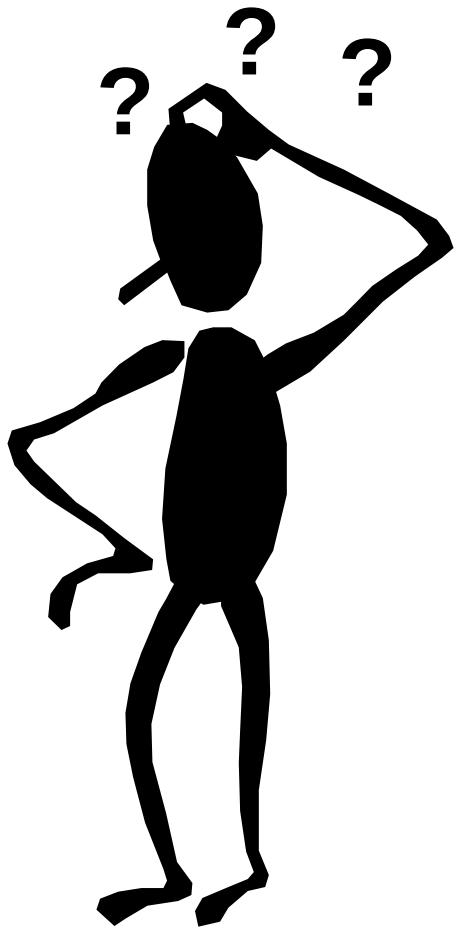


Analisi del dominio

Linea

Fermata

Percorso



Riusciamo a trovare altri dettagli nelle situazioni delle città analizzate?

Analisi del dominio: Linea

- Ogni ***Linea*** del trasporto pubblico, operata con i mezzi più diversi (bus, tram, metro, traghetti..),
  - è identificata da un *codice identificativo* (alfanumerico)
  - opera *in un'unica direzione,* dal *capolinea iniziale* al *capolinea finale*
    - l'eventuale linea operante nel senso inverso avrà un codice identificativo differente
- Le linee si distinguono in:
  - linee **da punto a punto** (PaP), che vanno dal capolinea iniziale a un capolinea finale *diverso dal primo*
  - linee **circolari**, che partono dal capolinea iniziale e, dopo un giro più o meno lungo, vi ritornano: per queste linee, a differenza delle precedenti, i *capilinea iniziale e finale coincidono sempre.*

Analisi del dominio: Fermata

- A ogni linea sono associate le corrispondenti ***Fermate***
  - ognuna caratterizzata dal relativo *orario di passaggio* della corsa
  - tale orario è espresso come *differenza in minuti rispetto alla partenza dal capolinea iniziale,* che ha orario di passaggio 0
- Semplificazioni
  - i tempi di percorrenza si assumono uguali a tutte le ore e in tutti i giorni della settimana (ossia, non si considera la distinzione fra *ore di punta* e *ore di morbida*, né fra i diversi giorni della settimana)

Analisi del dominio: Percorso

- Un ***Percorso*** dalla fermata X alla fermata Y è un *tragitto senza cambi* che collega X a Y
  - per le linee da punto a punto (PaP), ciò significa semplicemente un tragitto da X verso Y (con X ≠ Y)
  - per le linee circolari, il tragitto può invece anche "scavallare" il capolinea, quindi Y può anche *precedere* X: in tal caso, il percorso si intende da X fino al capolinea e poi, proseguendo, da lì fino a Y.
- La durata del percorso è definita rispettivamente come:
  - per le linee da punto a punto (PaP), la differenza (>0) tra gli orari di passaggio alle due fermate Y e X
  - per le linee circolari, a seconda della posizione relativa delle due fermate rispetto al capolinea: se la differenza Y-X è negativa, si intende la somma dei due sotto-tragitti da X al capolinea e dal capolinea a Y

Analisi del problema

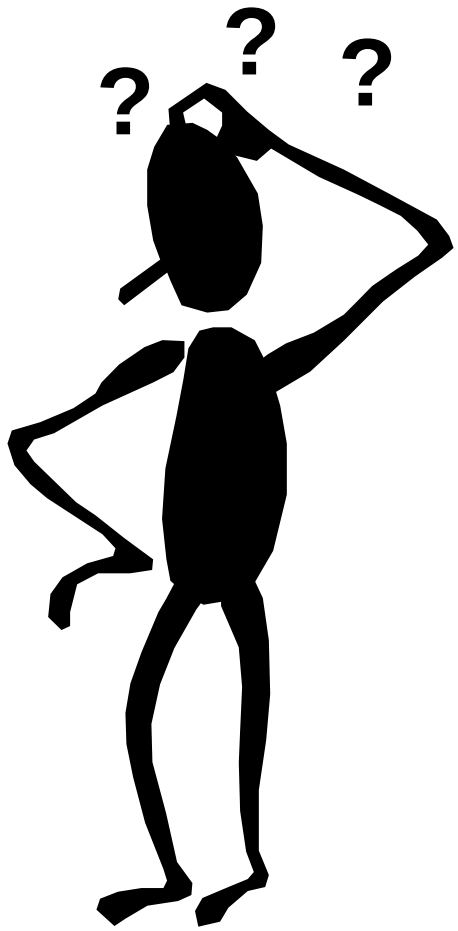
Linea

Fermata

Percorso

E quindi come le devo modellare??

Chi fa poi la ricerca del percorso?



Fermata (1/5)

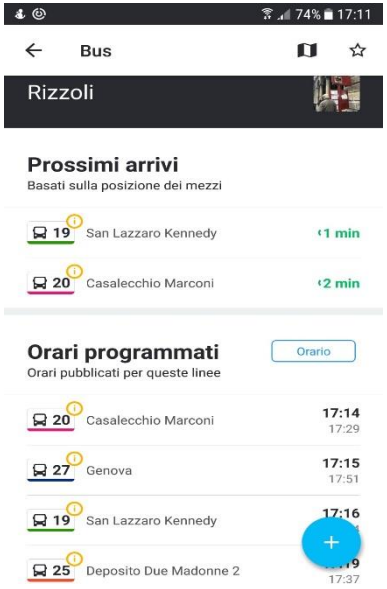
A ogni linea sono associate le corrispondenti *Fermate,* ognuna caratterizzata dal relativo *orario di passaggio* della corsa

Tale orario è espresso come *differenza in minuti rispetto alla partenza dal capolinea iniziale,* che ha orario di passaggio 0

- Dall'analisi del dominio si evince che dobbiamo modellare il concetto di Fermata
  - se operassimo «senza riflettere bene», diremmo che essa sia caratterizzata da *un identificatore* e *un orario* che specifica quando passa la corsa del mezzo di trasporto
  - MA nel mondo reale non è così!
    OCCHIO: evidentemente, stiamo mischiando informazioni che appartengono a concetti differenti….!!

Fermata (2/5)

- Ragioniamo un istante
  - ad una fermata del bus passano più linee
  - in una stazione ferroviaria (fermata) passano più treni
  - e così via per tutti gli altri mezzi di trasporto



Fermata (3/5)

- Mettere l'orario di passaggio *di una specifica linea* all'interno della Fermata non è *una grande idea!*
  L'orario di passaggio *non* è una caratteristica della Fermata, ma della *Linea!*
- Ogni Linea «passa» in una certa Fermata ad un dato orario
  - il 25 passa in Rizzoli alle 12:30
  - il 20 passa in Rizzoli alle 12:35
  - il 14 passa in Rizzoli alle 12:33
- Sarà quindi la Linea a doversi occupare dell' orario di passaggio alle singole fermate, non la Fermata.

Fermata (4/5)

- Tra l'altro, se per assurdo mettessimo l'orario di passaggio dentro a Fermata incapperemmo anche in un altro problema: *avremmo un numero «spropositato» di Fermate!*
- Infatti, ogni fermata andrebbe «replicata» per ogni linea
  - oneroso in termini di spazio occupato ma anche di gestione: ad esempio, come confronteremmo due Fermate??!?
- Soprattutto, non sarebbe vero nella realtà!
  - la fermata «Rizzoli» della linea 25 è *diversa* da quella della linea 20 e della linea 14?
- Per non parlare dei cambi…
  - in questa app *non* gestiamo percorsi con cambi, ma se lo facessimo sarebbe *molto* complicato operare con questa visione distorta!

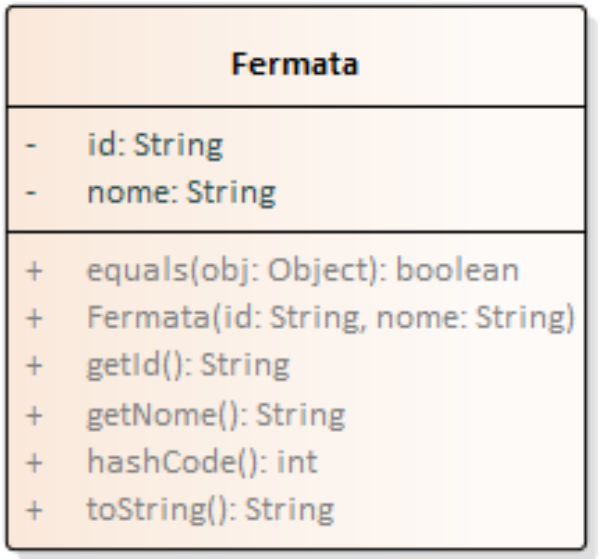
Fermata (4/5)

- Tra l'altro, se per assurdo mettessimo l'orario di passaggio dentro a Fermata incapperemmo anche in un altro problema: *avremmo un numero «spropositato» di Fermate!*
- Infatti, ogni fermata andrebbe «replicata» per ogni linea
  - oneroso in termini di spazio occupato e di gestione: ad esempio, come confro
- Soprattutto, non sare
  - la fermata «Riz la linea 20 e della linea 14?
- Per non parlare dei cam…
  - in questa app *non* gestiamo percorsi con cambi, ma se lo facessimo sarebbe *molto* complicato operare con questa visione distorta!

Progetto sbagliato
= delirio in ogni
cosa da fare!

Fermata (5/5)

- Progetto corretto: ***Fermata*** rappresenta una singola fermata, caratterizzata da
  - identificativo univoco
  - nome
  - metodi accessor
  - equals & hashcode
    - hashcode può essere generata automatica-mente da Eclipse
    - è necessaria per operare correttamente con HashMap (andate a curiosare sui JavaDocs)
- Già pronta nello Startkit



Linea (1/10)

Ogni ***Linea*** del trasporto pubblico è identificata univocamente da un *codice identificativo* (alfanumerico) *e* si intende operativa *in un'unica direzione,* dal *capolinea iniziale* al *capolinea finale:* l'eventuale linea operante nel senso inverso avrà un codice identificativo differente

- Prima cosa: su quali dati opera Linea?
- A grana grossa possiamo dire che una Linea è «*fatta di*»:
  - un insieme di Fermata con relativo orario di passaggio
  - un capolinea iniziale
  - un capolinea finale

Concentriamoci su queste!

Ai capolinea penseremo poi

Linea (2/10)

- Linea è «*fatta di*» fermate con relativo orario di passaggio
- Due problemi da risolvere:
  1. Cosa si intende per «orario di passaggio»? Come rappresentarlo?
  2. Associare ad ogni Fermata il suo orario di passaggio
- Per (1), nell'analisi del dominio si era stabilito che «l'orario è espresso come *differenza in minuti rispetto alla partenza dal capolinea iniziale*, che ha orario di passaggio 0»

  → l'orario di passaggio può essere modellato come un intero: non occorre LocalTime (né altri concetti legati a date/orari)

Linea (3/10)

- Per (2), esistono diverse possibilità:
  - usare due array avendo cura di mettere la Fermata (in un array) e il suo orario di passaggio (nell'altro array) nella medesima posizione

  - definire una nuova classe «fermata con orario» che rappresenti la coppia (ricorda tanto partiti e voti…)

  - sfruttare una delle strutture dati della JCF

Linea (4/10)

- La coppia Fermata/orario ha le caratteristiche di una tabella
- Sembra sensato adottare una Map<Fermata, Integer>

Capolinea iniziale	0
Fermata1	4
Fermata2	10
Fermata3	15
Fermata4	20
….	…
Capolinea finale	40

Linea (5/10)

- MA: come rappresentare i capilinea iniziale e finale?

- Analizziamo la realtà
- Un capolinea è una Fermata con
  - tempo di passaggio 0, se iniziale
  - tempo di passaggio N, se finale (dove N=durata di tutta la Linea)
- Dunque, un Capolinea è una Fermata con un *particolare tempo di passaggio*

Fermata0	0
Fermata1	4
Fermata2	10
Fermata3	15
Fermata4	20
….	…
FermataN	40

Linea (6/10)

- «Un Capolinea è una Fermata con un *particolare tempo di passaggio*»
- Questione di fondo: sono attributi «fissi»?
  - sì, se la Linea fosse immutabile nel tempo
  - sì, se non si introducessero mai nuove fermate allungando/accorciando i percorsi
  - MA nella realtà non è affatto così!
- In realtà, i percorsi delle Linee non sono immutabili nel tempo
  - possono essere aggiunte nuove fermate (e magari il capolinea non è più tale, o si sposta..)

Linea (7/10)

- Ci troviamo in uno di quei casi di «*attributi calcolati*»
  - capolinea iniziale e finali devono essere certamente *recuperabili* tramite opportuni metodi accessor..
  - …MA non «fisicamente» memorizzati in Linea, perché potrebbero facilmente variare (e in tal caso doverli cambiare sarebbe faticoso)
  - meglio quindi *ricalcolarli* ogni volta
- Ah, ok! Tutto a posto quindi..?

Linea (8/10)

- Scegliendo di *non memorizzare fisicamente* i due capilinea, però, una mappa Map<Fermata, Integer> non è molto pratica da usare
  - recuperare i capilinea sarà una azione frequente
  - MA con una mappa fatta così, ogni volta sarebbe un delirio
  - infatti, il solo modo per identificare un capolinea è legato ai tempi di passaggio → dovremmo ogni volta cercare in tutta la mappa!

Fermata0	0
Fermata1	4
Fermata2	10
Fermata3	15
Fermata4	20
….	…
FermataN	40

Linea (9/10)

- IDEA: invertiamo le colonne!
- Da Map<Fermata, Integer> a Map<Integer, Fermata>
- Così la ricerca dei capilinea diventa facile e veloce

0	Fermata0
4	Fermata1
10	Fermata2
15	Fermata3
20	Fermata4
..	….
40	FermataN

Linea (10/10)

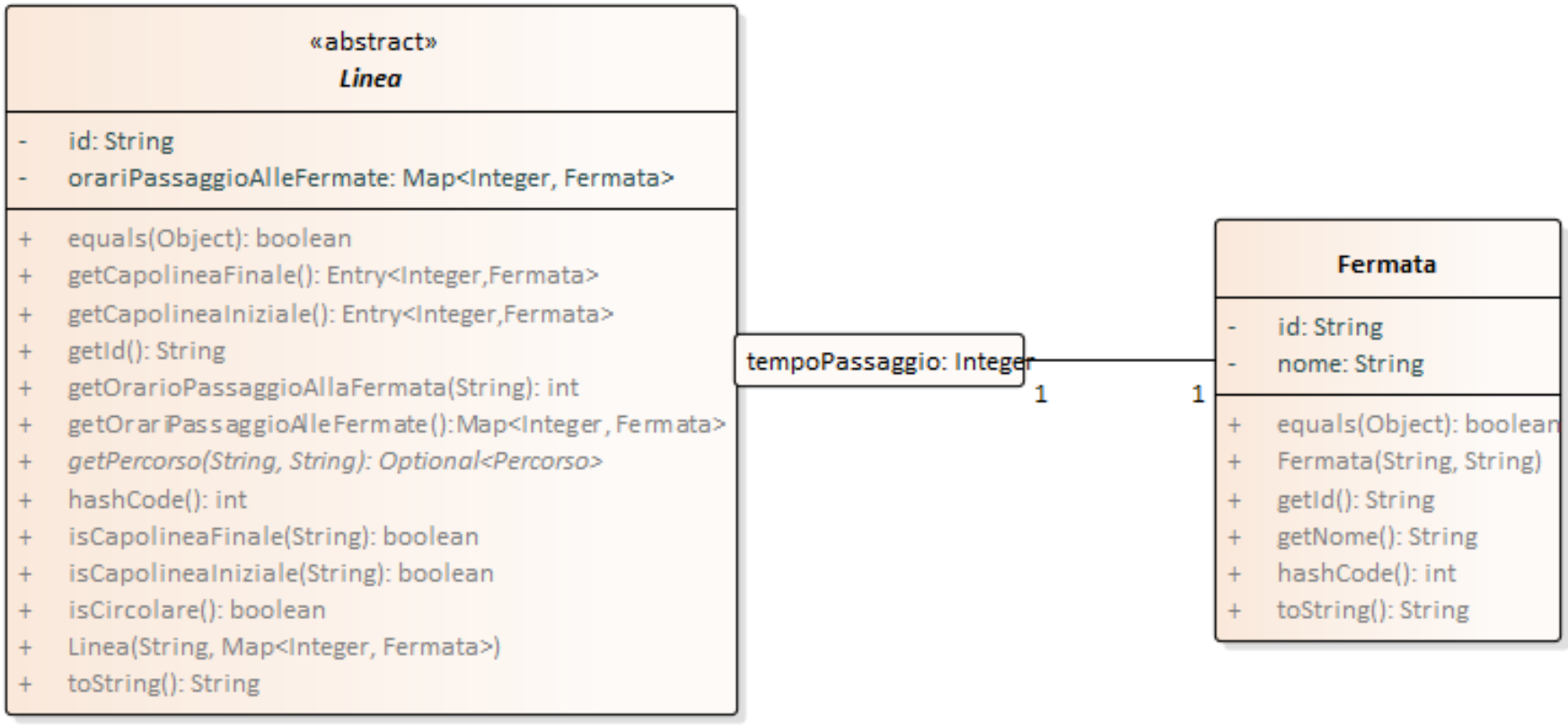
- Torniamo all'analisi del dominio

Le linee si distinguono in:

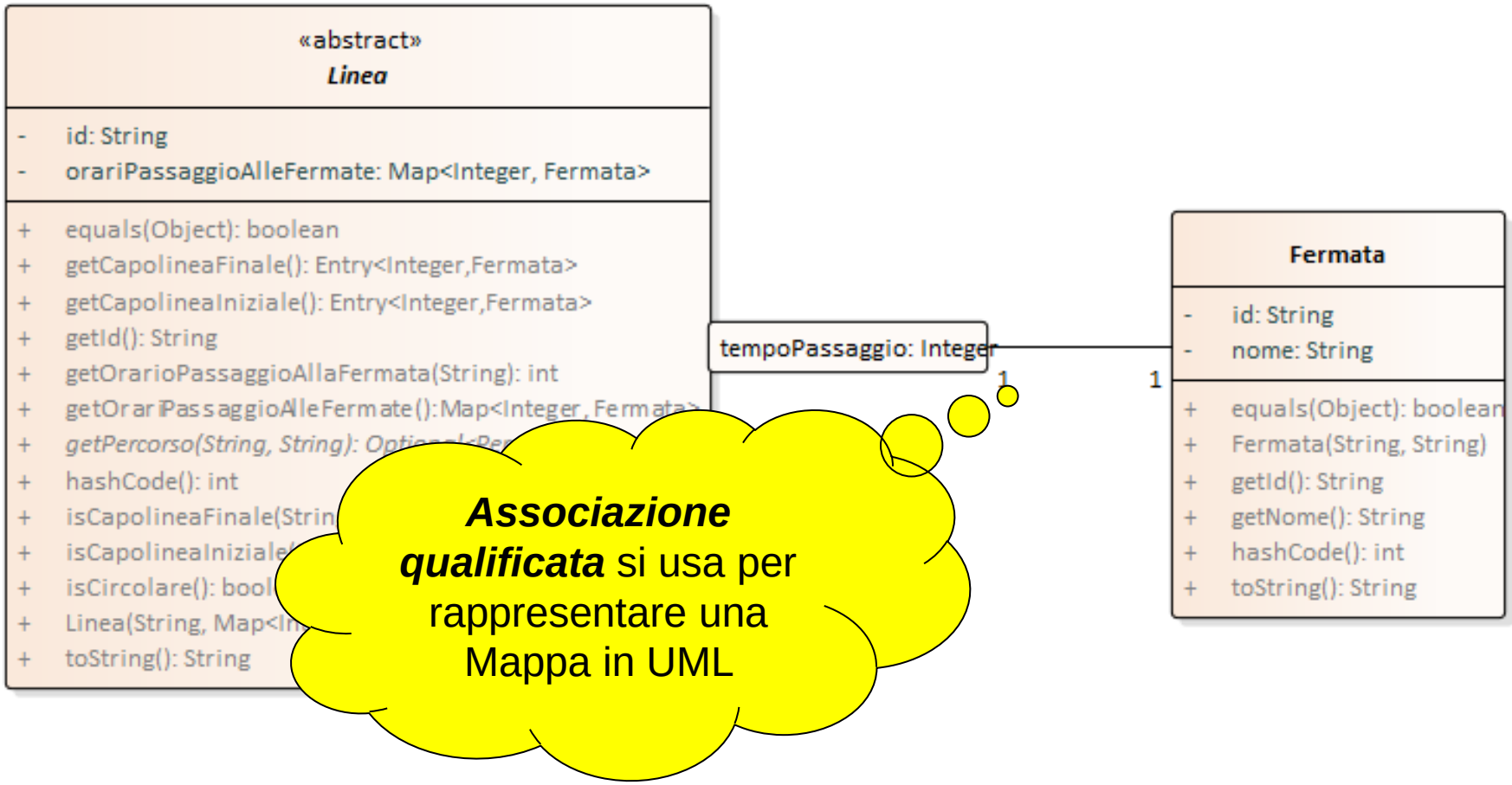
- linee **da punto a punto** (PaP), che vanno dal capolinea iniziale a un capolinea finale *diverso dal primo*
- linee **circolari**, che partono dal capolinea iniziale e, dopo un giro più o meno lungo, vi ritornano: per queste linee, i *capilinea iniziale e finale coincidono sempre.*

- Ci sono quindi due tipi di linee *alternativi fra loro*
- Dubbio: ha senso istanziare la Linea «qualsiasi»?
  - una Linea «qualsiasi» sarebbe in grado di rispondere a domande relative al Percorso tra due fermate?
  - chiaramente, NO → è una nozione (e quindi una classe) ***astratta***

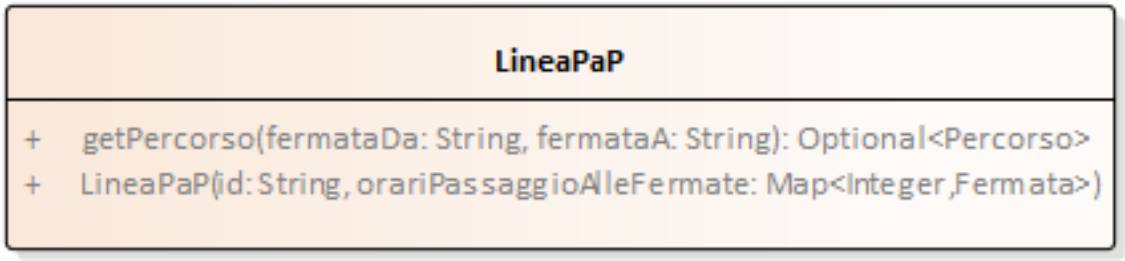
Linea: il modello



Linea: il modello

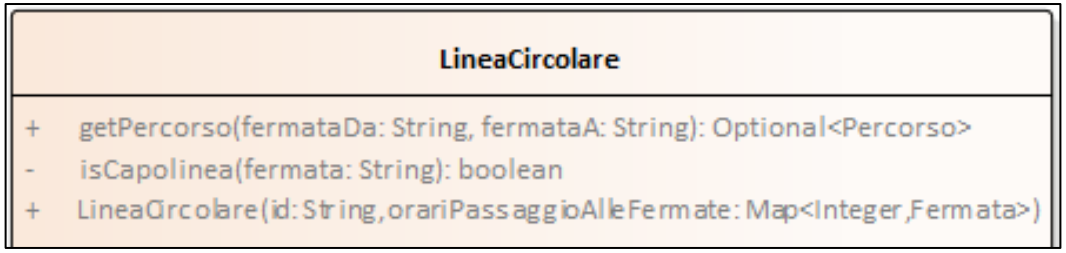


Linea da punto a punto



- La classe **`LineaPaP`** rappresenta una linea da punto a punto
  - il costruttore delega la costruzione alla classe base, ma *verifica anche che la linea non sia circolare* – altrimenti, violazione di precondizione
  - il metodo **`getPercorso`** è qui concretizzato nel caso semplice di linea punto a punto *monodirezionale*:
    - se una fermata non appartiene alla linea o esse non sono in ordine (ossia, se la seconda precede la prima), il percorso non esiste e viene restituito un Optional vuoto

Linea circolare

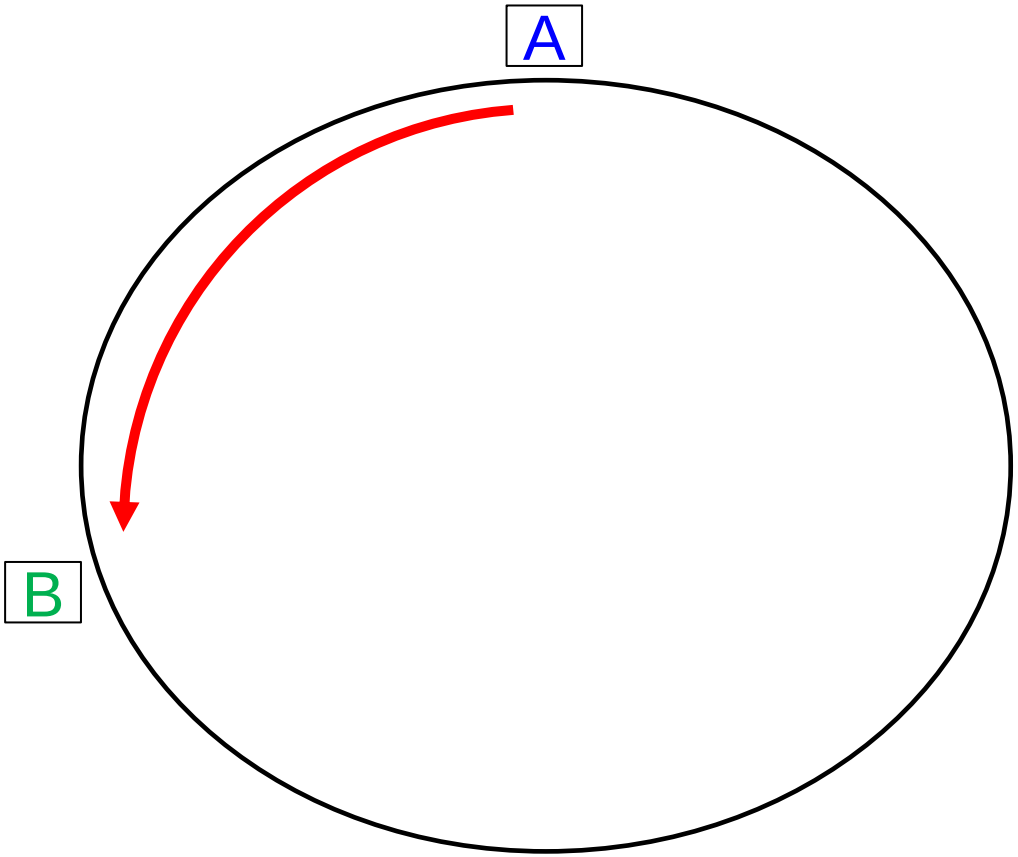


- La classe **`LineaCircolare`** rappresenta una linea circolare
  - il costruttore è analogo a quello del caso precedente, *salvo ovviamente la verifica di circolarità* che dev'essere invertita
  - il metodo getPercorso deve invece considerare anche i percorsi circolari via capolinea, distinguendo quindi due situazioni-base:
    - da fermataDa a fermataA, con fermataDa < fermataA
      → caso standard come PaP
    - da fermataDa a fermataA, con fermataDa >= fermataA
      → da interpretare via capolinea

Linea circolare

- In realtà, le situazioni da distinguere sono più di due
  - alle situazioni base, infatti, occorre aggiungere il *trattamento dei casi limite* che coinvolgono uno o più capilinea
- Distinguiamo perciò cinque situazioni (le precedenti + altre tre):
  - da capolinea a fermataA (diversa dal capolinea)
  - da fermataDa (diversa dal capolinea) al capolinea
  - da capolinea a capolinea (giro completo)
  - da fermataDa a fermataA, con fermataDa < fermataA
    → caso standard come PaP
  - da fermataDa a fermataA, con fermataDa >= fermataA
    → da interpretare via capolinea
  - NB: come per la Linea PaP, se una delle due fermate non appartiene alla linea viene restituito un *Optional* vuoto

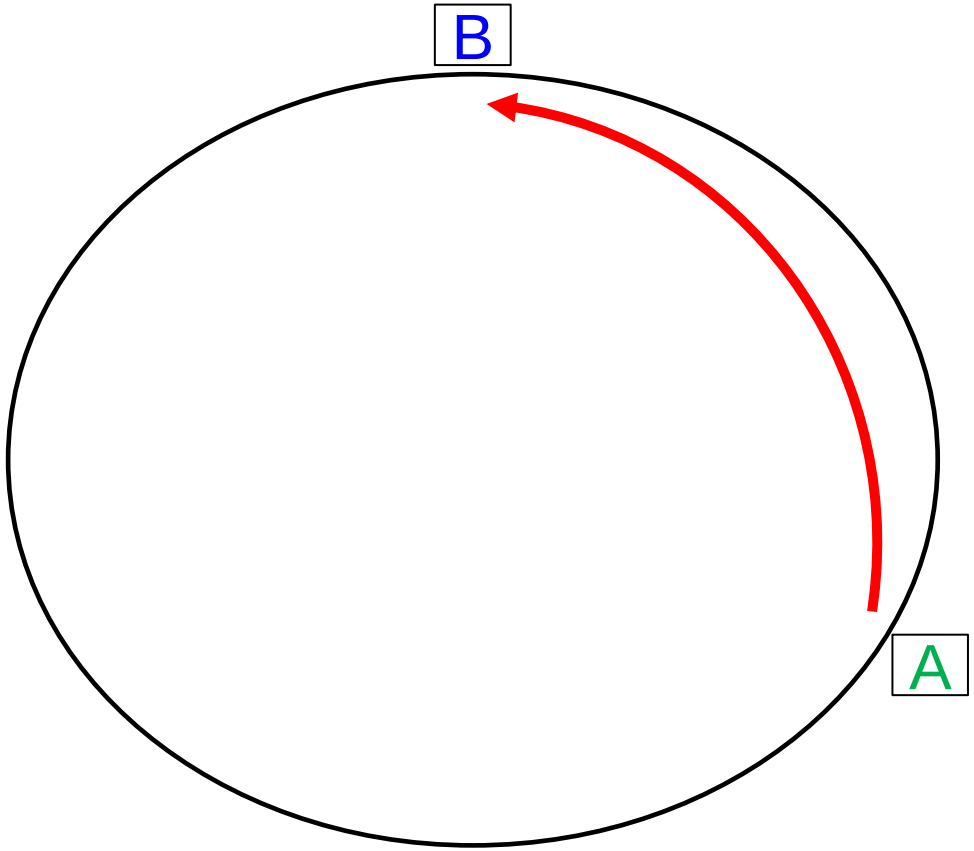
Linea circolare: getPercorso (1)



CASO 1:
da A (capolinea iniziale)
a B

Durata percorso =
orario di passaggio in B

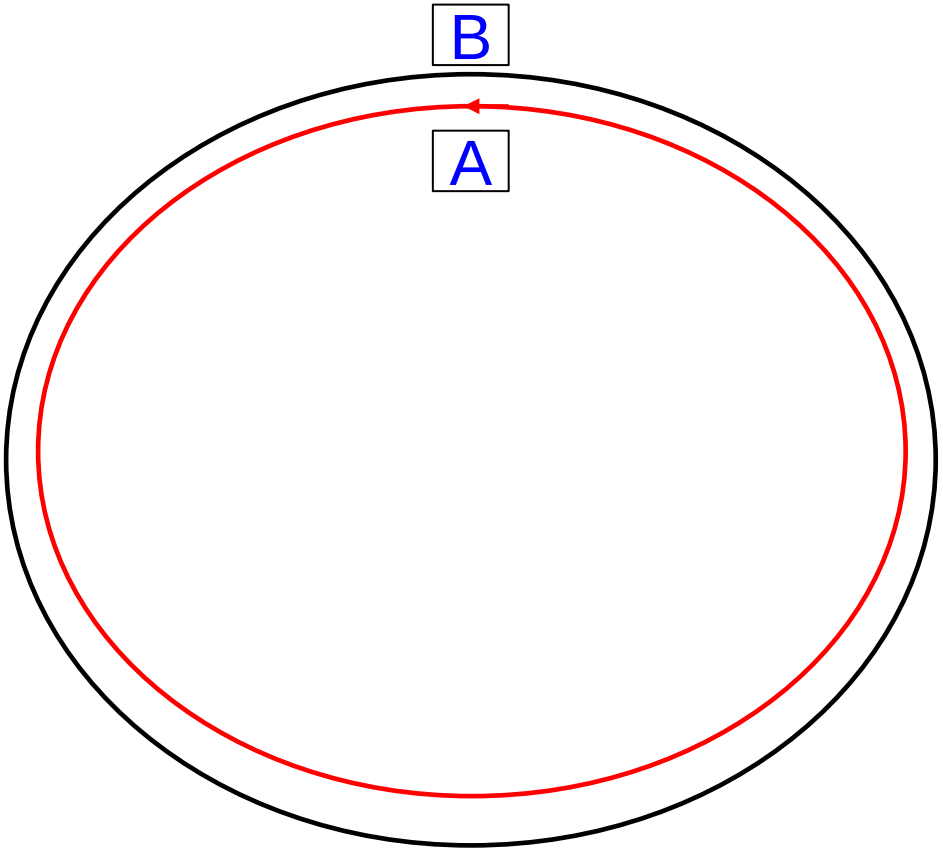
Linea circolare: getPercorso (2)



CASO 2:
da A
a B (capolinea finale)

Durata percorso = orario di passaggio al capolinea finale (B) – orario di passaggio in A

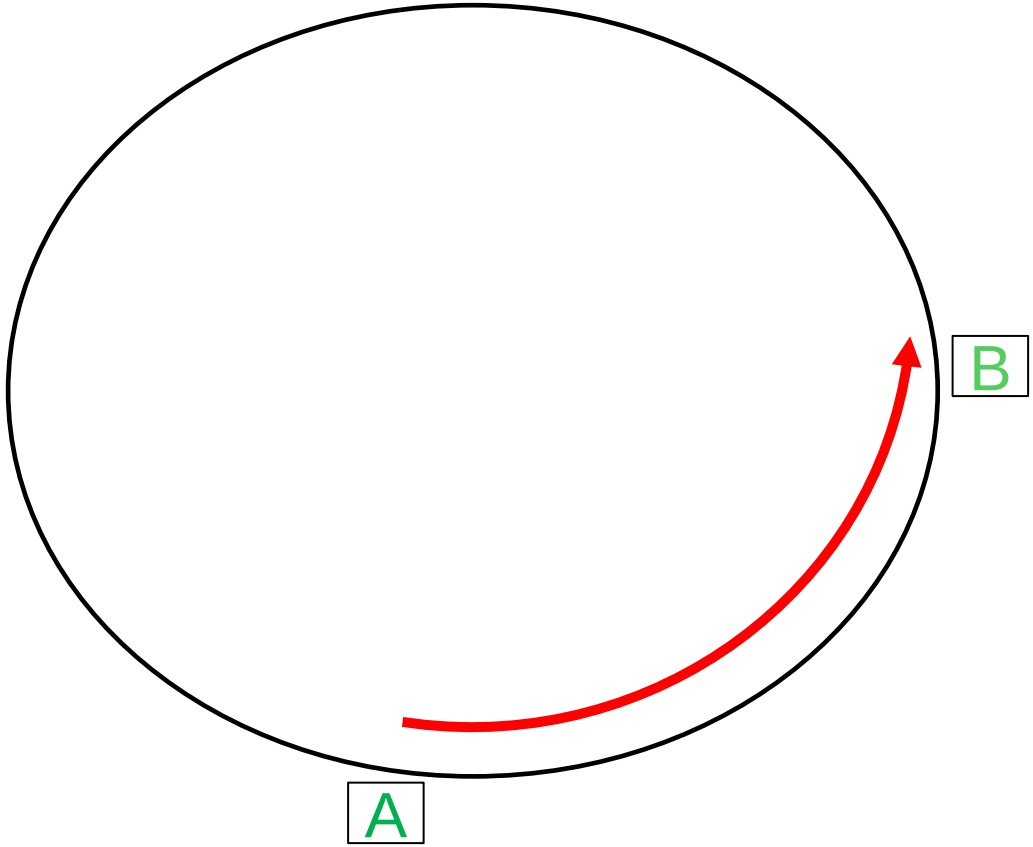
Linea circolare: getPercorso (3)



CASO 3:
da capolinea a capolinea

Durata percorso =
orario di passaggio al
capolinea finale

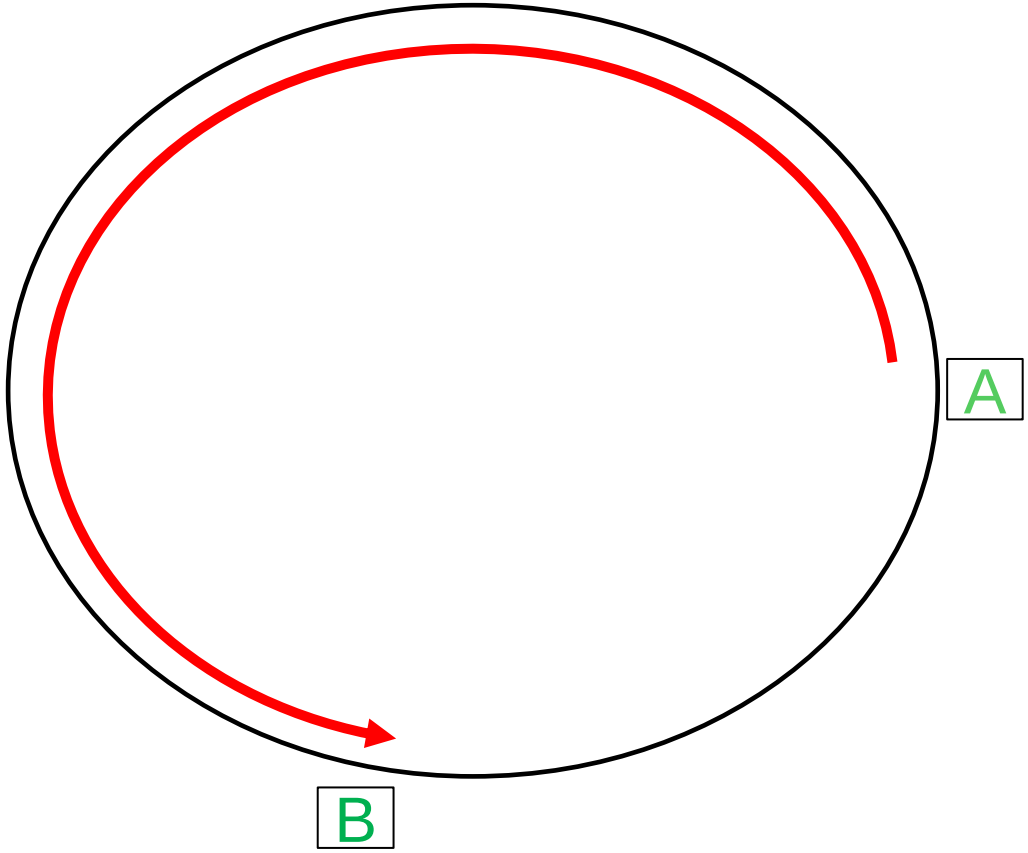
Linea circolare: getPercorso (4)



CASO 4:
da A a B (A precede B)

Durata percorso =
come nel caso Linea PaP

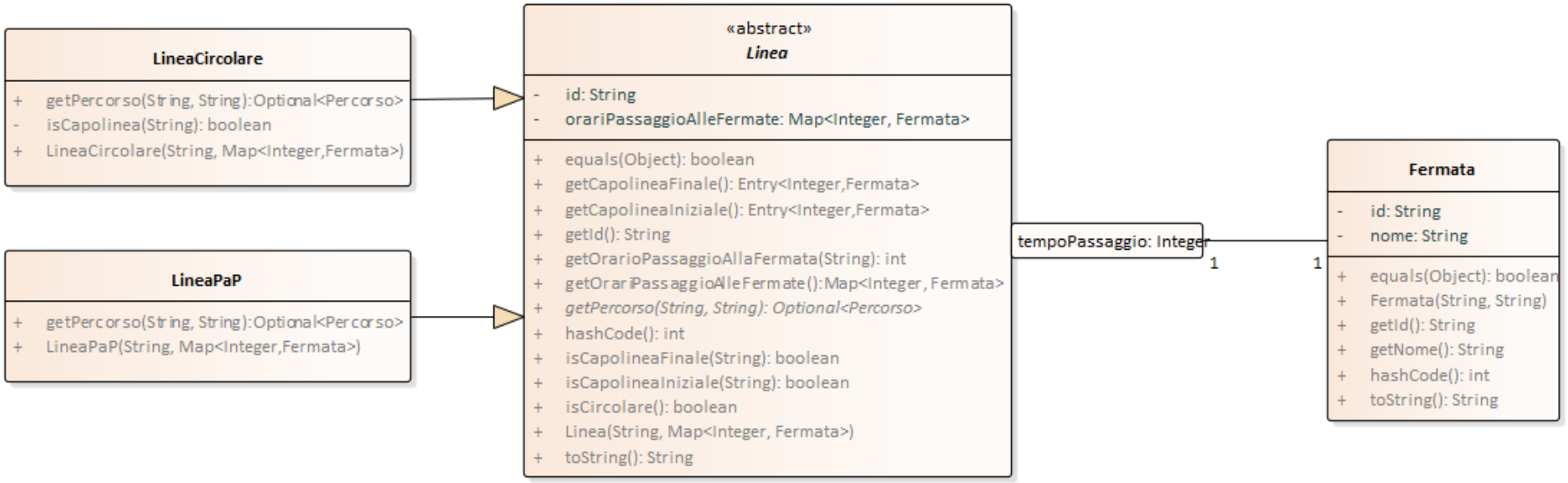
Linea circolare: getPercorso (5)



CASO 5:
da A a B (B precede A)

Durata percorso =
durata giro completo –
durata percorso da B ad A

Il modello (quasi) completo



Percorso (1/2)

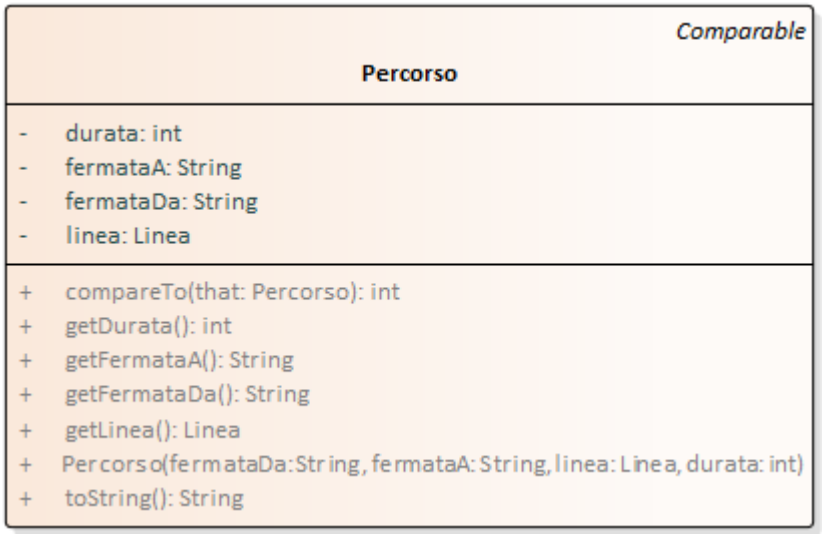
Un ***Percorso*** dalla fermata X alla fermata Y è un *tragitto senza cambi* che collega X a Y con un certa durata

- Dall'analisi del dominio si evince che dobbiamo modellare il concetto di Percorso
  - se operassimo «senza riflettere bene», diremmo che esso sia caratterizzata da *due Fermata* che rappresentano la fermata di partenza e quella di arrivo e *una durata* che specifica il tempo di percorrenza tra le due Fermata
  - MA stiamo perdendo un pezzo di informazione..
    OCCHIO: possono esistere Linee differenti che «toccano» le due fermate considerate con tempi di percorrenza diversi..

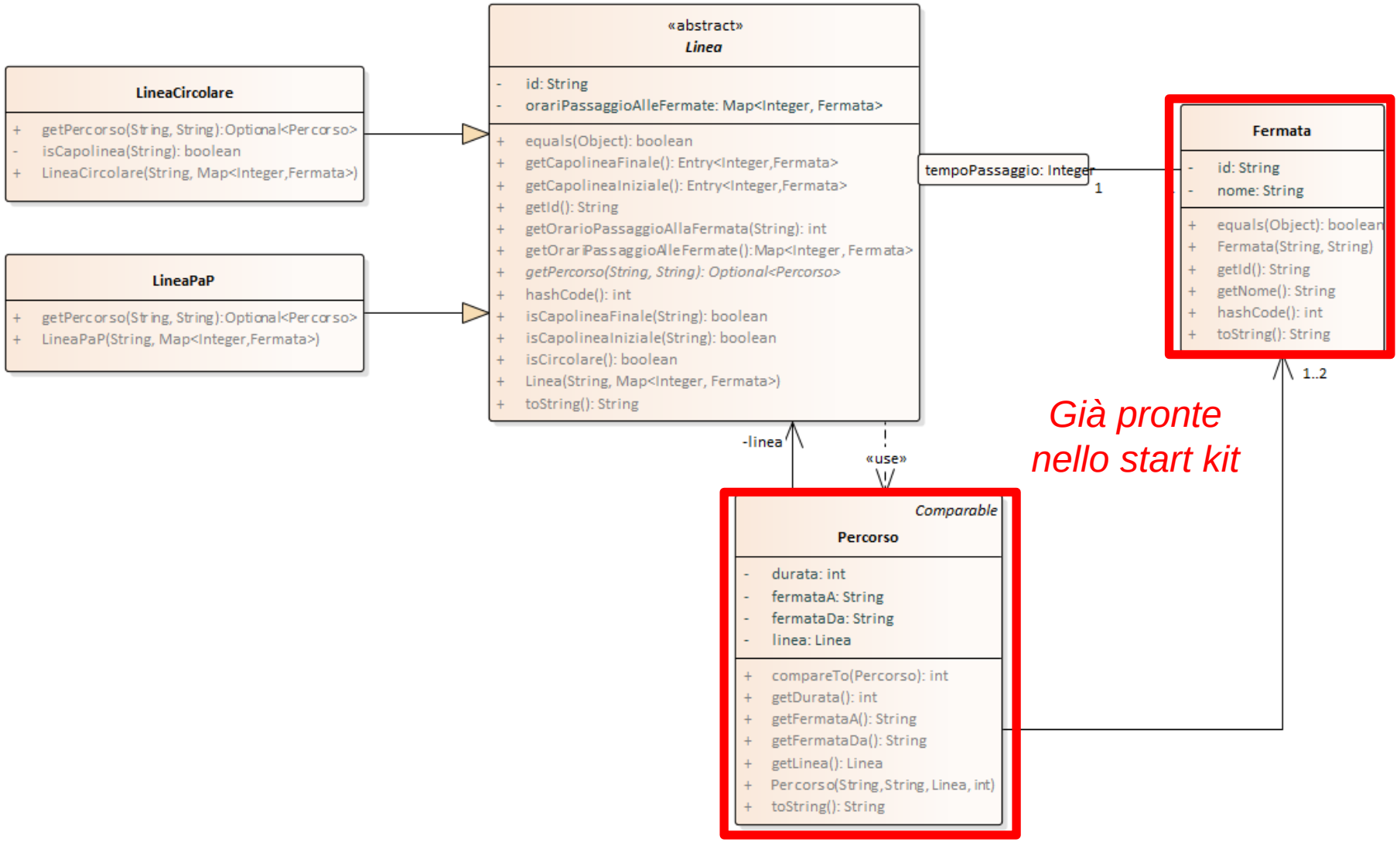
Percorso (2/2)

- Progetto corretto: *Percorso* rappresenta un tragitto tra due Fermata su una certa Linea che ha una certa durata
- Fornisce gli opportuni accessor ed è *Comparable*

- Già pronta nello Startkit



Il modello completo



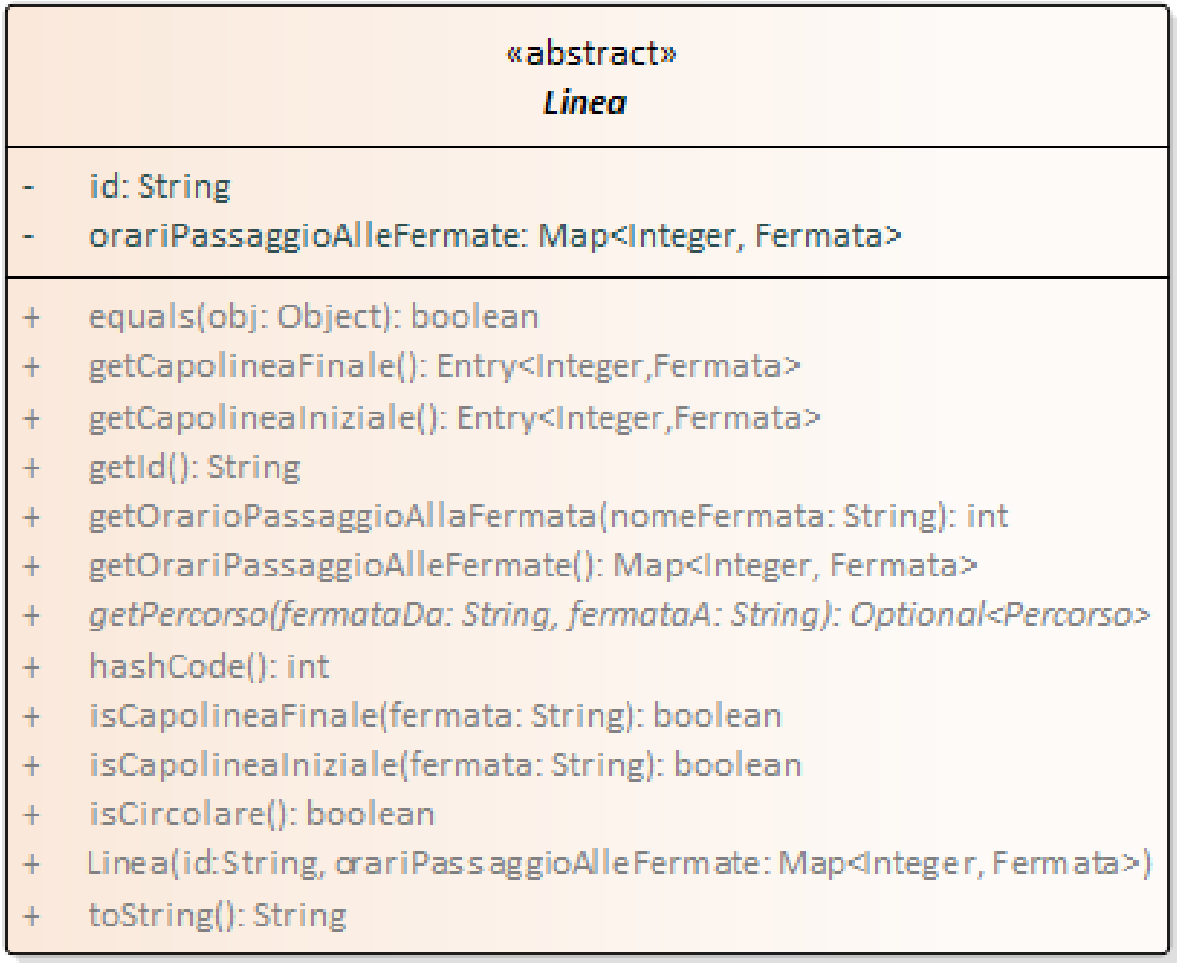
Verso l'implementazione
Verifica delle precondizioni

- Molti metodi ricevono argomenti che potrebbero essere sbagliati → *dobbiamo controllarli* prima di usarli
- MA POI, se ci accorgiamo che un argomento è errato, che si fa?
  - è una violazione di precondizioni: il cliente ha violato il contratto!
  - non si può proseguire se manca un dato essenziale!
- Occorre *lanciare un allarme* e *interrompere il normale flusso* del programma
- Il *tipo di allarme* lanciato deve *rappresentare l'accaduto*
- Al momento non sappiamo ancora come di lancia un allarme (lo scoprirete presto ☺) ci facciamo aiutare → **`ErrorMessage`**

Verso l'implementazione
Verifica delle precondizioni

- La classe `ErrorMessage` mette a disposizione un metodo statico `emit(String message)`
- Tale metodo permette di lanciare l'allarme nel caso gli argomenti passati ai metodi presentino problemi
- Nel parametro message occorre indicare quale tipo di allarme sia scattato
- Trovate `ErrorMessage` già pronta nello StartKit nel package del model

Riprendiamo Linea



Riprendiamo Linea

- Metodi di Linea
  - Costruttore: riceve l'identificativo della linea e la mappa con l'elenco delle fermate, indicizzate in base al minuto di passaggio: il capolinea iniziale ha indice 0 → **`Map<Integer, Fermata>`**
    NB: occorre controllare i dati ricevuti dal costruttore: se incoerenti, occorre *lanciare il giusto allarme* → `ErrorMessage`
  - Accessor: **`getOrariPassaggioAlleFermata`** restituisce la mappa ricevuta dal costruttore
  - Accessor: **`getOrarioPassaggioAllaFermata(String nome)`** restituisce l'intero corrispondente all'orario di passaggio *in quella fermata*
    → se tale fermata non è presente nella linea cosa fa?
    → non è un dato opzionale, è una violazione di precondizioni!
    → *lancia un apposito allarme,* `ErrorMessage`

Riprendiamo Linea

- Metodi di Linea
  - Accessor: **`getCapolineaIniziale`** e **`getCapolineaFinale`** estraggono dalla mappa rispettivamente la prima/ultima fermata
    → Anche qui, se non esistono è una violazione di precondizioni
    → `ErrorMessage`
  - Predicati: **`isCapolineaIniziale(String nomeFermata)`**, **`isCapolineaFinale(String nomeFermata)`** e **`isCircolare`**, con ovvio significato
  - Accessor astratto: **`getPercorso(String nomeFermataDa, String nomeFermataA)`** restituisce il percorso, se esiste, fra le due fermate, inteso nella direzione da nomeFermataDa a nomeFermataA; *se non esiste, restituisce un optional vuoto* perché è normale che possa non esserci, *non è una violazione di precondizioni!*

Gestione anomalie

- ESEMPIO: il metodo **`getCapolineaIniziale`** di **`Linea`**

```
public Entry<Integer,Fermata> getCapolineaIniziale()<{
   ……
   if (entryIniziale.isPresent()) {
       return entryIniziale.get();
   }
   else ErrorMessage.emit(
             "lista fermate vuota o illegale");
}
```

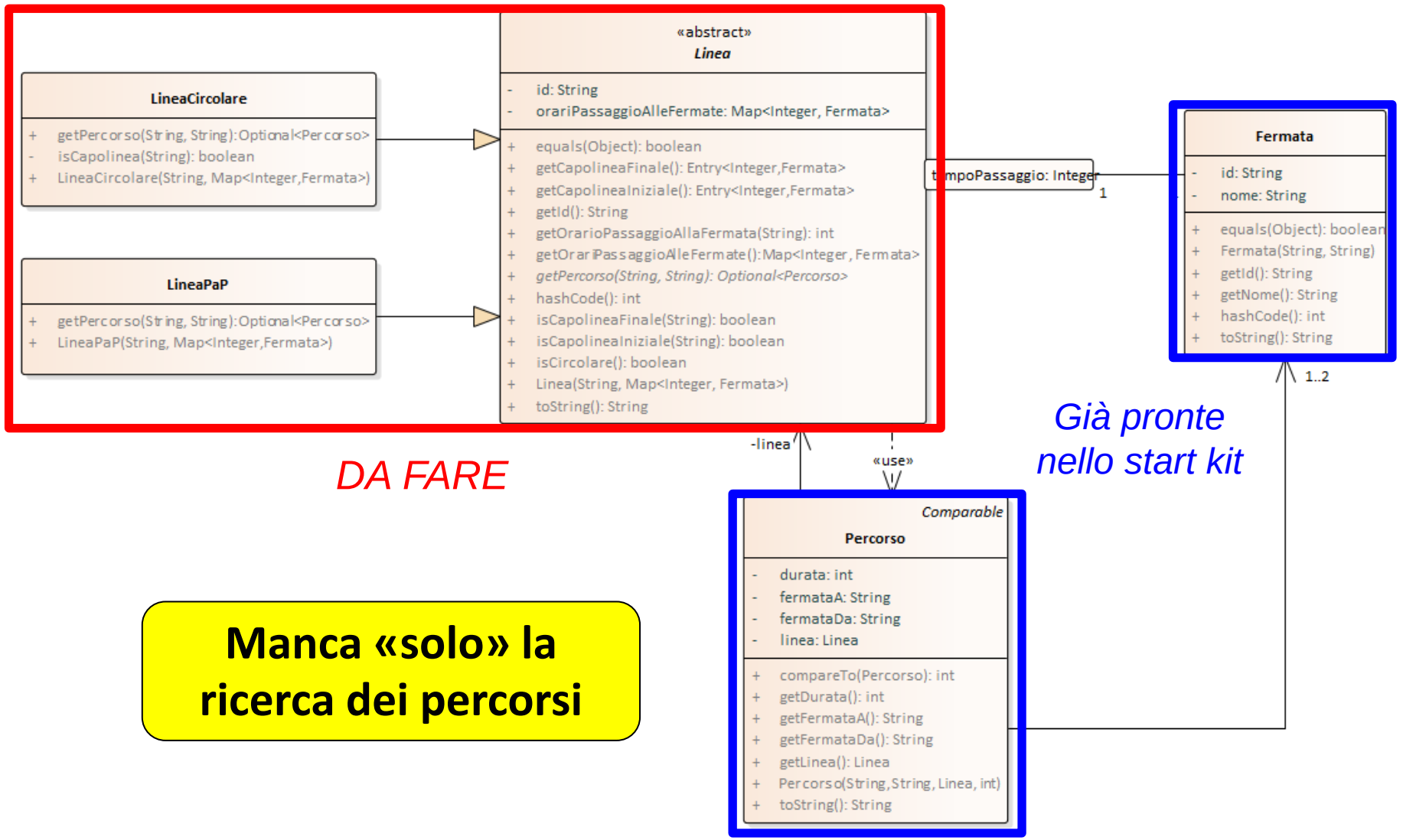
Il messaggio descrive l'accaduto

Gestione anomalie

- ESEMPIO: il metodo **`getOrarioPassaggioAllaFermata`** di **`Linea`**

```
public int getOrarioPassaggioAllaFermata(String nomeFermata){
   ……
   if (!optionalEntry.isPresent()) {
        ErrorMessage.emit(
              "Non esiste " + nomeFermata +
              " nella linea " + this.getId() );
   }
   return optionalEntry.get().getKey();
}
```

Il modello completo (fin qui)



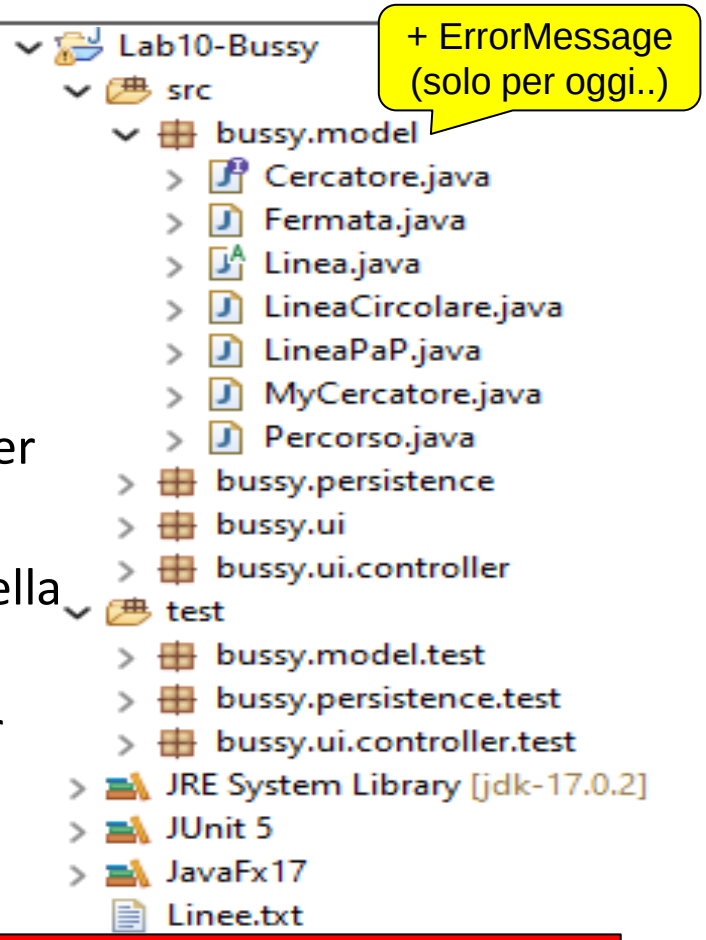
DA FARE

Già pronte nello start kit

Manca «solo» la ricerca dei percorsi

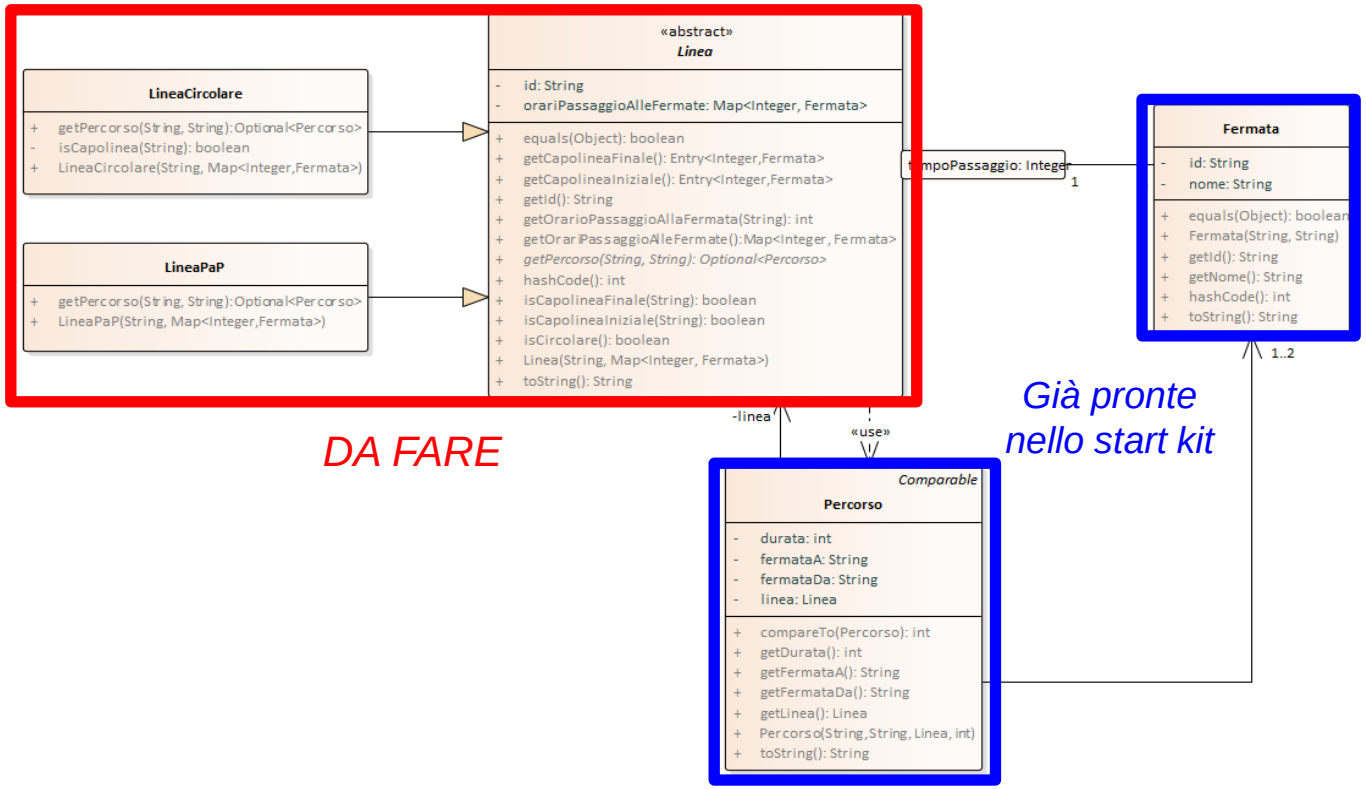
Il progetto Eclipse

- Due source folder:
  - **`src`**
    - package **`bussy.model`**
      →qui vanno inserite tutte le classi che dovrete fare
    - package **`bussy.persistence`** classi per gestione della persistenza, nello Startkit
    - package **`bussy.ui`** classi per gestione della grafica, nello Startkit
    - package **`bussy.controller`** classi per gestione del control, nello Startkit
  - **`test`** contiene i test, forniti già pronti



OCCHIO ai i nomi dei package, delle classi e dei metodi, altrimenti i test …. ☹

Il modello completo (fin qui)



Tempo a disposizione: 90 minuti

Analisi del problema

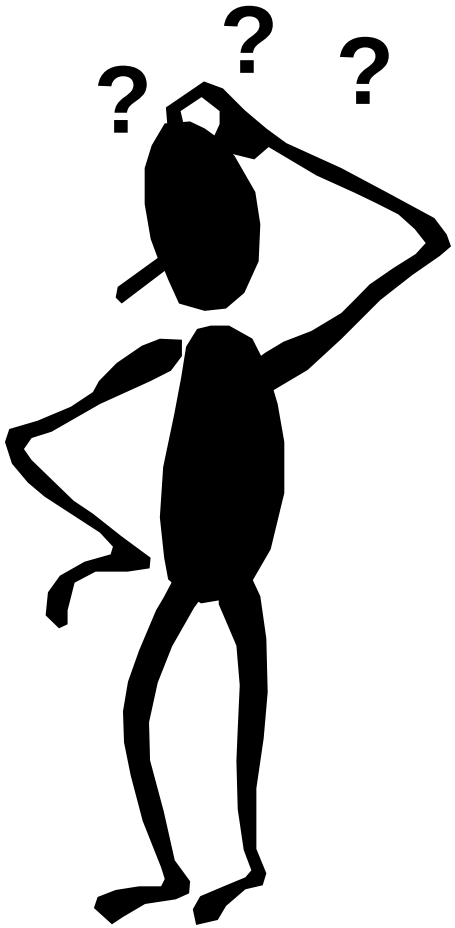
Linea

Fermata

Percorso

OK, i concetti li abbiamo modellati

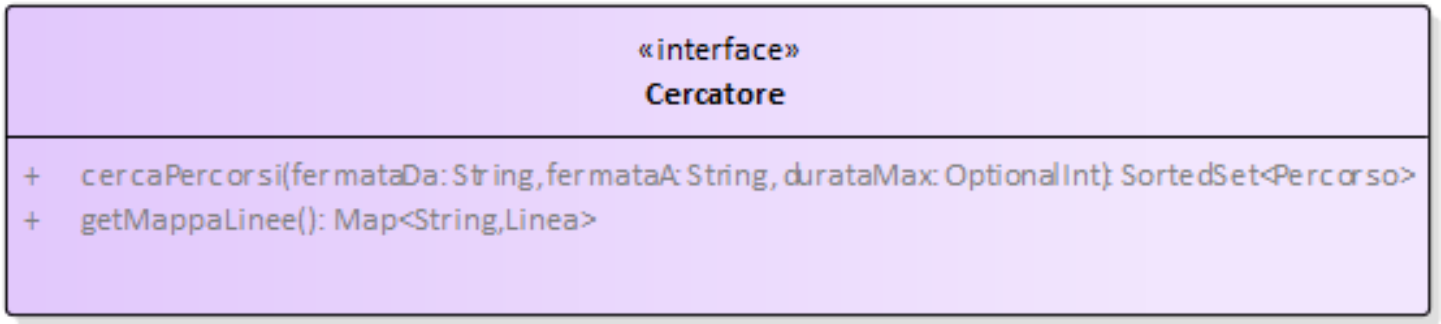
Chi fa la ricerca del percorso?



Cercatore (1/2)

- Occorre modellare una nuova entità che chiameremo «***Cercatore***» il cui compitò sarà quello di ricercare i possibili percorsi date due fermate
- Come modelliamo questa entità?
  - una classe? una classe astratta? una interfaccia?
- Considerando che:
  - la tipologia di ricerca che facciamo NON è l'unica possibile (non stiamo considerando i cambi di linea, *per ora*)
  - non c'è nulla che possa essere messo a fattor comune tra le diverse ricerche a parte la signature del metodo
- La scelta migliore è probabilmente adottare un'***interfaccia***

Cercatore (2/2)

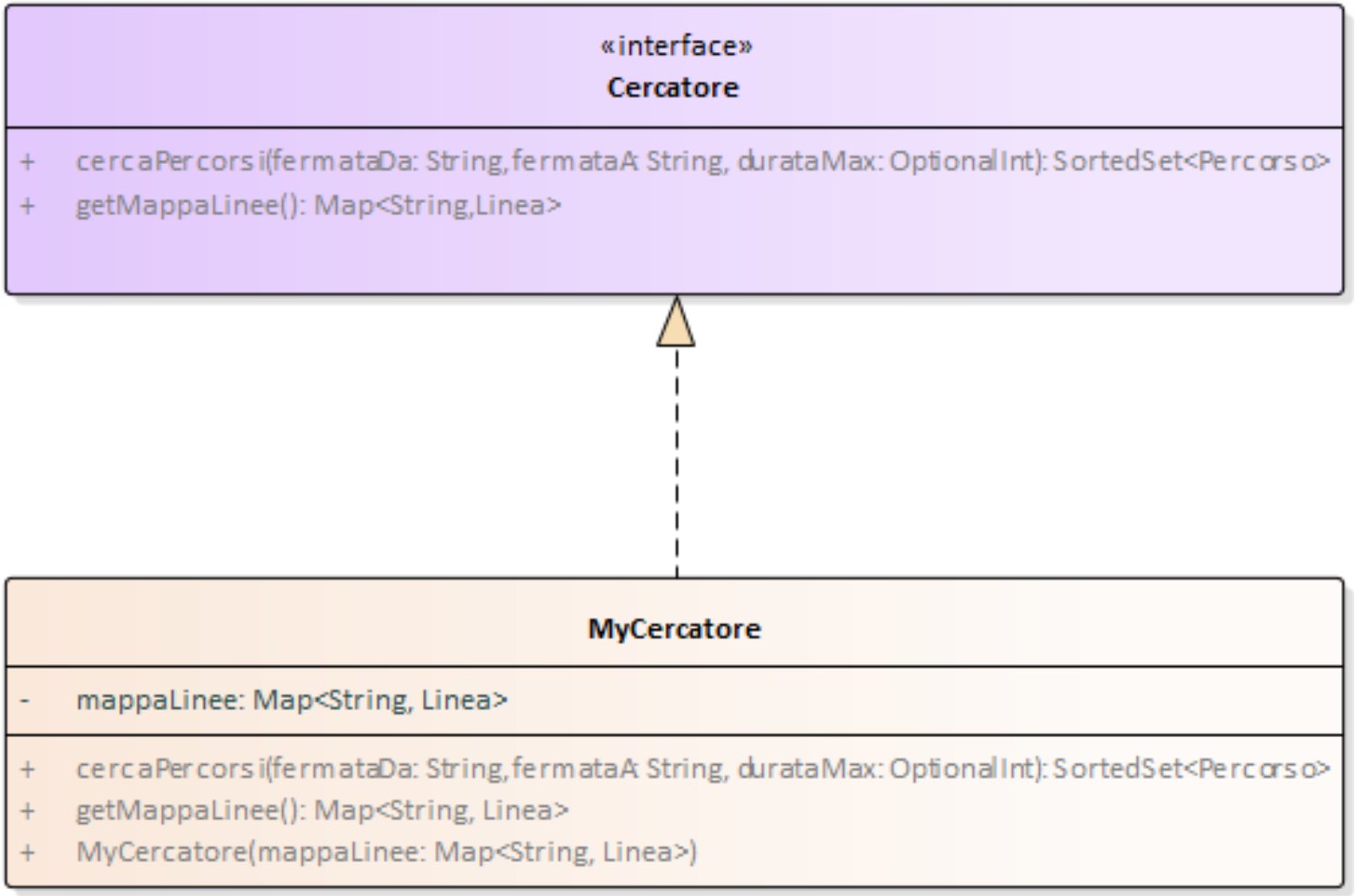


- L'interfaccia **`Cercatore`** rappresenta un'entità capace di cercare percorsi da una fermata A ad una fermata B
  - il metodo **`cercaPercorsi`** riceve come i nomi delle due fermate e un intero optional che rappresenta la durata massima accettabile del percorso: eventuali percorsi esistenti ma più lunghi saranno perciò esclusi dal risultato. Il risultato è un **`SortedSet<Percorsi>`** ordinati dal più breve al più lungo.
  - Il metodo ausiliario **`getMappaLinee`** restituisce la mappa **`Map<String,Linea>`** di tutte le linee del trasporto pubblico

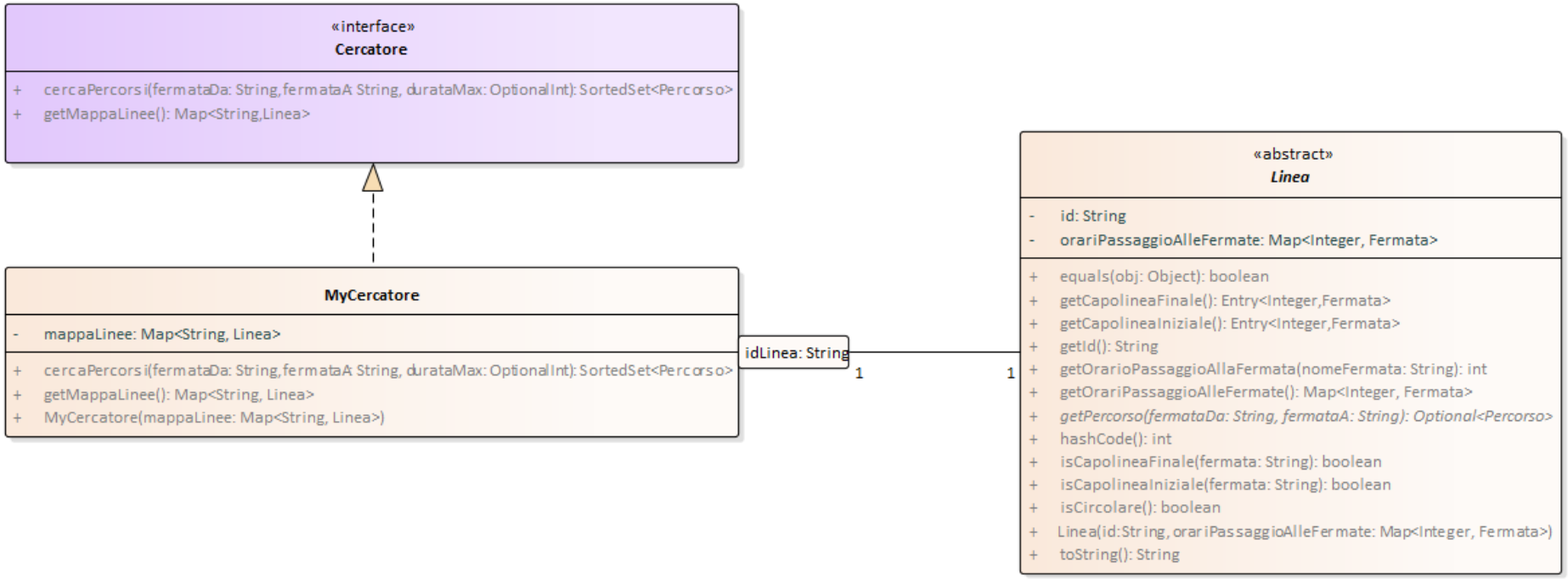
MyCercatore (1/4)

- La classe **`MyCercatore`** implementa **`Cercatore`**:
  - il costruttore riceve la mappa **`Map<String,Linea>`** di tutte le linee del trasporto pubblico
  - **`cercaPercors`**i deve verificare che i nomi delle due fermate non siano nulli → altrimenti **`ErrorMessage`**
  - inoltre, entrambi i metodi devono emettere un **`ErrorMessage`** in caso di *parametri di ingresso nulli o vuoti*

MyCercatore (2/4)



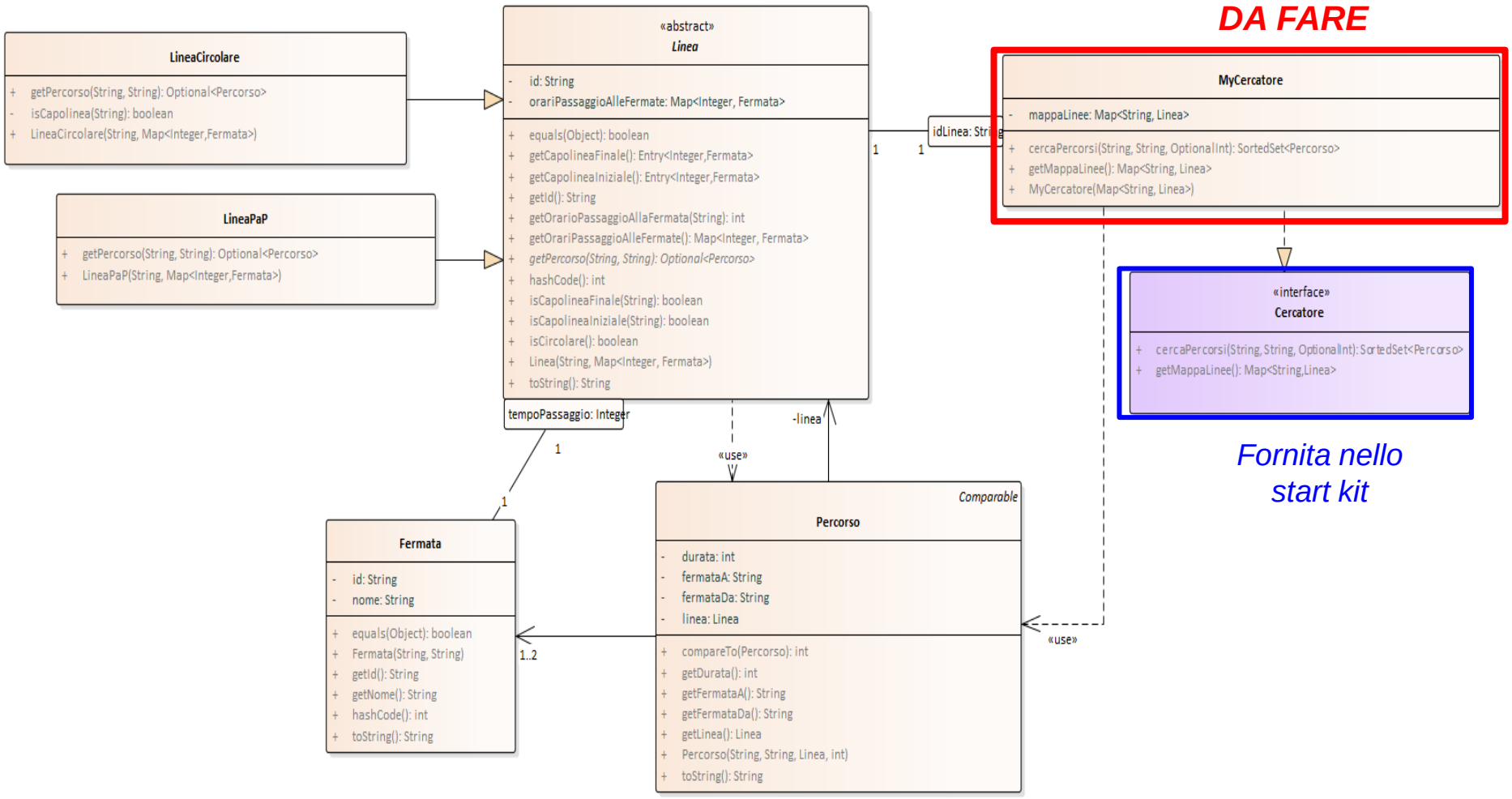
MyCercatore (3/4)



MyCercatore (4/4)

- Verificati i parametri, **`cercaPercorsi`** deve:
  - per ogni **`Linea`** della Mappa
    - chiedere alla **`Linea`** di restituire il **`Percorso`** tra le due fermate e, se questo esiste, memorizzarlo in **`SortedSet<Percorso>`**
    - se è presente l'**`OptionalInt`** che specifica la durata massima consentita, verificare i **`Percorso`** ottenuti e restituire in uscita solo quei **`Percorso`** la cui durata è ≤ a quella specificata;
      altrimenti (cioè se l'**`OptionalInt`** è assente), vanno restituiti tutti

Il modello (davvero) completo



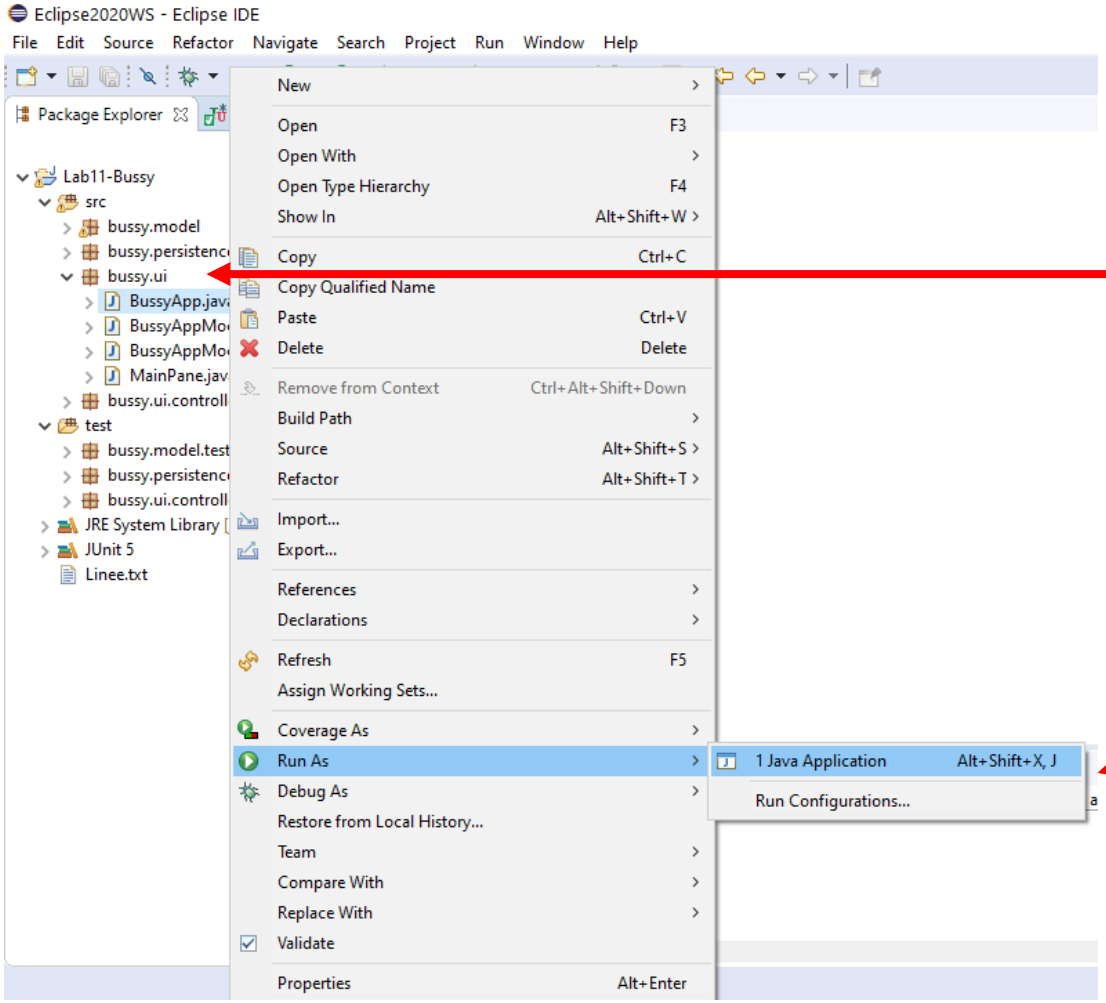
Da Fare

- Nello Startkit:
  - **`Cercatore`**
- Da fare
  - **`MyCercatore`**
- Dove lo metto:
  - **`bussy.model`**

OCCHIO ai i nomi dei package, delle classi e dei metodi, altrimenti i test …. ☹

Tempo a disposizione: 40 minuti

Per fare il run…



Dentro al package **`bussy.ui`** trovate **`BussyApp`**, tasto destro. Dovete impostare Javafx come spiegato nelle slide di installazione di Javafx. Poi potete fare il run

Hey!

KEEP
CALM
AND
HAPPY
CODING