Alma Mater Studiorum-Università di Bologna
Scuola di Ingegneria

Fondamenti di Informatica T2
Zanno Tassametro

Corso di Laurea in Ingegneria Informatica

Anno accademico 2021/2022

Prof. ROBERTA CALEGARI

Prof. AMBRA MOLESINI

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Il problema

"La compagnia Zann-O-Taxi, operante nella ridente ☺ cittadina di Zann-O-Town, ha richiesto un'applicazione che simuli il funziona-mento del tassametro dei propri mezzi, secondo la tipica tariffa mista a tempo e chilometri usata dai taxi in quasi tutte le città."

L'algoritmo più complesso che vedrete in questo corso

Lo presentiamo qui per rafforzare le competenze di algoritmica in un caso "decisamente tosto"

..ma niente paura! L'esame sarà assai più standard.. ! ☺ ☺

Analisi del dominio

"La compagnia Zann-O-Taxi, operante nella ridente cittadina di Zann-O-Town, ha richiesto un'applicazione che simuli il funzionamento del tassametro dei propri mezzi, secondo la tipica tariffa mista a tempo e chilometri usata dai taxi in quasi tutte le città."

Analisi dei dominio

- Il servizio taxi segue una tariffa con **scatto iniziale** + **tariffa tempo/distanza**
- Lo **scatto iniziale** è diverso per **corse diurne** o **notturne**
  - di giorno (dalle 6.00 alle 21.59 comprese): € 4.00
  - di notte (dalle 22.00 alle 5.59 comprese): € 6.00
- La tariffa progressiva a base mista tempo/distanza calcola il costo della corsa **in base sia alla sua durata sia alla distanza percorsa,** secondo certe regole
  - possono esserci più tariffe, applicabili in base a determinati parametri

Tariffe miste tempo / distanza

- Idea di fondo
  - quando il taxi è **in movimento**, la tariffazione avanza in proporzione alla **distanza percorsa**
  - quando invece il taxi è **“quasi fermo”** (code, etc.), la tariffazione procede **a tempo**

- Entrambe le modalità operano a scatti
  - quando il taxi è **in movimento**, uno scatto "ogni tot **metri** percorsi"
  - quando il taxi è **“quasi fermo”**, uno scatto "ogni tot **secondi** trascorsi"

- Nel caso di Zann-O-Town:
  - taxi **in movimento** = velocità ≥ 27 km/h
  - taxi **“quasi fermo”** = velocità < 27 km/h

..calcolata come?

Quali e quante tariffe?

- Il consiglio comunale di Zann-O-Town ha stabilito che:
  - per i taxi **in movimento** ci siano tre tariffe diverse [tariffe **T1**, **T2**, **T3**]
    in cui il costo/km aumenta man mano che la corsa si allunga
    (obiettivo: disincentivare le corse troppo lunghe)
  - per i taxi **"quasi fermi"** ci sia una tariffa unica [tariffa **T0**]
    uno scatto ogni 12 s (valore scatto: **15 € cent**)

- Tariffe a distanza (escluso scatto iniziale):
  - a partire da € 0.00 e fino a € 10.00 (esclusi):
    uno scatto ogni 100 m (valore scatto: **25 € cent**) [tariffa **T1**]
  - a partire € 10.00 e fino a € 25.00 (esclusi):
    uno scatto ogni 85 m (valore scatto: **20 € cent**) [tariffa **T2**]
  - a partire da € 25.00 in poi:
    uno scatto ogni 65 m (valore scatto: **15 € cent**) [tariffa **T3**]

..calcolate come?

Algoritmo: specifica

- Il tassametro ricalcola la velocità *ogni secondo*
  - ogni secondo il tassametro riceve una rilevazione dello spazio percorso e la usa per calcolare la *velocità media* mantenuta *dall'ultimo scatto*
- In base a ciò decide ogni volta se operare a tempo o a distanza
  - se la velocità misurata non supera i 27 km/h → opera a tempo [T0]
  - se la velocità misurata supera i 27 km/h → opera a distanza [T1,T2,T3]
- e stabilisce se debba o meno esserci uno scatto:
  - se opera a tempo, verificando se tempo trascorso ≥ durata scatto
    - T0: uno scatto ogni 12 s (valore scatto: 15 € cent)
  - se opera a distanza, verificando se spazio trascorso ≥ distanza scatto
    - T1: uno scatto ogni 100 m (valore scatto: 25 € cent)
    - T2: uno scatto ogni 85 m (valore scatto: 20 € cent)
    - T3: uno scatto ogni 65 m (valore scatto: 15 € cent)

Costo totale corsa = somma degli scatti

Algoritmo: dettaglio

- Per stabilire se debba esserci uno scatto, il tassametro deve
  - mantenere **lo spazio totale** percorso dall'ultimo scatto
  - mantenere **il tempo totale** trascorso dall'ultimo scatto
  - usare **l'uno o l'altro** (tariffa a distanza / a tempo) **secondo necessità**.
- Se c'è uno scatto, lo stato va resettato…
  - **sottrarre lo spazio totale** percorso dall'ultimo scatto
  - **sottrarre il tempo totale** trascorso dall'ultimo scatto
- …e al contempo si deve applicare la tariffa appropriata:
  - da € 0.00 e fino a € 10.00 (esclusi): valore scatto: **25 € cent** [tariffa **T1**]
  - da € 10.00 e fino a € 25.00 (esclusi): valore scatto: **20 € cent** [tariffa **T2**]
  - da € 25.00 in poi: valore scatto: **15 € cent** [tariffa **T3**]

E se lo scatto non c'è?

NB: alla fine della corsa, eventuali metri/secondi residui, che non abbiano dato luogo a uno scatto, sono ignorati (cioè non sono tariffati e vanno persi).

Algoritmo: esempio 1

- Corsa taxi composta da due sottotratti:
  - primo tratto a 10 km/h per 100 secondi
  - secondo tratto a 15 km/h per 100 secondi
- In entrambi i tratti la velocità è < 27 km/h → tariffa a tempo
  - T0 prevede uno scatto (da 15 cent) ogni 12 secondi
  - primo tratto: 100 s / 12 s = 8 scatti
  - secondo tratto: 100 s / 12 s = 8 scatti
  - totale: 16 scatti da 15 cent → corso corsa = € 2.40 [+ scatto iniziale]

Algoritmo: esempio 2

- Corsa taxi composta da due sottotratti:
  - primo tratto a 30 km/h per 60 secondi
  - secondo tratto a 60 km/h per 60 secondi
- In entrambi i tratti la velocità è > 27 km/h → tariffa a distanza
  - T1 prevede uno scatto (da 25 cent) ogni 100 m, fino a max € 10
  - primo tratto: 30 km/h x 60 s = 0.5 km → 500 m / 100 m = 5 scatti
  - secondo tratto: 60 km/h x 60 s = 1.0 km → 1000 m / 100 m = 10 scatti
  - totale: 15 scatti da 25 cent → costo corsa = € 3.75 [+ scatto iniziale]

Algoritmo: esempio 3

- Corsa taxi composta da due sottotratti:
  - primo tratto a 10 km/h per 60 secondi
  - secondo tratto a 60 km/h per 60 secondi
- Nel primo tratto la velocità è < 27 km/h → tariffa a tempo
  - T0 prevede uno scatto (da 15 cent) ogni 12 s
  - primo tratto: 60 s / 12 s = 5 scatti
  - subtotale: 5 scatti da 15 cent → costo parziale = € 0.75
- Nel secondo tratto la velocità è > 27 km/h → tariffa a distanza
  - T1 prevede uno scatto (da 25 cent) ogni 100 m, fino a max € 10
  - plafond disponibile a questa tariffa: € 9.25 (€ 10 – € 0.75)
  - secondo tratto: 60 km/h x 60 s = 1.0 km → 1000 m / 100 m = 10 scatti
  - subtotale: 10 scatti da 25 cent → costo parziale = € 2.50 (ben < 9.25: ok)
- Costo totale: € 0.75 + € 2.50 + scatto iniziale = € 3.25 + sc. iniziale

Algoritmo: esempio 4

- Corsa taxi composta da un'unica tratta:
  - viaggio a 60 km/h per 600 secondi (10 minuti)
  - quindi, lo spazio percorso totale è 10 km (1 km/min x 10 minuti)
- La velocità è costante > 27 km/h → più tariffe a distanza
  - T1 prevede uno scatto (da 25 cent) ogni 100 m, fino a max € 10 ossia fino a max 40 scatti, equivalenti a una distanza di 4 km
  - quindi, i successivi 6 km devono essere tariffati a tariffa T2 (e/o T3)
  - T2 prevede uno scatto (da 20 cent) ogni 85 m, fino a max € 25 → plafond effettivo disponibile a questa tariffa € 15 (perché €10 già usati)
  - tratto unico a 60 km/h = 1.0 km/min → 6000 m / 85 m = 70 scatti
  - subtotale 70 scatti da 20 cent = € 14 (di poco, ma entro il plafond!)
  - costo totale corsa: € 10 + € 14 + scatto iniziale = € 24 + scatto iniziale

Dal problema alla struttura

- **Ma… è complicatissimo!! ☹**
  - eh, it's the real world, my friend…
- **Come organizzare un sistema software del genere?**
  - estrarre dalla descrizione del problema alcune *entità chiave*
  - minimizzare le dipendenze reciproche → interfacce prima, classi poi
  - se ci si accorge che un'entità deve *conoscere (troppi) dettagli di altre* per fare certe operazioni, probabilmente *la struttura è sbagliata* e va rivista
- **Come scegliere fra interfacce e classi?**
  - se una entità rappresenta qualcosa di piuttosto concreto, di cui sappiamo anche i dettagli e non si prevedono implementazioni variegate, classe
  - se viceversa una entità rappresenta qualcosa di più astratto, *di cui non si possono in generale precisare i dettagli* ed è invece prevedibile possano esistere implementazioni (anche molto) diverse, interfaccia

Entità concettuali

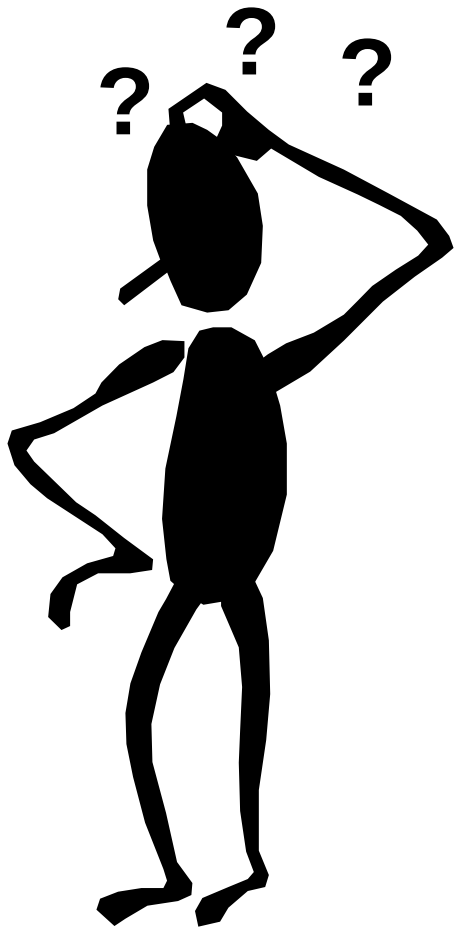
Corsa Taxi

Tariffa

Tassametro

Scatto

Fascia Oraria



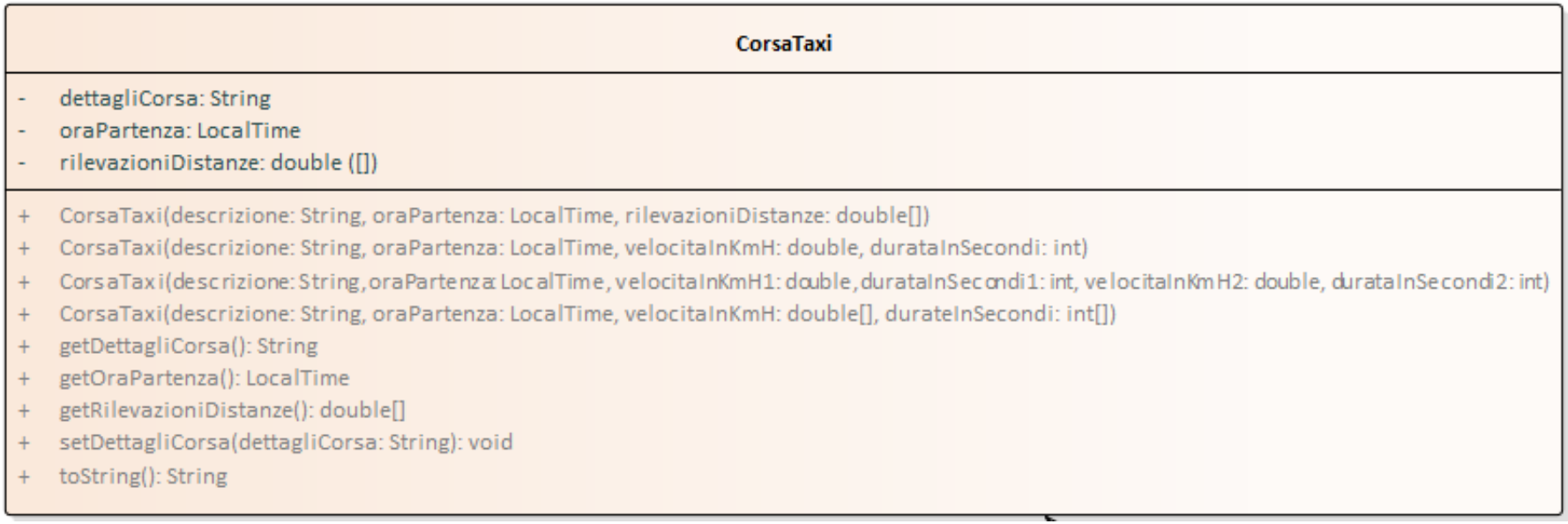
Analisi del problema

- **Dalla descrizione del dominio emerge in primis la *corsa taxi***
  - entità caratterizzata da *ora di inizio e rilevazioni di distanze percorse* misurate sul taxi ogni secondo → decisamente alquanto *concreta*
  - non prevediamo specializzazioni: è una descrizione autocontenuta → classe

- **CorsaTaxi** racchiude in sé tutte le informazioni relative ad una corsa:
  - ora di partenza
  - array di **rilevazioni di distanze** (**una per secondo**) misurate in **metri**: per recuperare la rilevazione della distanza totale percorsa sino all'n-esimo secondo basta accedere alla casella n-esima dell'array
- **CorsaTaxi** è dotata di costruttori extra che agevolano la scrittura dei test

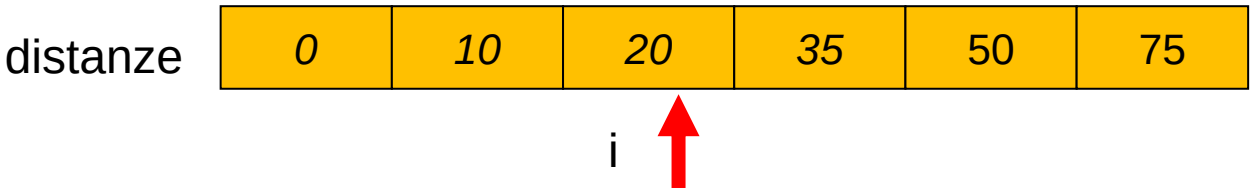
Corsa Taxi

- **CorsaTaxi** racchiude in sé tutte le informazioni relative ad una corsa:
  - ora di partenza
  - array di **rilevazioni di distanze** (**una per secondo**) misurate in **metri**: per recuperare la rilevazione della distanza totale percorsa sino all'n-esimo secondo basta accedere alla casella n-esima dell'array
- **CorsaTaxi** è dotata di costruttori extra che agevolano la scrittura dei test



Corsa Taxi

- **CorsaTaxi** racchiude un array di **rilevazioni di distanze** (**una per secondo**) misurate in **metri**: per recuperare la rilevazione di distanza totale percorsa sino all'n-esimo secondo basta accedere alla casella n-esima dell'array



- Ogni casella dell'array contiene la rilevazione **dello spazio totale percorso** sino al all'istante t=i
- Attenzione per il calcolo della velocità media occorre calcolare il lo spazio percorso tra due rilevazioni adiacenti -->

  distanza = distanze[i] – distanze[i-1]

Analisi del problema

- **A fianco della Corsa taxi, dalla descrizione emerge la *Tariffa taxi***
  - qui la faccenda è diversa: il dominio prevede *varie e diverse tariffe, con caratteristiche piuttosto diverse fra loro*
  - opportuno predisporre una interfaccia che si possa poi concretizzare in molti modi diversi secondo necessità

- **ITariffaTaxi** definisce una tariffa taxi come entità caratterizzata da:
  - nome
  - uno **scatto** che incapsula il cambiamento di stato (ossia, ciò che determina l'avanzamento della tariffazione)
  - il **valore** di quello scatto (ossia, quanto aumenta il costo della corsa)

Ma lo scatto c'è sempre?

Analisi del problema

- Osserviamo che **non tutte le rilevazioni** (di tempo e distanza) **portano ad originare uno scatto**
  - se il taxi si è mosso poco e/o non è passato abbastanza tempo
- Problema: in tali casi, *cosa restituisce* **`getScattoCorrente`** *?*
  - il metodo dovrebbe restituire uno **`Scatto`**, ma se lo scatto non c'è..?
  - un'idea potrebbe essere restituire **`null`**…
    - MA restituire davvero un `null` *inietterebbe fragilità* nel software
    - i clienti dovrebbero sempre verificare se la funzione "restituisce davvero" qualcosa → codice pieno di `if(res!=null)`
    - si renderebbero indistinguibili situazioni strutturalmente diverse (veri errori vs. risultato non trovato/non prodotto)
  - decisamente più appropriato restituire un **`Optional<Scatto>`**

ITariffaTaxi

- **ITariffaTaxi** definisce una tariffa taxi come entità caratterizzata da:
  - nome
  - uno **scatto** che incapsula il cambiamento di stato
    (ossia, ciò che determina l'avanzamento della tariffazione)
  - il **valore** di quello scatto (di quanto aumenta il costo della corsa)

Analisi del problema

- **Infine, la descrizione pone al centro la figura del *Tassametro* come entità che calcola il costo della corsa in base alle tariffe**
  - concreta? astratta?
- Una prima, ovvia idea è definire una classe **ZannoTassametro** cablando in essa l'algoritmo specifico di Zann-O-Town
  - in tal caso, il metodo **calcolaCostoCorsa** prende in ingresso una **CorsaTaxi** e restituisce il costo della corsa

Mmmh… è una buona soluzione?

In realtà, non tanto: anzi,
è una soluzione alquanto "bovina"

Perché è una soluzione  ?

- Quante cose deve fare la classe ZannoTassametro?
- Si può garantire che l'algoritmo sia l'unico possibile?
- Si può garantire che i parametri considerati siano gli unici?

Se la risposta è NO, è bene **interporre opportune astrazioni** e pensare ad una ***soluzione più generale***

- Si può garantire che l'algoritmo sia l'unico possibile? **NO!**
- Si può garantire che i parametri considerati siano gli unici? **NO!**

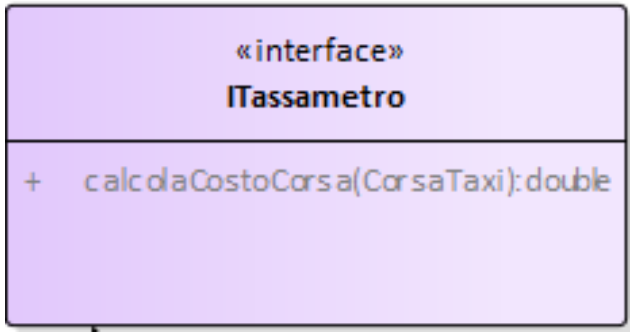
Soluzione NON Bovina

- **Design for Change**
  - non si può **prevedere** quale sarà l'evoluzione del progetto (Arti Divinatorie non è nel piano di studi)
  - MA si può e si deve **predisporre** il software per reggere al **cambiamento**

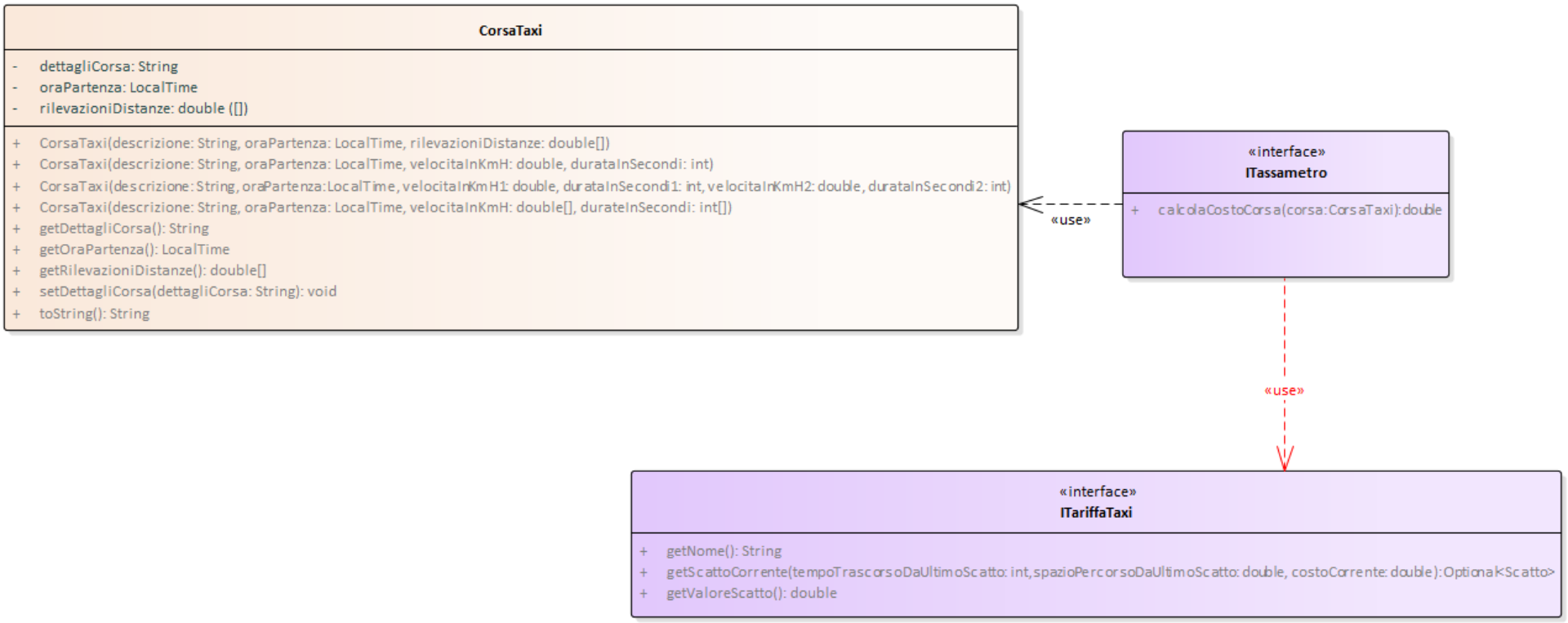
- **Cablare la soluzione vs. configurare la soluzione**
  - IDEA: predisporre **tante e tali astrazioni** da rendere la **soluzione al caso specifico** una ***configurazione specifica di una soluzione generale***
  - OVVERO: rendere il problema da risolvere un *caso particolare di una soluzione più generale*

Il tassametro generico

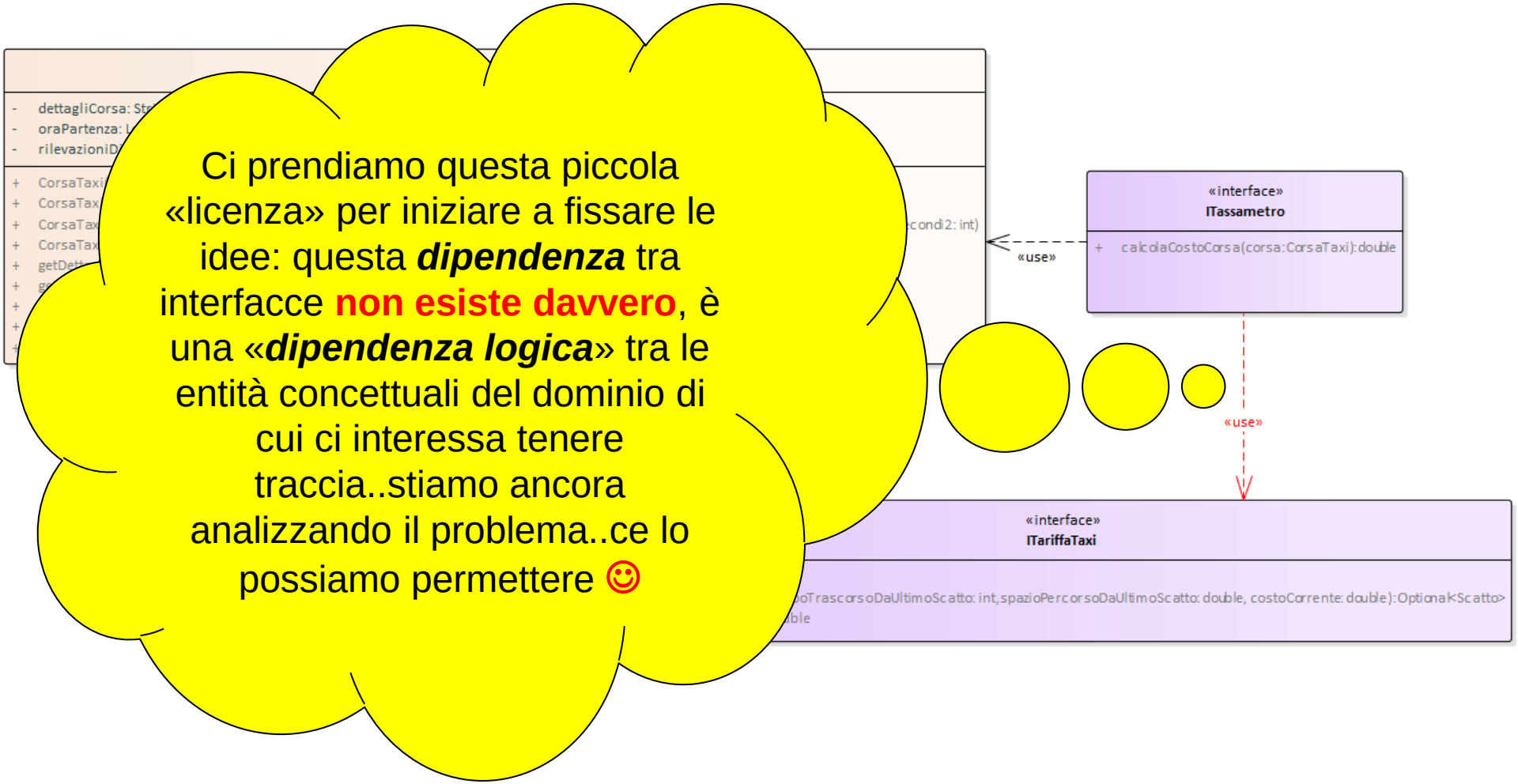
- Un tassametro generico calcola il costo di una corsa di taxi
  - non prefigura COME
  - non prestabilisce NIENTE
  - espone soltanto un'ESIGENZA
- Serve una astrazione che *esponga la funzionalità* richiesta senza aggiungere altre specifiche: è il ritratto di **un'interfaccia**
  - si etichetta come **`ITassametro`** qualunque entità in grado di rispondere alla domanda "Mi calcoli il costo di una corsa di taxi?"



Il modello (1)


CorsaTaxi
- dettagliCorsa: String
- oraPartenza: LocalTime
- rilevazioniDistanze: double ([])
+ CorsaTaxi(descrizione: String, oraPartenza: LocalTime, rilevazioniDistanze: double[])
+ CorsaTaxi(descrizione: String, oraPartenza: LocalTime, velocitaInKmH: double, durataInSecondi: int)
+ CorsaTaxi(descrizione: String, oraPartenza: LocalTime, velocitaInKmH1: double, durataInSecondi1: int, velocitaInKmH2: double, durataInSecondi2: int)
+ CorsaTaxi(descrizione: String, oraPartenza: LocalTime, velocitaInKmH: double[], durateInSecondi: int[])
+ getDettagliCorsa(): String
+ getOraPartenza(): LocalTime
+ getRilevazioniDistanze(): double[]
+ setDettagliCorsa(dettagliCorsa: String): void
+ toString(): String
«use»
«interface»
ITassametro
+ calcolaCostoCorsa(corsa: CorsaTaxi): double
«use»
«interface»
ITariffaTaxi
+ getNome(): String
+ getScattoCorrente(tempoTrascorsoDaUltimoScatto: int, spazioPercorsoDaUltimoScatto: double, costoCorrente: double): Optional<Scatto>
+ getValoreScatto(): double

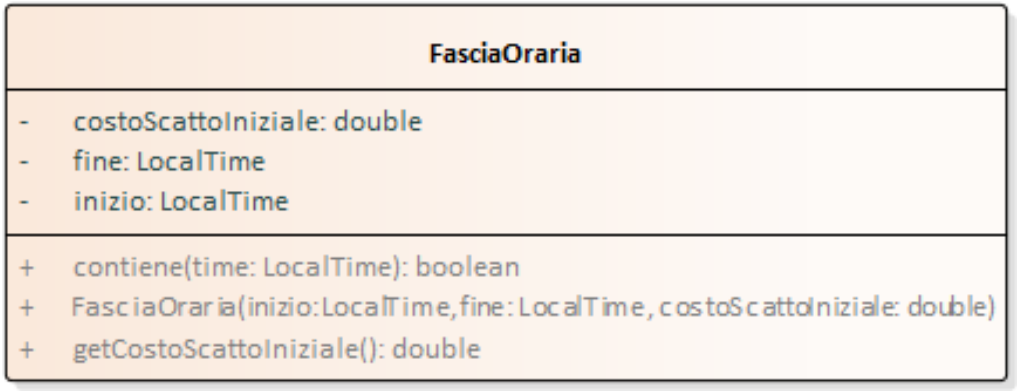
Il modello (1)



Ci prendiamo questa piccola «licenza» per iniziare a fissare le idee: questa ***dipendenza*** tra interfacce **non esiste davvero**, è una «***dipendenza logica***» tra le entità concettuali del dominio di cui ci interessa tenere traccia..stiamo ancora analizzando il problema..ce lo possiamo permettere ☺

Tassametro a fasce orarie (1/5)

- Il tassametro a fasce orarie diventa solo *uno dei tanti possibili*
  - quello che serve ora
  - a Zann-O-Town è basato sul concetto di **fascia oraria**
- Nuova entità concreta → nuova classe:



  - **`contiene(LocalTime): boolean`**
    indica se l'orario passato è contenuto nella fascia oraria
  - **`getCostoScattoIniziale(): double`**
    restituisce il costo dello scatto iniziale (se la corsa inizia entro la fascia oraria)

Tassametro a fasce orarie (2/5)

- L'interfaccia **`ITariffaTaxi`** è quindi *ancor più strategica*
- Espone le funzionalità di una "tariffa taxi" *senza anticipare particolari scelte*

- Dovrà essere concretizzata in almeno due modi:
  - tariffe a distanza: scattano in base ai *chilometri* percorsi
  - tariffe a tempo: scattano in base ai *secondi* trascorsi

→ due o più classi

Le riprendiamo a breve, ora pensiamo al Tassametro

Tassametro a fasce orarie (3/5)

- Il tassametro a fasce orarie concretizza l'astrazione **`ITassametro`**
  - al suo interno, deve effettuare il calcolo del costo della corsa
- **Come si fa il calcolo?**
  - idea brutale: a ogni rilevazione, determinare se agire a tempo o a distanza
  - stabilito ciò, recuperare dalla tariffa rispettivamente il tempo o la distanza di scatto e fare i calcoli
  - **sembra una buona idea.. ma il diavolo è nei dettagli!**

    - La tariffa è un oggetto generico, accessibile tramite l'interfaccia **`ITariffaTaxi`** *messa lì apposta* per non vedere dettagli
    - MA per "recuperare dalla tariffa" il tempo o la distanza di scatto bisogna invece sapere *di che classe sia* l'oggetto-tariffa concreto
      - solo la classe TariffaADistanza ha il metodo getDistanzaDiScatto
      - solo la classe TariffaATempo ha il metodo getTempoDiScatto
    - **inevitabile uso di `instanceof` + cast ☹☹**

Tassametro a fasce orarie (3/5)

- Il tassametro a fasce orarie concretizza l'astrazione **`ITassametro`**
  - al suo interno, deve effettuare il calcolo del costo della corsa
- **Come si fa il calcolo?**
  - idea brutale: a ogni rilevazione, determinare se agire a tempo o a distanza
  - stabilito ciò, recuperare dalla tariffa rispettivamente il tempo o la distanza di scatto e fare i calcoli
  - **sembra una buon**

    - La tariffa è u **`ITariffa`**
    - MA per "rec … bisogna inve…
      - solo la cla…
      - solo la classe Tariff… …mpo ha il metodo getTempoDiScatto
    - **inevitabile uso di `instanceof` + cast** ☹☹

Rientrerebbero dalla finestra quelle *dipendenze dagli oggetti specifici* che, con **`ITariffaTaxi`**, abbiamo proprio cercato invece di chiudere fuori dalla porta!

È una impostazione sbagliata e naif

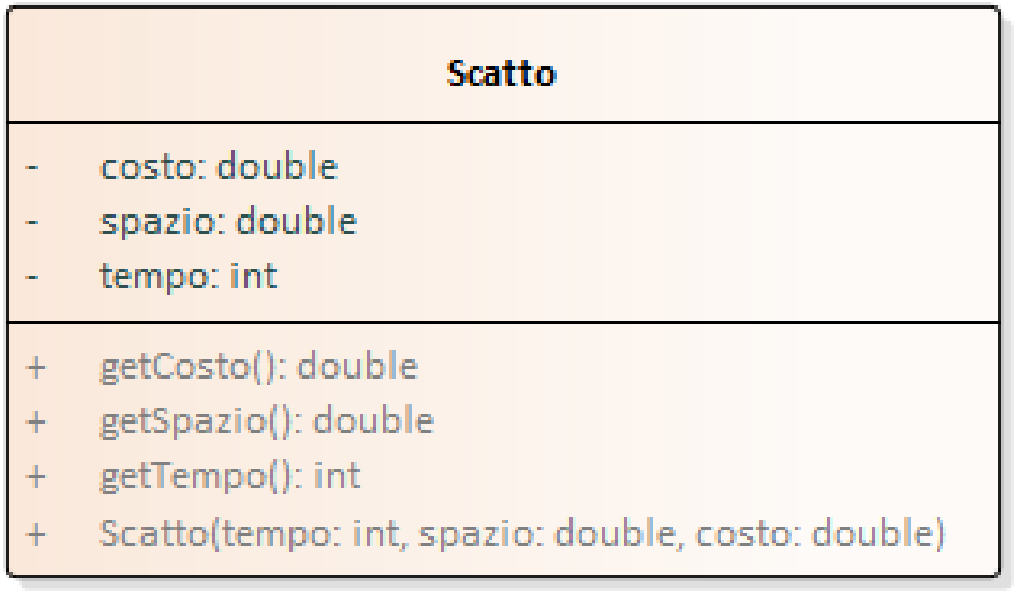
Tassametro a fasce orarie (4/5)

- Il tassametro a fasce orarie concretizza l'astrazione **`ITassametro`**
  - al suo interno, deve effettuare il calcolo del costo della corsa
- **Punto chiave: *far fare il lavoro all'oggetto "giusto"***
  - chi sa di che tariffa si tratta..? Ovvio: la tariffa stessa!
  - ergo, a ogni rilevazione, chiediamo alla tariffa di *calcolare e restituire lei* lo "scatto", ossia la *variazione di stato* necessaria
  - si sfrutta il polimorfismo: **MIGLIORE STRUTTURA = MIGLIOR CODICE**

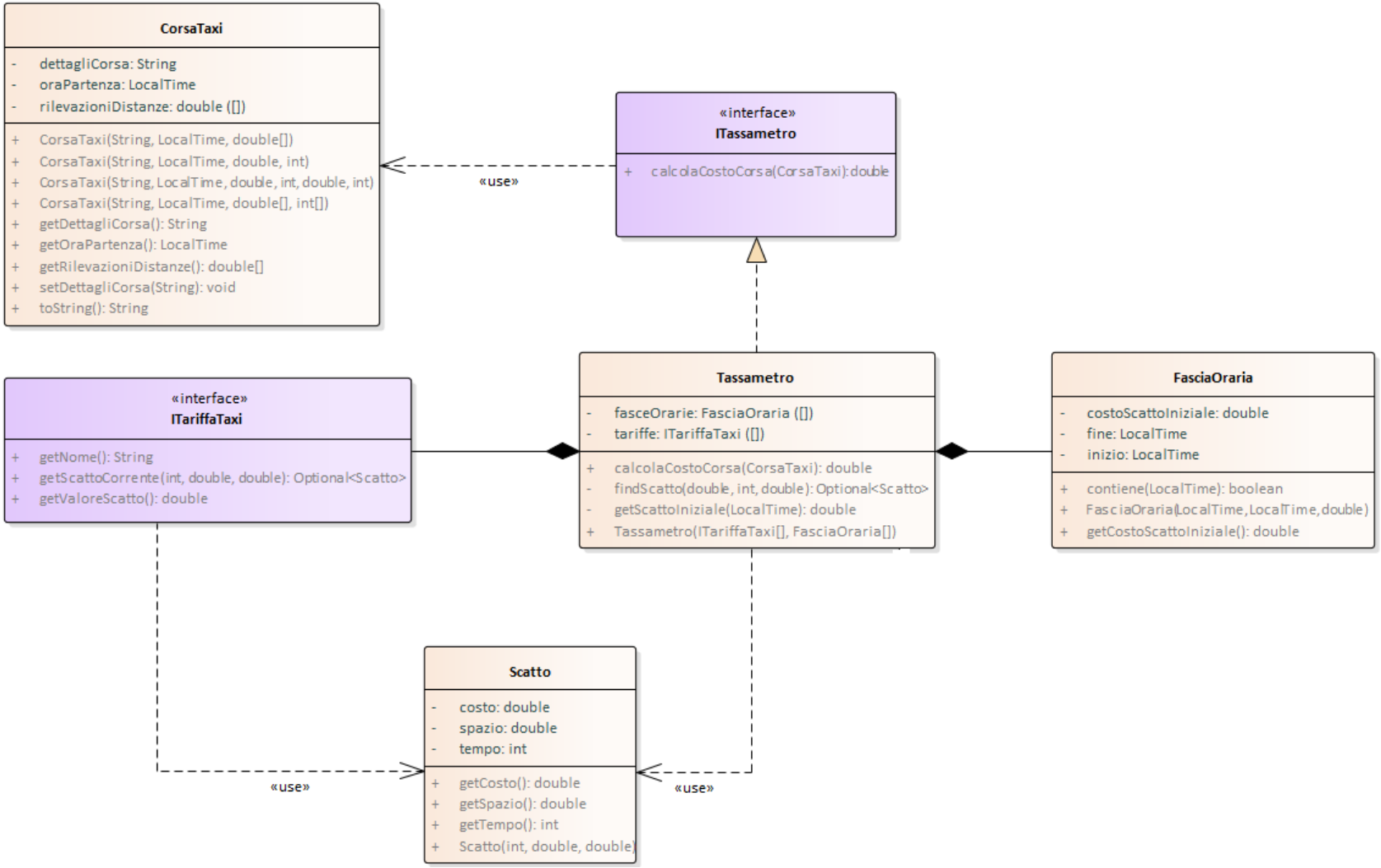
- La tariffa è un'entità astratta, accessibile tramite l'interfaccia **`ITariffaTaxi`** (*fatta apposta* per non vedere dettagli)
- **Giustamente, continuiamo a trattarla come tale:** NON occorre MAI sapere di che classe sia l'oggetto-tariffa concreto!
- Non c'è alcun bisogno di **`instanceof`** + cast ☺ ☺

Tassametro a fasce orarie (5/5)

- La classe **`Scatto`** incapsula i dati del nuovo stato dopo lo scatto
  - tempo, spazio, costo
  - metodi accessor



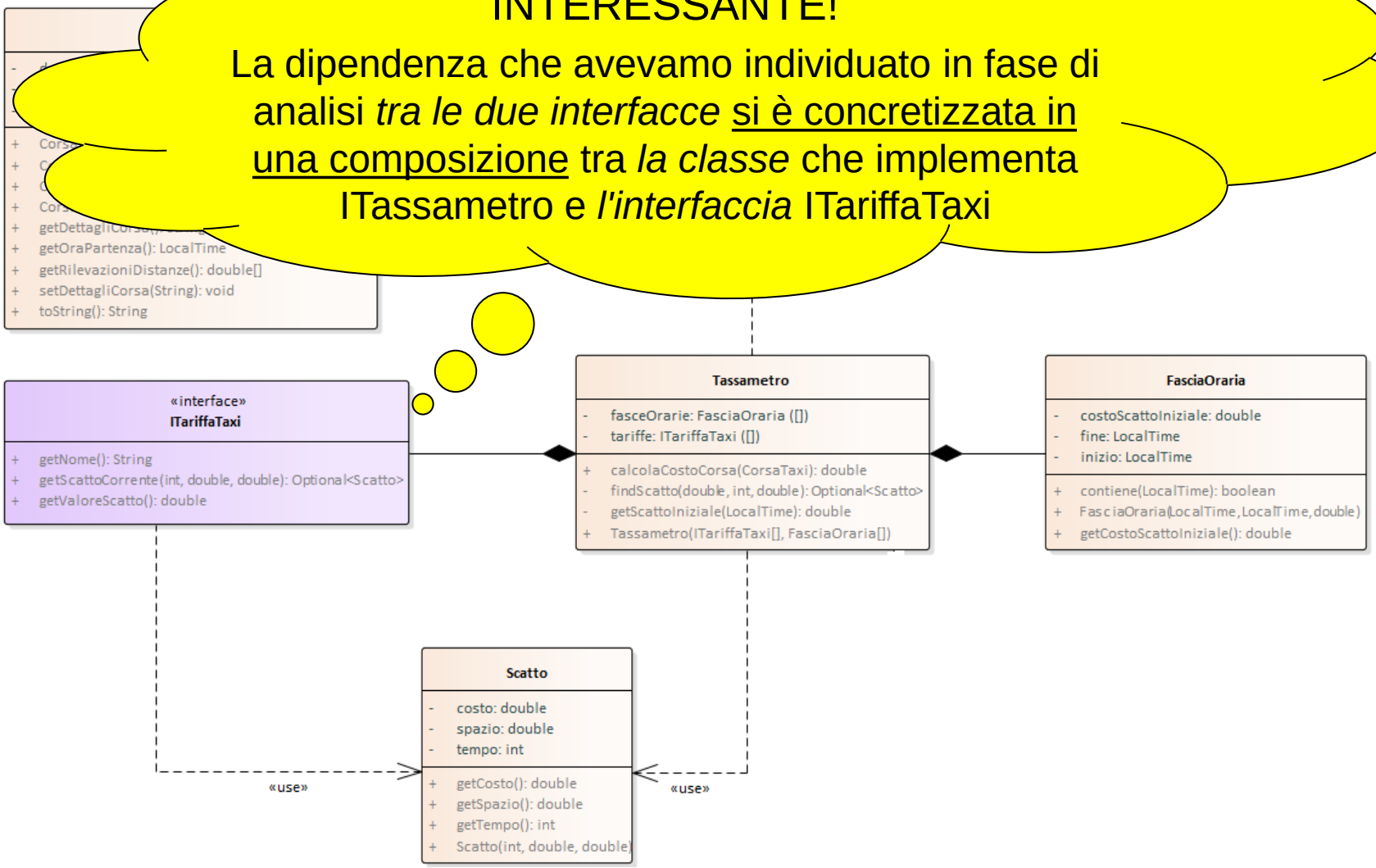
Il modello complessivo



Il modello complessivo

INTERESSANTE!

La dipendenza che avevamo individuato in fase di analisi *tra le due interfacce* si è concretizzata in una composizione tra *la classe* che implementa ITassametro e *l'interfaccia* ITariffaTaxi



Algoritmo di massima

- Dobbiamo mantenere (e all'inizio azzerare):
  - il «costo corrente» (**cv**)
  - il «tempo trascorso dall'ultimo scatto» (**t**)
  - lo «spazio percorso dall'ultimo scatto» (**s**)
- Logica delle rilevazioni:
  - viene fatta una rilevazione di distanza ogni secondo
  - di conseguenza:
    - il «tempo trascorso dall'ultimo scatto» aumenta di T=1 sec: **t** += T
    - lo «spazio percorso dall'ultimo scatto» aumenta di S: **s** += S
- Logica degli scatti:
  - ogni secondo si cerca se c'è una tariffa che possa scattare
  - se sì, le si fa calcolare lo **scatto** e lo si somma al costo: **cv** += **scatto**
- Alla fine, il costo della corsa è ***cv + scatto iniziale***

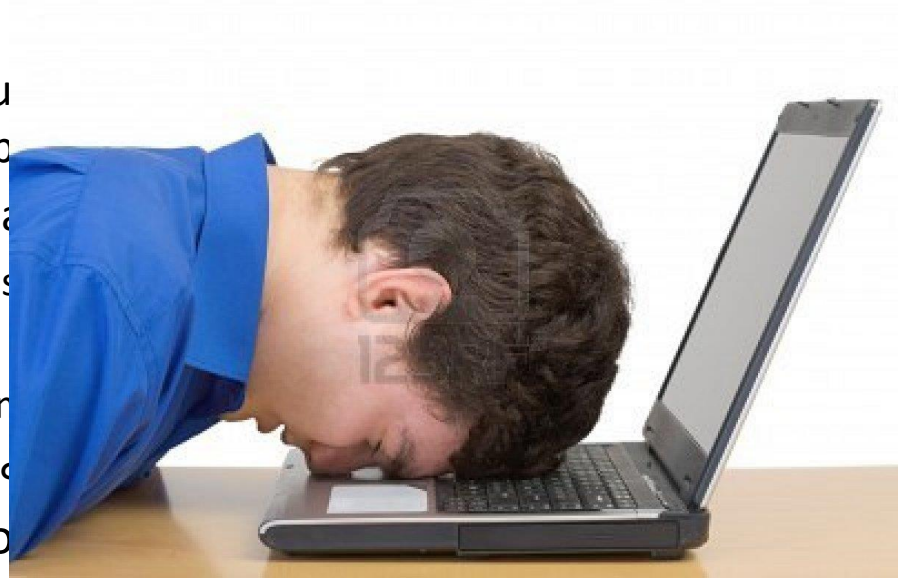
Algoritmo: specifiche

- **Algoritmo `calcoloCostoCorsa(CorsaTaxi)`**
  - azzerare il «costo variabile corrente» (**cv**), il «tempo trascorso dall'ultimo scatto» (**t**) e lo «spazio percorso dall'ultimo scatto» (**s**)
  - poi, per ogni rilevazione di distanza (una al secondo):
    - sommare ad **s** lo spazio percorso a partire dalla rilevazione precedente
    - incrementare **t** di un secondo
    - cercare la prima tariffa che può scattare (tenendo conto di **cv**, **t**, **s**) e restituire lo **`Scatto`** corrispondente → **`findScatto(…)`**
    - se l'ha trovato, recuperare i dati rilevanti dallo **`Scatto`** e:
      - sommare a **cv** il costo dello scatto
        → **`scatto.getCosto`**
      - sottrarre a **t** ed **s**, rispettivamente, tempo e spazio di scatto
        → **`scatto.getTempo`**, **`scatto.getSpazio`**
  - al termine, **cv** rappresenta il costo variabile della corsa…
  - …a cui va sommato lo scatto iniziale → **`getScattoIniziale(…)`**

Algoritmo: specifiche

- **Algoritmo `calcoloCostoCorsa(CorsaTaxi)`**
  - azzerare il «costo variabile corrente» (**cv**), il «tempo trascorso dall'ultimo scatto» (**t**) e lo «spazio percorso dall'ultimo scatto» (**s**)
  - [illegible]stanza (NB: sono una al secondo):
    - [illegible]rso a partire dalla rilevazione precedente
    - [illegible]do
    - [illegible]a che pu
    - e restituire lo **`Scatto`** corrisp
    - se l'ha trovato, recuperare i da
      - sommare a **cv** il costo dello s
        - → `scatto.getCosto`
      - sottrarre a **t** ed **s**, rispettivar
        - → `scatto.getTempo`, `sc`
  - al termine, **cv** rappresenta il co
  - …a cui va sommato lo scatto iniziale → `getScattoIniziale(…)`

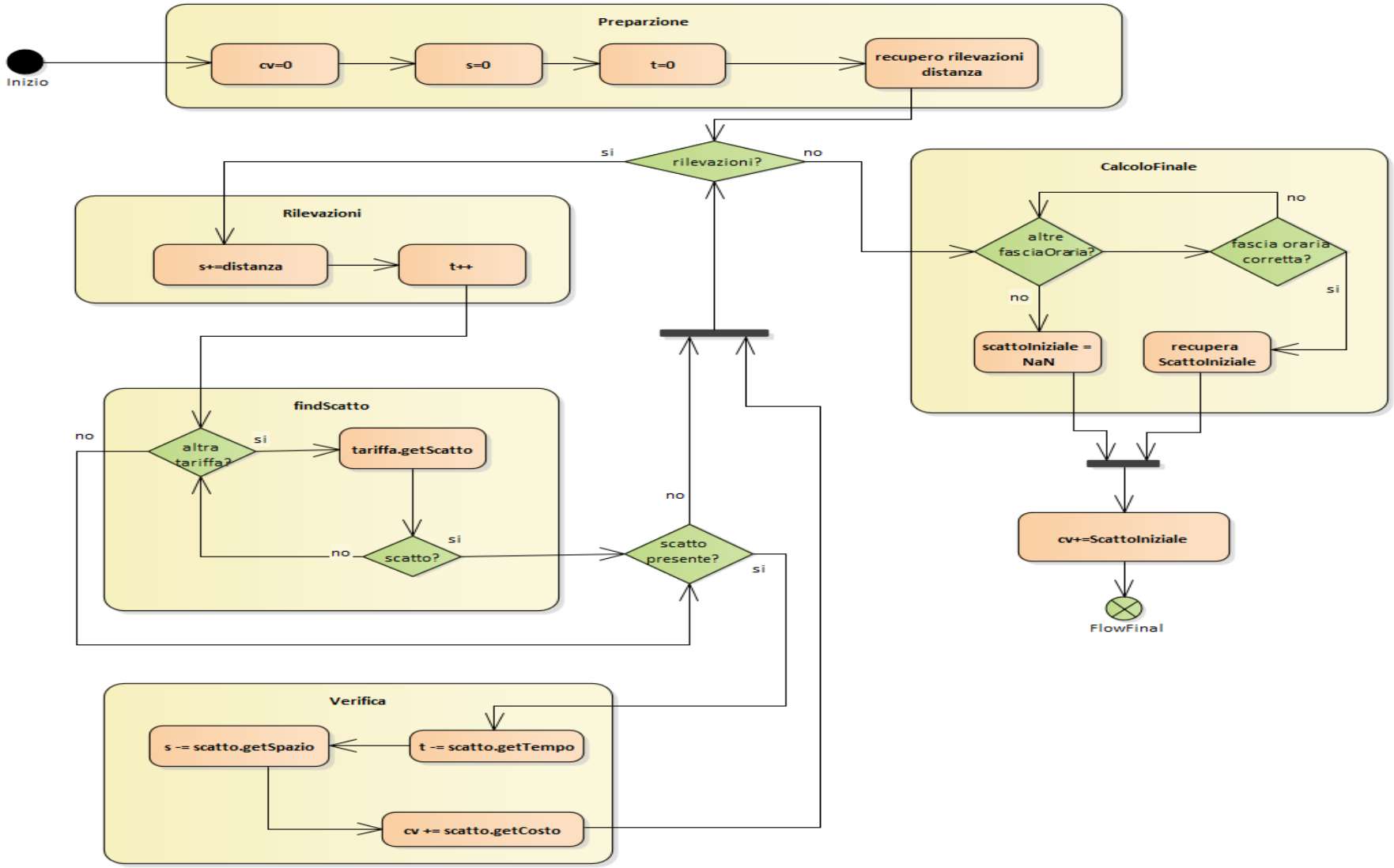
Tutto chiaro??



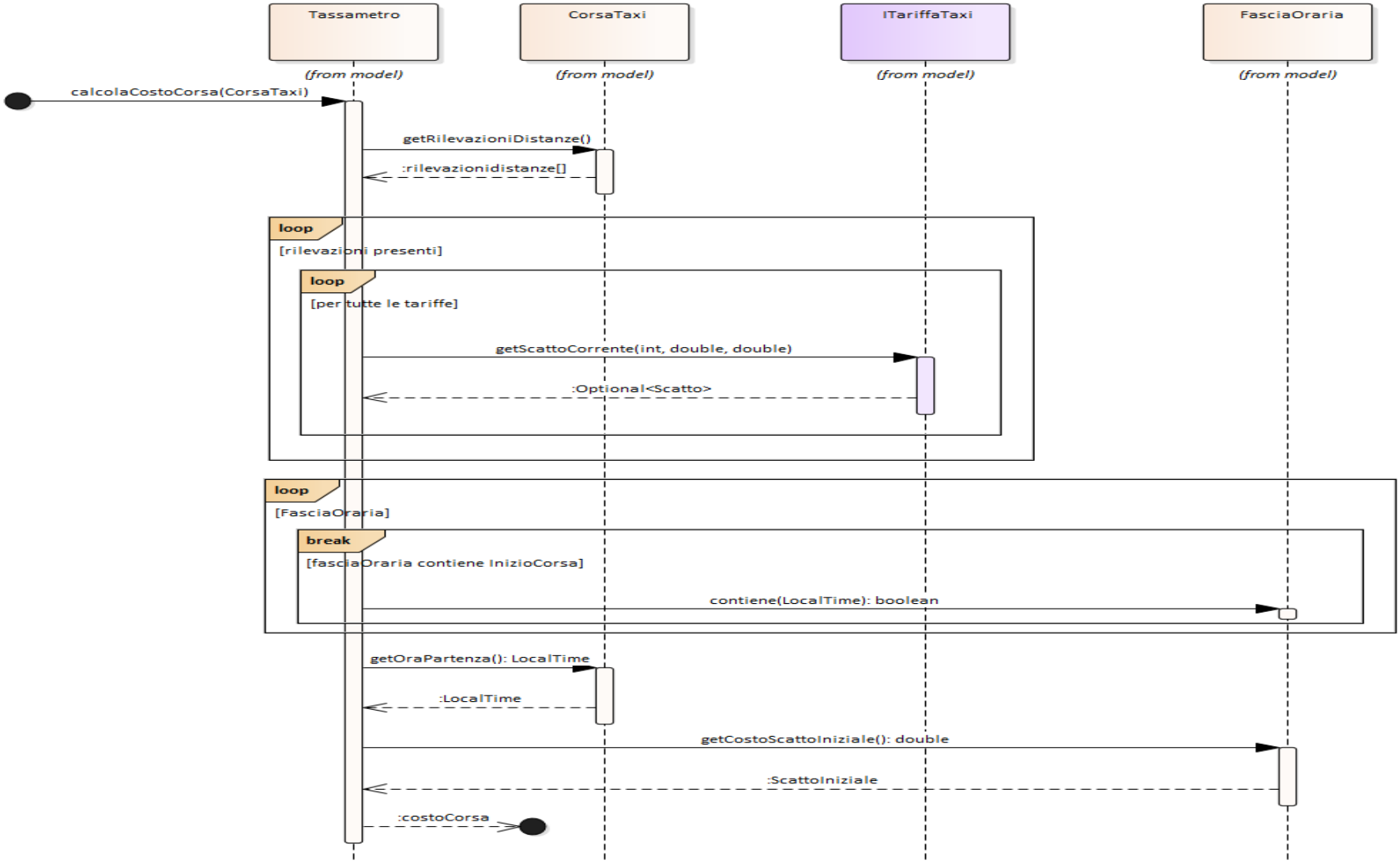
Algoritmo: dinamica e attività

- Per seguire il funzionamento di questo algoritmo la sola vista strutturale non è sufficiente: *LA STRUTTURA NON È TUTTO!*
- Ci servono altre «viste» che illustrino la *DINAMICA*
  - «interazione» = scambio di messaggi (« chi chiama chi»)
  - «comportamento» = flusso di esecuzione (« cosa si fa e in che ordine »)
- A tale scopo si sfruttano *altri tipi di diagrammi UML*
  - «interazione» = DIAGRAMMA DI SEQUENZA
  - «comportamento» = DIAGRAMMA DELLE ATTIVITÀ
- **NB: conoscerli NON è obiettivo di questo corso**
  - semplicemente, li usiamo in questo esercizio perché fanno comodo..
  - .. e in questo caso sono alquanto "intuitivi"

Tassametro: diagramma attività

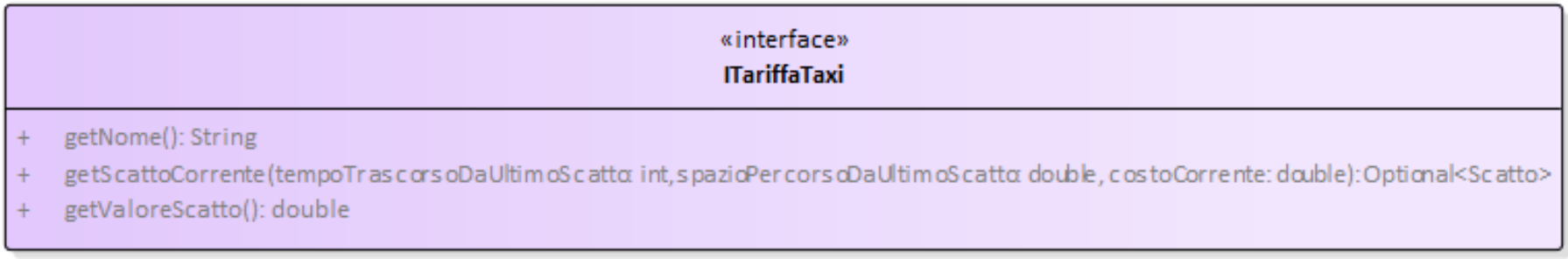


Algoritmo: diagramma di sequenza



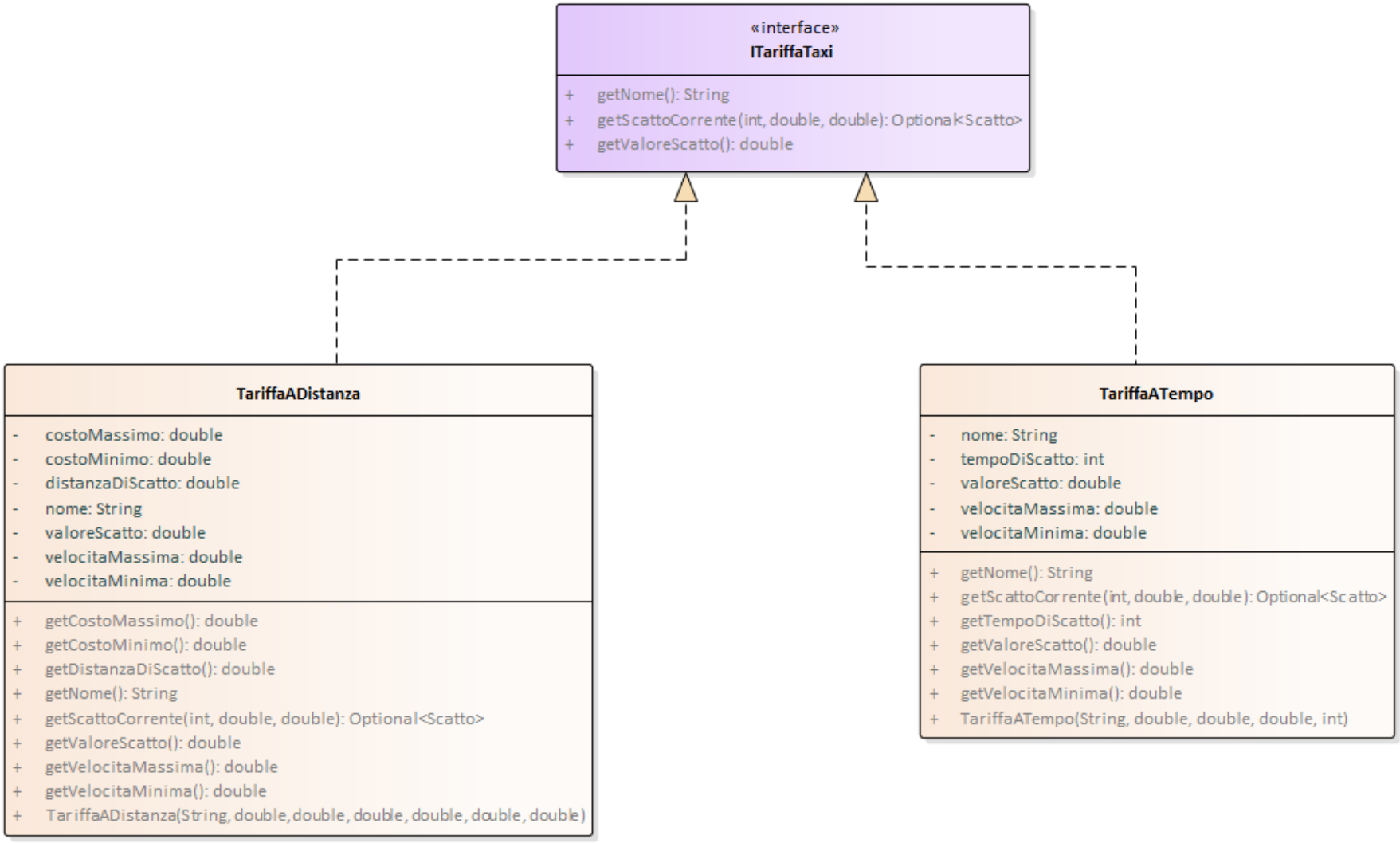
Tariffe varie ed eventuali

- Avevamo scelto un'interfaccia per le tariffe perché possono essere *molte e varie* e non volevamo anticipare troppe scelte

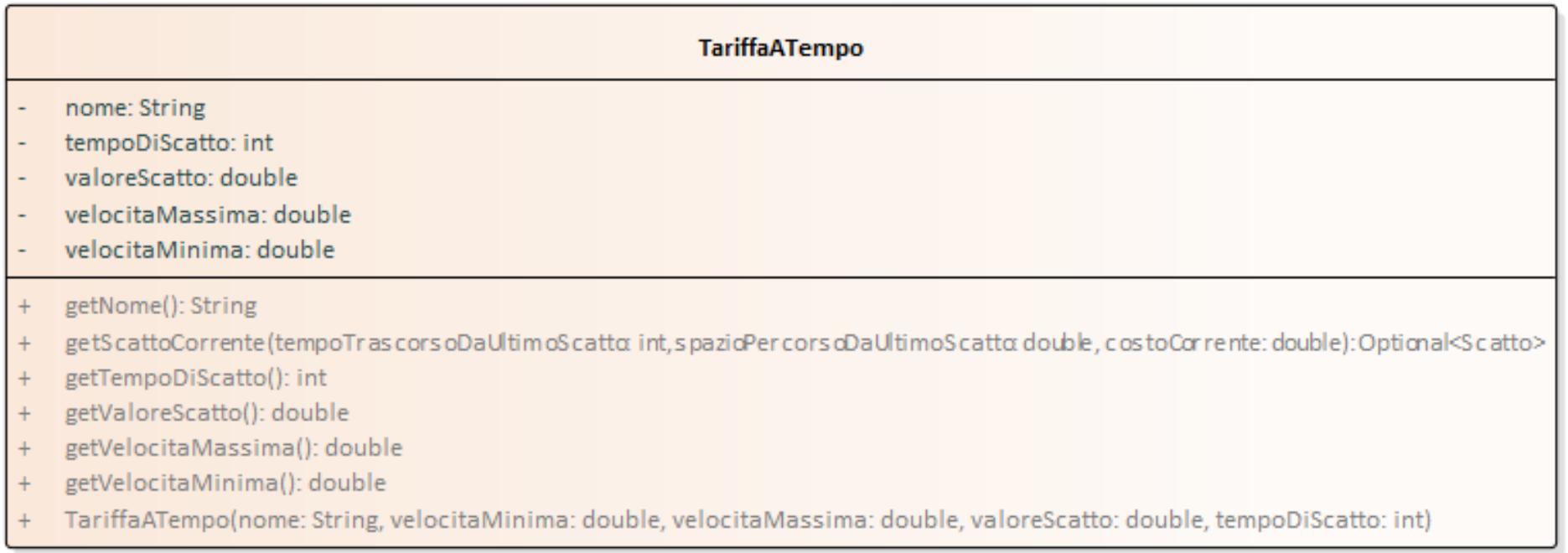


- **Ora dobbiamo concretizzare quell'astrazione**
- Servono almeno due classi che la realizzino nei due casi di:
  - tariffe a **tempo**
    → per Zann-O-Town: **una sola istanza** (T0)
  - tariffe a **distanza**
    → per Zann-O-Town: **tre istanze** (T1, T2, T3) **diversamente configurate**

Tariffe: il modello



Tariffa a tempo



ATTENZIONE alle unità di misura

- il tassametro misura i tempi in *secondi* e le distanze in *metri*
- ma le tariffe richiedono velocità espresse in *km/h* …!

Tariffa a tempo

- Scatta quando sono vere tutte le condizioni che seguono:
  - la velocità media dall'ultimo scatto è **maggiore o uguale** alla velocità minima configurata (eventualmente 0)
  - la velocità media dall'ultimo scatto è **minore** della velocità massima configurata (eventualmente infinito)
  - il tempo trascorso dall'ultimo scatto è **maggiore o uguale** al tempo di scatto configurato
- In tal caso, produce e restituisce lo Scatto appropriato:

Indica se lo scatto s'ha da fare

```
boolean effettuaScatto = ....
return effettuaScatto
        ? Optional.of(new Scatto(tempoDiScatto, spazioPercorsoDaUltimoScatto, valoreScatto))
        : Optional.empty();
```

Tariffa a distanza

TariffaADistanza

- costoMassimo: double
- costoMinimo: double
- distanzaDiScatto: double
- nome: String
- valoreScatto: double
- velocitaMassima: double
- velocitaMinima: double

+ getCostoMassimo(): double
+ getCostoMinimo(): double
+ getDistanzaDiScatto(): double
+ getNome(): String
+ getScattoCorrente(tempoTrascorsoDaUltimoScatto: int, spazioPercorsoDaUltimoScatto: double, costoCorrente: double): Optional<Scatto>
+ getValoreScatto(): double
+ getVelocitaMassima(): double
+ getVelocitaMinima(): double
+ TariffaADistanza(nome: String, velocitaMinima: double, velocitaMassima: double, costoMinimo: double, costoMassimo: double, valoreScatto: double, distanzaDiScatto: double)

ATTENZIONE alle unità di misura

- il tassametro misura i tempi in *secondi* e le distanze in *metri*
- ma le tariffe richiedono velocità espresse in *km/h* …!

Tariffa a distanza

- Scatta quando sono vere tutte le condizioni che seguono:
  - la velocità media dall'ultimo scatto è **maggiore o uguale** alla velocità minima
  - la velocità media dall'ultimo scatto è **minore** della velocità massima
  - il costo variabile corrente è **maggiore o uguale** al costo minimo
  - il costo variabile corrente è **minore** del costo massimo
  - lo spazio percorso dall'ultimo scatto ***ARROTONDATO*** (`Math.round`) è **maggiore o uguale** alla distanza di scatto
- In tal caso, produce e restituisce lo Scatto appropriato:

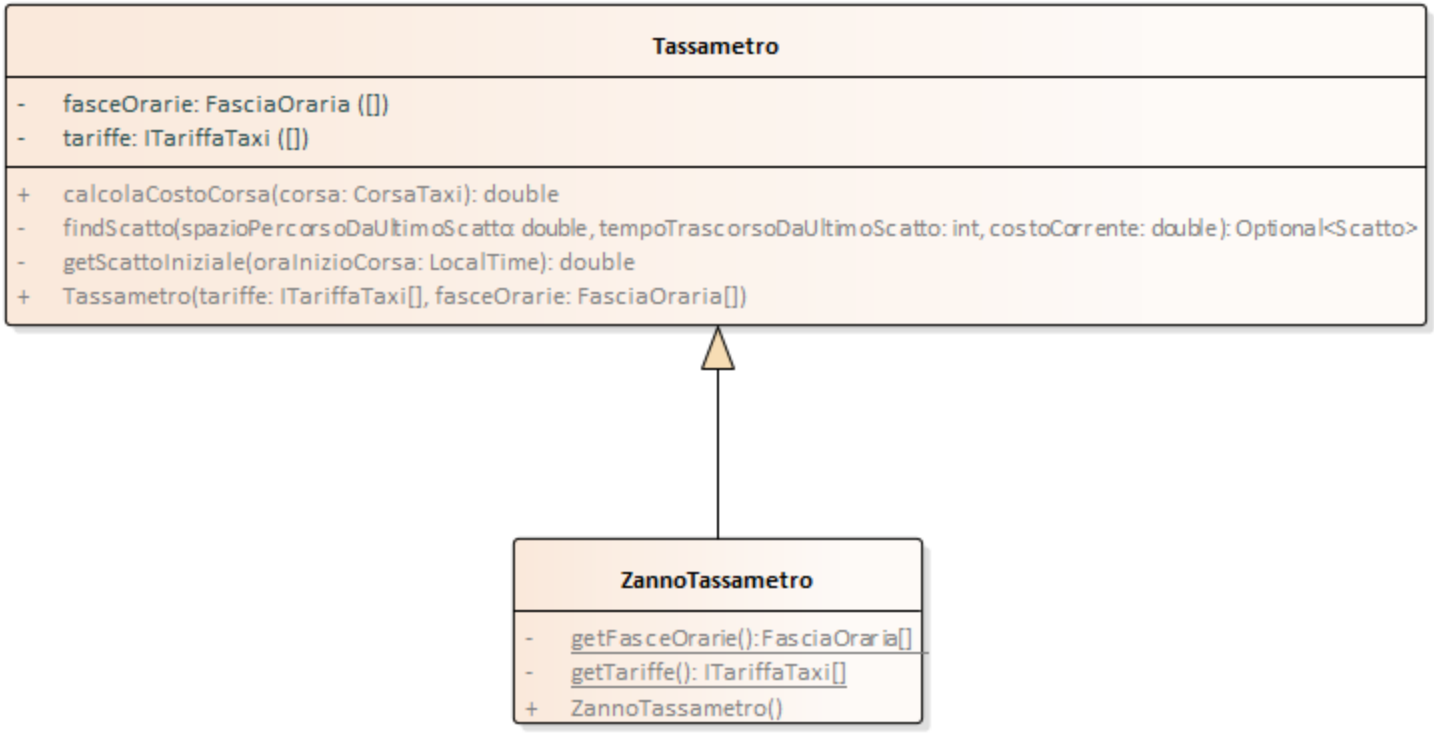
Indica se lo scatto s'ha da fare

```
boolean effettuaScatto = .....
return effettuaScatto
        ? Optional.of(new Scatto(tempoTrascorsoDaUltimoScatto, distanzaDiScatto, valoreScatto))
        : Optional.empty();
```

Da Tassametro a ZannoTassametro

- L'interfaccia **`ITassametro`** esprime il concetto generale
  - una entità in grado di calcolare il costo di una corsa di taxi
- La classe **`Tassametro`** cattura *la tipologia di tassametri basati su fasce orarie*
  - **`Tassametro implements ITassametro`**
  - concreta, ma ancora *da configurare* per uno specifico *set di tariffe*
- La classe **`ZannoTassametro`** è il Tassametro di Zann-O-Town
  - un **`Tassametro`** configurato in base alle *regole deliberate dal Consiglio comunale di Zann-O-Town*
  - **`ZannoTassametro extends Tassametro`**
    (e quindi indirettamente **`implements ITassametro`**)

ZannoTassametro

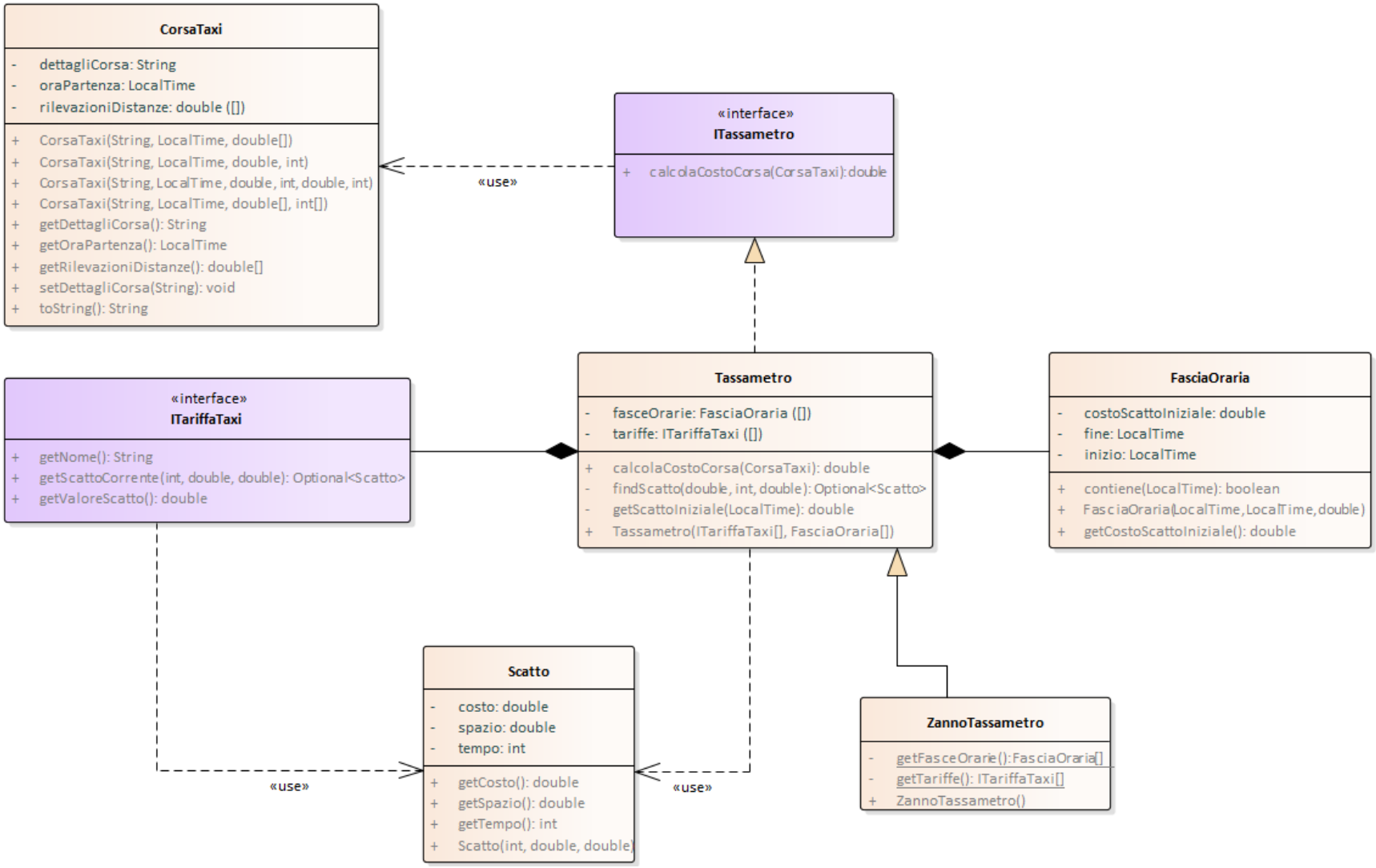


ZannoTassametro

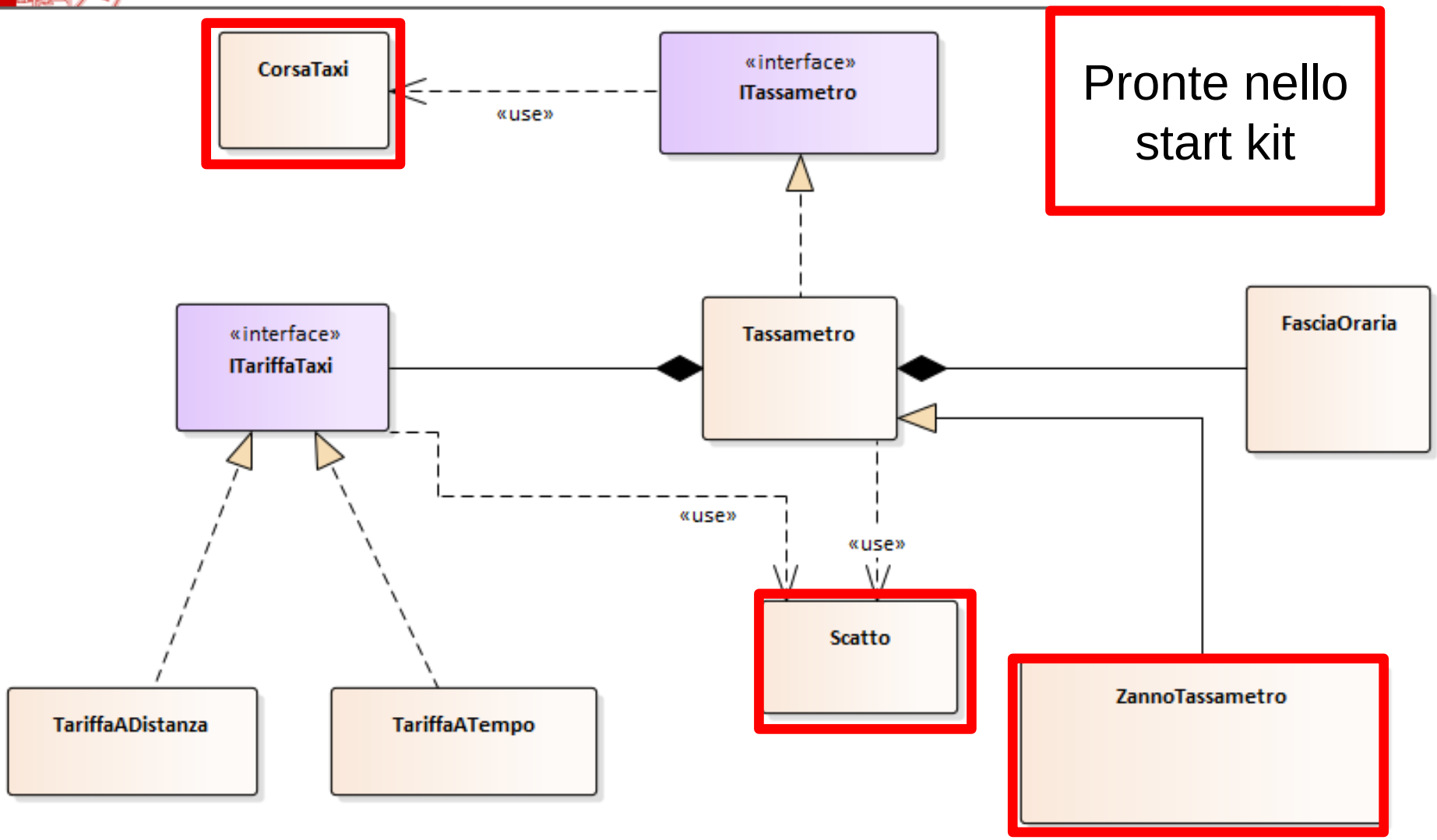
ZannoTassametro.java

```
2
3  import java.time.LocalTime;
4
5  public class ZannoTassametro extends Tassametro {
6
7      private static ITariffaTaxi[] getTariffe() {
8          return new ITariffaTaxi[] {
9                  new TariffaATempo("T0", 0, 27, 0.15, 12),
10                 new TariffaADistanza("T1", 27, Double.MAX_VALUE, 0, 10, 0.25, 100),
11                 new TariffaADistanza("T2", 27, Double.MAX_VALUE, 10, 25, 0.20, 85),
12                 new TariffaADistanza("T3", 27, Double.MAX_VALUE, 25, Double.MAX_VALUE, 0.15, 65)
13         };
14     }
15
16     private static FasciaOraria[] getFasceOrarie() {
17         return new FasciaOraria[] {
18                 new FasciaOraria(LocalTime.of(06, 00), LocalTime.of(21, 59), 4),
19                 new FasciaOraria(LocalTime.of(22, 00), LocalTime.of(23, 59), 6),
20                 new FasciaOraria(LocalTime.of(00, 00), LocalTime.of(05, 59), 6),
21                 };
22     }
23
24     public ZannoTassametro() {
25         super(getTariffe(), getFasceOrarie());
26     }
27
28 }
```

Modello quasi completo

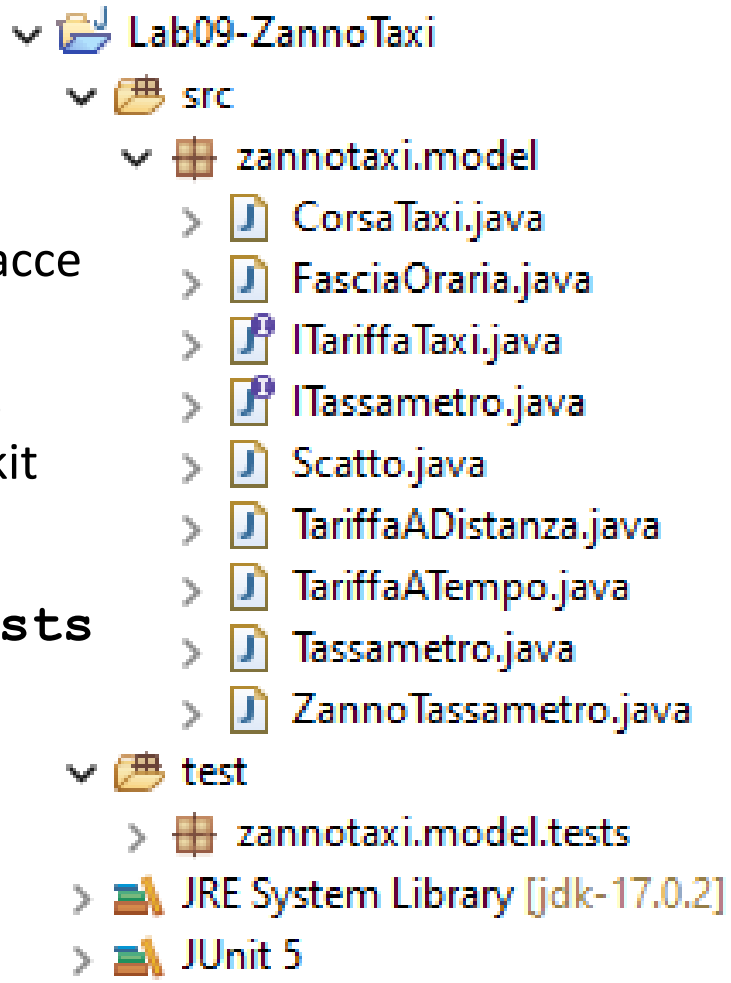


Modello completo



Il progetto Eclipse

- Due source folder:
  - **`src`**
    - package **`zannotaxi.model`**
    - qui vanno inserite le classi e le interfacce del modello
    - **`CorsaTaxi`**, **`ZannoTassametro`**, **`Scatto`** sono già pronte nello start kit
  - **`test`**
    - il package **`zannotaxi.model.tests`**
    - contiene i test, forniti già pronti



To Do

- Fornite già pronte nello Start kit
  - **`CorsaTaxi`**
  - **`ZannoTassametro`**
  - **`Scatto`**
- **Da realizzare**
  - Interfacce: **`ITassametro`**, **`ITariffaTaxi`**
  - Classi: **`FasciaOraria`**, **`Tassametro`**, **`TariffaADistanza`**, **`TariffaATempo`**
- Test
  - pronti nello Start kit

Hey!

KEEP
CALM
AND
HAPPY
CODING

And…