

Fondamenti di Informatica T2

Lab07 – il caso di studio Phone Plan

Corso di Laurea in Ingegneria Informatica

Anno accademico 2021/2022

Prof. ROBERTA CALEGARI

Prof. AMBRA MOLESINI

Dipartimento di Informatica – Scienza e Ingegneria (DISI)



Tariffazione Telefonica

OBIETTIVO

modellare un sistema di tariffazione telefonica

- Progettare il **piano telefonico**
- Dal piano telefonico si calcola il costo di una **chiamata** sulla base di:
 - istanti di inizio/fine chiamata
 - numero chiamato
(il costo può dipendere dall'operatore, dal paese di destinazione, etc.)
- Il numero chiamato determina la **tariffa** applicata
- La tariffa è basata su determinate **fasce orarie e giorni**

Parte 0: analisi del problema

Il dominio del problema

OBIETTIVO

modellare un sistema di tariffazione telefonica

- Progettare il **piano telefonico**
- Dal piano telefonico si calcola il costo di una **chiamata** sulla base di:
 - istanti di inizio/fine chiamata
 - numero chiamato
(il costo può dipendere dall'operatore, dal paese di destinazione, etc.)
- Il numero chiamato determina la **tariffa** applicata
- La tariffa è basata su determinate **fasce orarie e giorni**

Piano Telefonico

Entità da modellare

Chiamata

Entità da modellare

Tariffa

Entità da modellare

Banda

Entità da modellare



Analisi del dominio: Piano Telefonico

- Ogni piano telefonico stabilisce la **tariffa applicabile** a ogni possibile numero chiamato: ad esempio,
 - se chiami gli Stati Uniti di giorno spendi X
 - se chiami il tuo vicino di casa al fisso spendi Y
 - se chiami la tua «dolce metà» di notte spendi Z

```
Rate[] arrayTariffe = [TIMVersoAmoreMio, TIMVersoTIM, TIMVersoTutti];  
PhonePlan plan = new PhonePlan("TIM", arrayTariffe);  
PhoneCall call = new PhoneCall(start, end, "+3933912312312");  
double cost = plan.getCallCost(call);
```


Il piano telefonico riceve una chiamata e ne calcola il costo in base al numero chiamato, che stabilisce la tariffa da applicare.



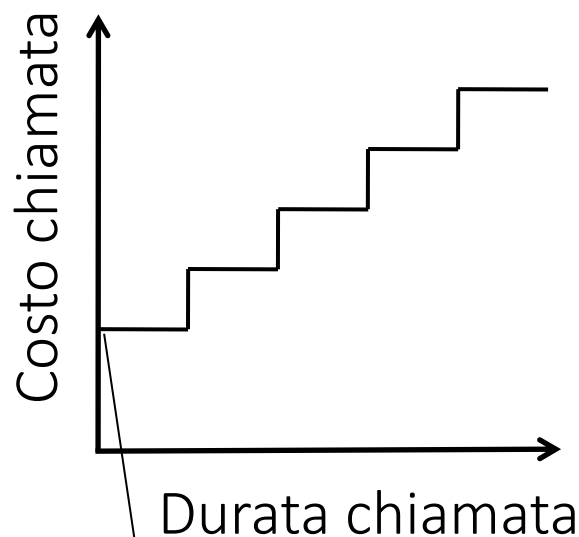
Analisi del dominio: Tariffa (1/3)

- Una **Tariffa** è caratterizzata da:
 - **Costo dello scatto alla risposta** (eventualmente zero)
 - **Durata dell'intervallo di tempo «standard»** (almeno in millisecondi)
 - in questo modo si modellano sia **tariffe a scatti**, sia **tariffe a secondi**
 - per avere una **tariffa a secondi** → impostare l'intervallo ad 1 secondo
 - per avere una **tariffa a scatti** → impostare l'intervallo alla durata dello scatto
 - **Costo dell'intervallo** che può **variare** secondo:
 - l'**ora** della chiamata (fascia oraria)
 - il **giorno** della chiamata (mercoledì, domenica, ...)

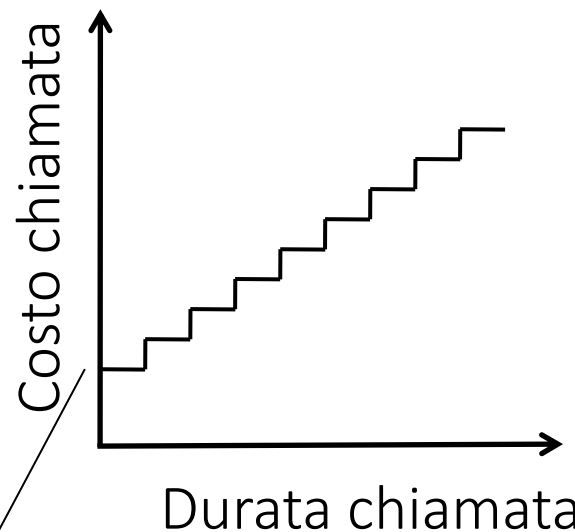
Analisi del dominio: Tariffa (2/3)

- Il numero chiamato determina la **tariffa** applicata
 - Italia, fisso / mobile
 - UE / USA / Canada, fisso / mobile
 - ...
- Tariffa a **secondi** o a **scatti**?
- Con o senza **scatto alla risposta**? 
- Con una **tariffa a scatti** in cui sia possibile calibrare **tempo di scatto** e relativo **costo** si riesce anche a realizzare una **tariffa a tempo**
 - Al secondo?
 - Al millisecondo?
- Sempre con **scatto alla risposta** eventualmente di costo **nullo**

Analisi del dominio: Tariffa (3/3)

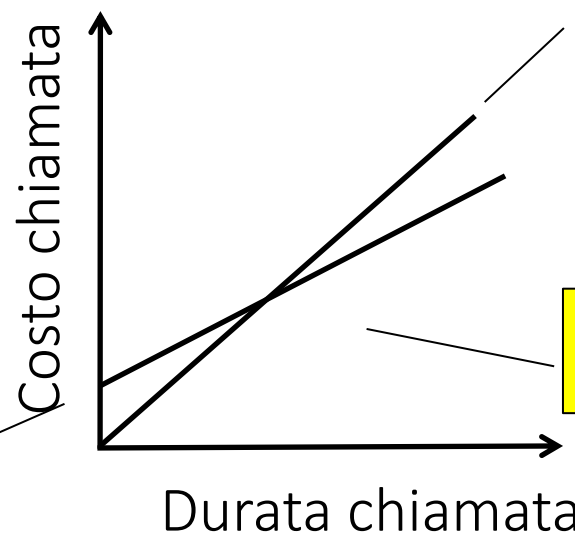


Scatto alla risposta più primo scatto



Scatto alla risposta nullo

Scatto alla risposta più primo scatto



Tempo di scatto molto piccolo

Analisi del dominio: Bande (1/2)

- La giornata è suddivisa in **fasce orarie (bande)**, caratterizzate da **ora di inizio** e **fine** validità e **costo per intervallo** di tempo
 - Una stessa fascia oraria potrebbe valere per più giorni della settimana, ad esempio:
 - dal lunedì al venerdì dalle 8:00 alle 18:00
 - dal lunedì al venerdì dalle 18:00 alle 24:00
 - sabato e domenica dalle 00:00 alle 24:00
- **Vincolo di coerenza:** per essere *valida*, una tariffa deve coprire completamente le 24h → **condizione di adiacenza**
 - non possono esserci buchi!
 - a qualunque orario si telefoni, ci dev'essere una qualche tariffa applicabile (non esiste che a un certo orario non si sappia che tariffa applicare..!)

Banda

Entità da modellare

Esistono davvero le
"ore 24" ?



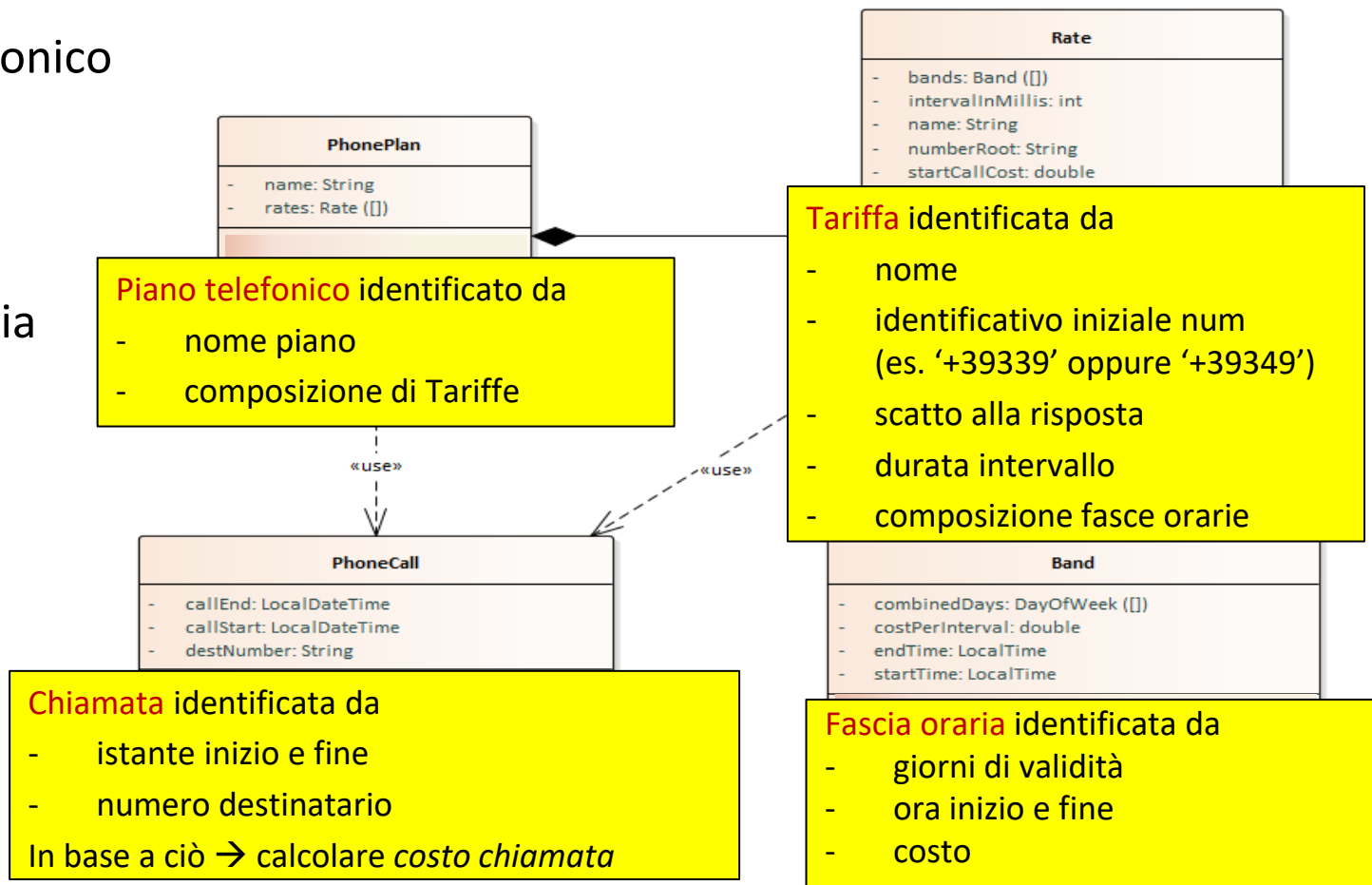
Analisi del dominio: Bande (2/2)

- **Un problema: la mezzanotte**
 - Abbiamo l'abitudine di dire "fino alle 24"..
 - **..MA le "ore 24" non esistono!**
 - infatti, neanche **LocalTime.of(24,0)** esiste!
- **In effetti, la giornata va dalle ore 0:00 alle ore 23:59**
 - dovremo ricordarcelo!
 - in particolare, NON potremo adottare intervalli *aperti a destra*...
 - es. "dalle 8 alle 18" inteso come $[8,18)$
 - .. perché poi non riusciremmo a rappresentare l'ultimo intervallo della giornata
 - infatti, "dalle 18 alle 24" sarebbe $[18, 24)$, *ma 24 non si può scrivere!*


Modello del dominio

- L'analisi precedente ha messo in evidenza le **entità in gioco**:

- Piano telefonico
- Chiamata
- Tariffa
- Fascia oraria



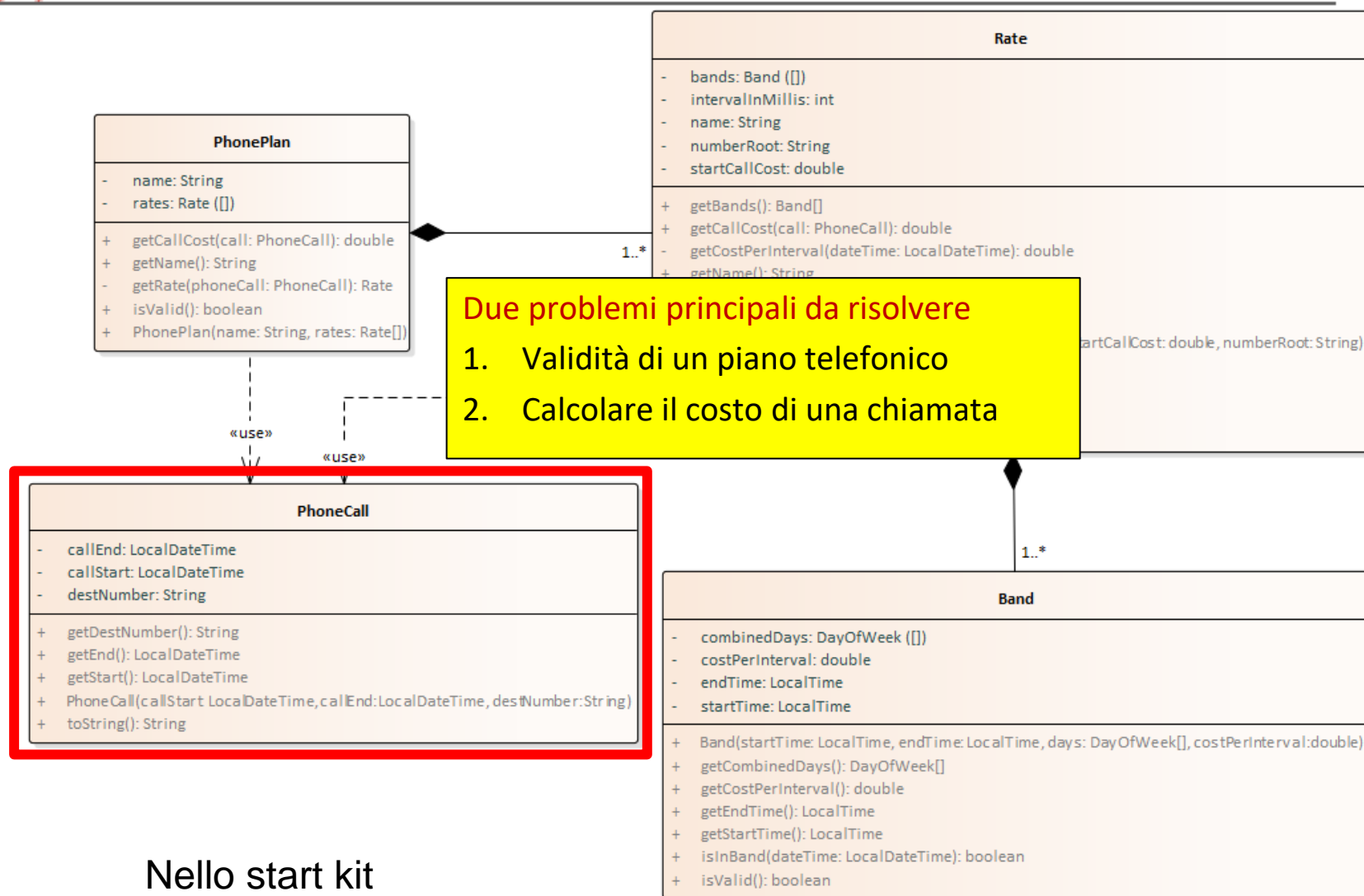
Modello del dominio

- L'analisi precedente ha messo in evidenza le **entità in gioco**:
 - Piano telefonico: **PhonePlan**
 - Chiamata: **PhoneCall** 
 - Tariffa: **Rate**
 - Fascia oraria: **Band**
- La descrizione ha già evidenziato alcune *caratteristiche generali* di tali entità, ma ora ci serve *un'analisi più dettagliata*.

Regola aurea:
più struttura = meno codice negli algoritmi

compliance by design

Modello del dominio

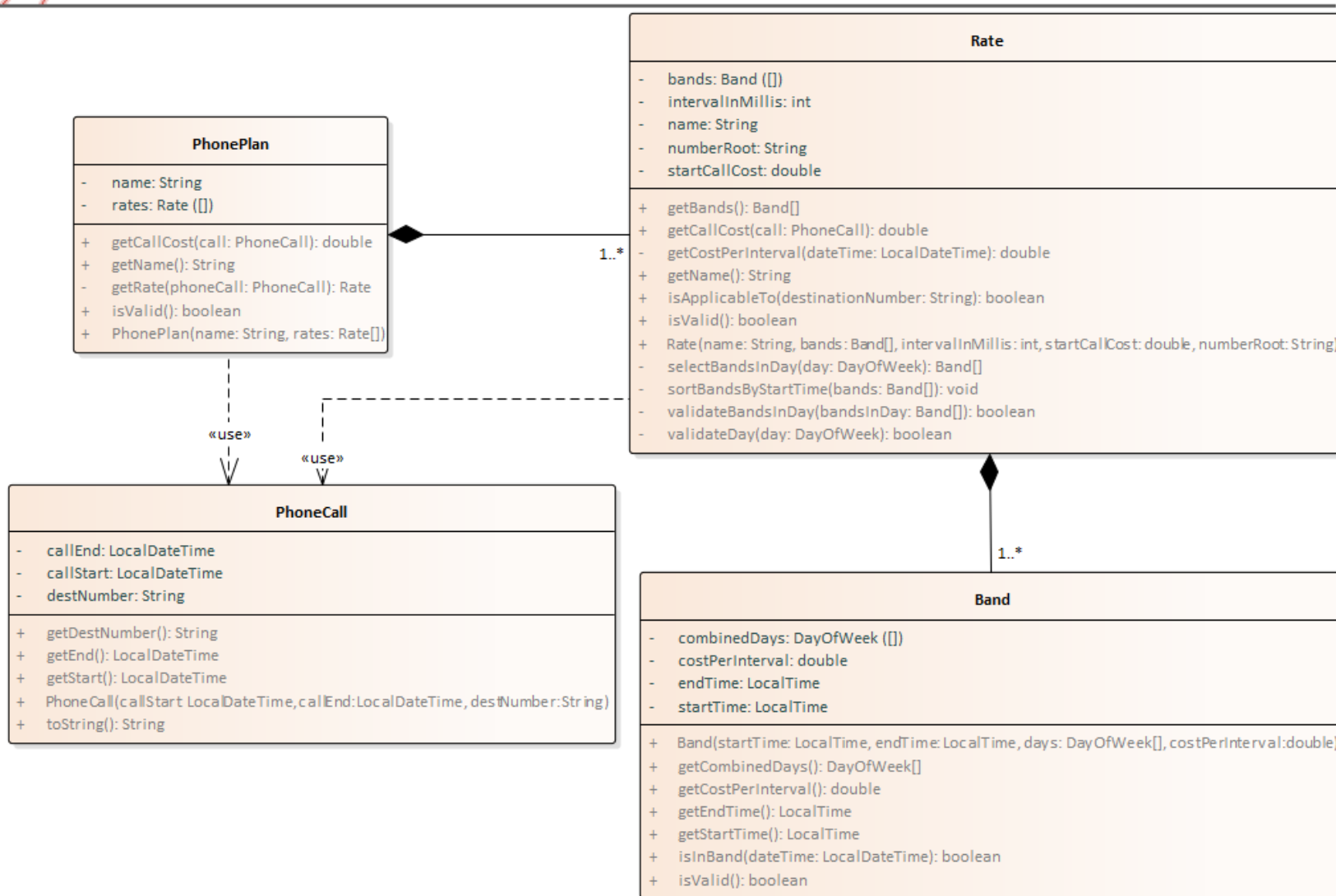


Parte I: modello ed entità in gioco

Modello delle entità

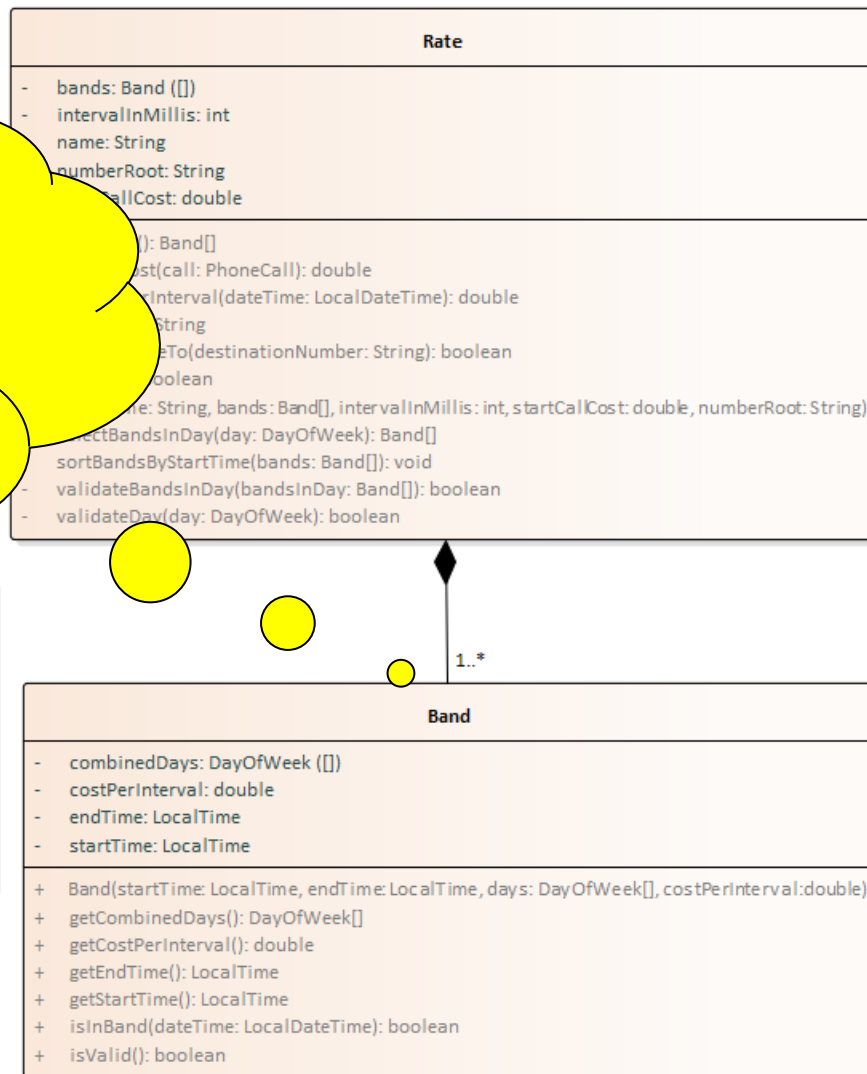
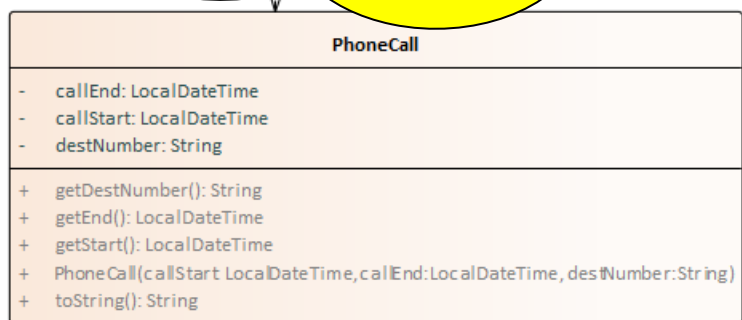
- Passiamo a modellare le entità del sistema, con approccio bottom-up
- Lasciamo gli algoritmi per la prossima volta
 - Fascia oraria: **Band**
 - Tariffa: **Rate**
 - Chiamata: **PhoneCall** Fornita già fatta nello start kit
 - Piano telefonico: **PhonePlan**

Modello del dominio



Modello del dominio

Vi ricordate le regola?
Parto ad implementare
dalla classe che ha
meno «legami», in
questo caso Band



Band (1/3)

- Modella la fascia oraria ed è caratterizzata da:
 - Ora di inizio/fine e giorni della settimana in cui è valida

LocalTime

DayOfWeek []

- Costo dell'intervallo
- i relativi metodi accessor **getXXX**

– ...

Band
<ul style="list-style-type: none">- combinedDays: DayOfWeek ([])- costPerInterval: double- endTime: LocalTime- startTime: LocalTime
<ul style="list-style-type: none">+ Band(startTime: LocalTime, endTime: LocalTime, days: DayOfWeek[], costPerInterval:double)+ getCombinedDays(): DayOfWeek[]+ getCostPerInterval(): double+ getEndTime(): LocalTime+ getStartTime(): LocalTime

Band (2/3)

- Modella la fascia oraria ed è caratterizzata da:
 - ...
 - Un metodo (**isInBand**) che consente di **verificare se la fascia è applicabile** per una **data&ora** passate come parametro
 - dalla data passata in ingresso devo ricavare il giorno della settimana e il LocalTime associato
 - verifico che il giorno trovato sia tra i giorni in cui la banda è attiva
 - verifico che l'orario ricevuto cada all'interno dell'orario coperto dalla banda

LocalDateTime

Band	
-	combinedDays: DayOfWeek ([])
-	costPerInterval: double
-	endTime: LocalTime
-	startTime: LocalTime
<hr/>	
+	Band(startTime: LocalTime, endTime: LocalTime, days: DayOfWeek[], costPerInterval:double)
+	getCombinedDays(): DayOfWeek[]
+	getCostPerInterval(): double
+	getEndTime(): LocalTime
+	getStartTime(): LocalTime
+	isInBand(dateTime: LocalDateTime): boolean
+	isValid(): boolean

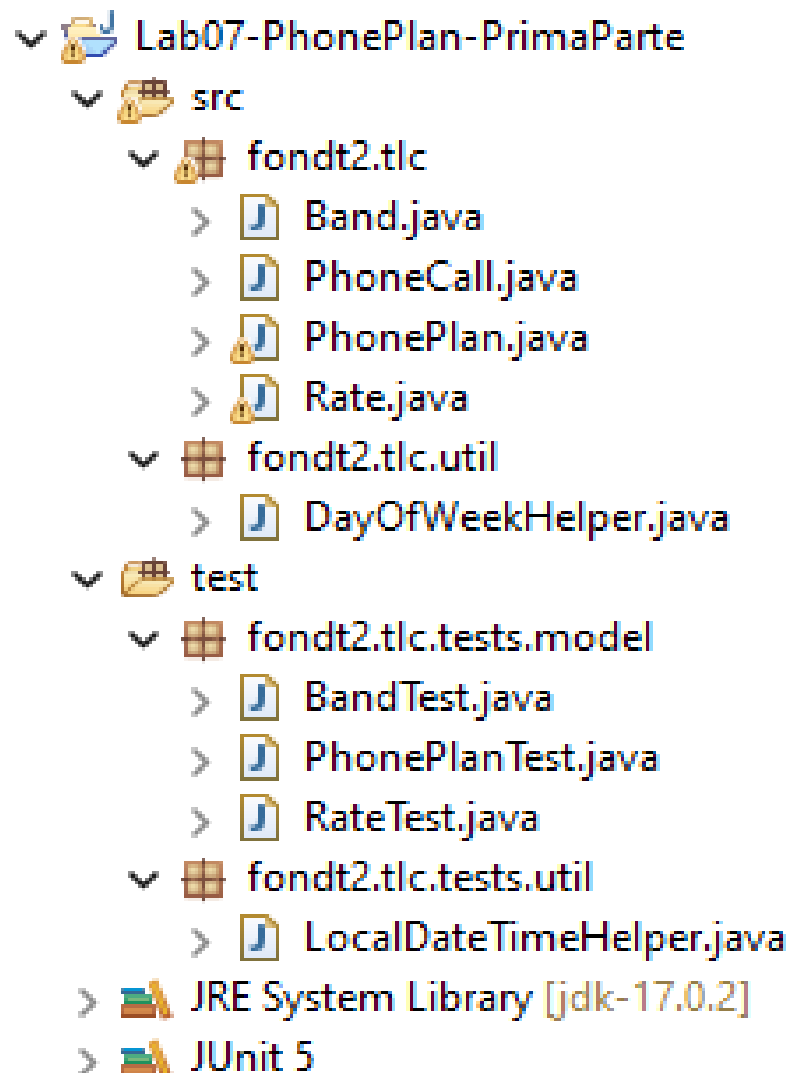
Band (3/3)

- Un metodo (**isValid**) che consente di **verificare se la fascia è valida...**
 - **La fascia è valida se**
 - l'ora di inizio precede l'ora di fine
 - l'insieme dei giorni di applicabilità non è vuoto
 - il costo per intervallo non è negativo

Band	
-	combinedDays: DayOfWeek []
-	costPerInterval: double
-	endTime: LocalTime
-	startTime: LocalTime
<hr/>	
+	Band(startTime: LocalTime, endTime: LocalTime, days: DayOfWeek[], costPerInterval: double)
+	getCombinedDays(): DayOfWeek[]
+	getCostPerInterval(): double
+	getEndTime(): LocalTime
+	getStartTime(): LocalTime
+	isInBand(dateTime: LocalDateTime): boolean
+	isValid(): boolean

Start Kit

- **Nello start kit ci sono:**
 - il progetto già fatto
 - la classe **PhoneCall**
 - Nel package **util** troverete **DayOfWeekHelper**.. usatelo 😊
 - Test già pronti nel source folder «test»



Band

Band	
-	<code>combinedDays: DayOfWeek []</code>
-	<code>costPerInterval: double</code>
-	<code>endTime: LocalTime</code>
-	<code>startTime: LocalTime</code>
+	<code>Band(LocalTime, LocalTime, DayOfWeek[], double)</code>
+	<code>getCombinedDays(): DayOfWeek[]</code>
+	<code>getCostPerInterval(): double</code>
+	<code>getEndTime(): LocalTime</code>
+	<code>getStartTime(): LocalTime</code>
+	<code>isInBand(LocalDateTime): boolean</code>
+	<code>isValid(): boolean</code>

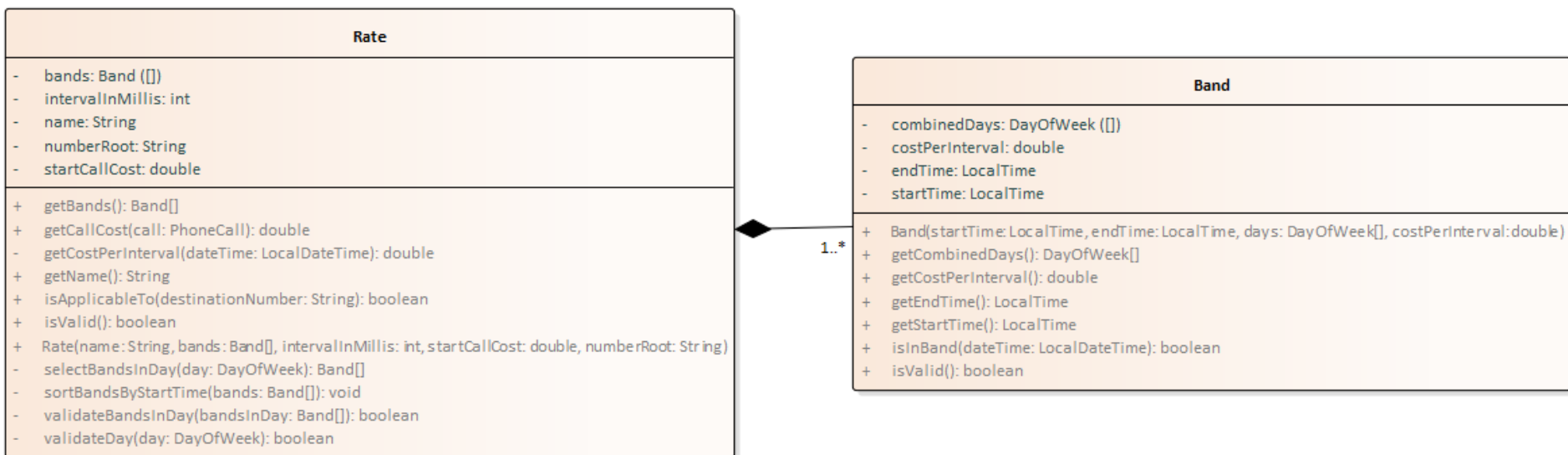
Tempo a disposizione: 45 minuti

Nello start kit troverete la classe `DayOfWeekHelper`: usatela!

Rate

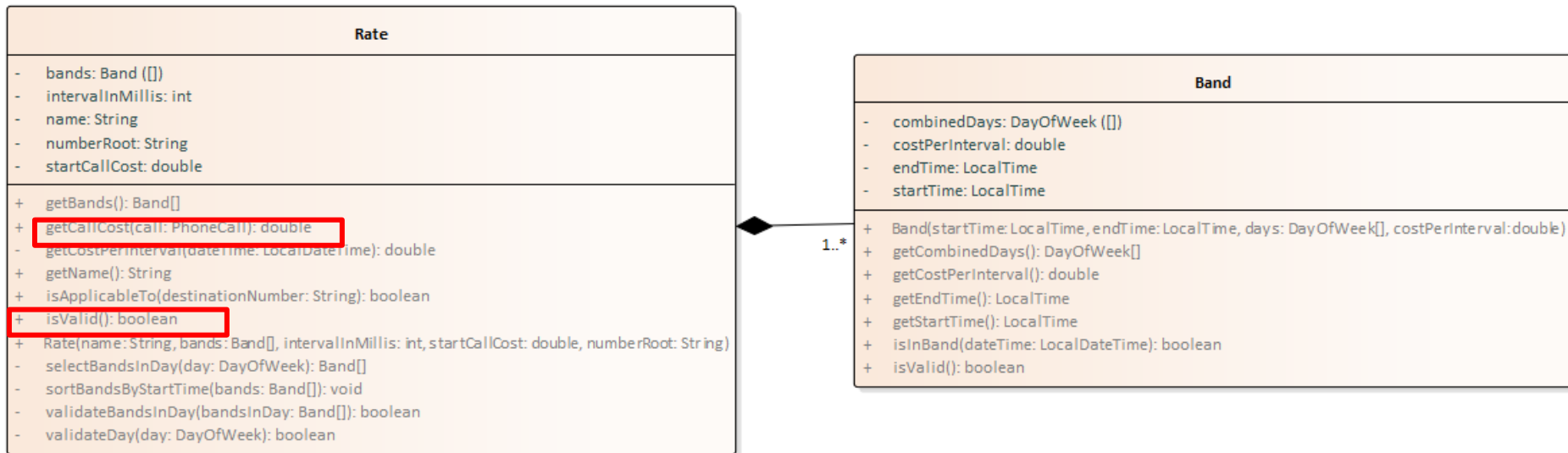
- Una **tariffa** è caratterizzata dalle seguenti proprietà:
 - un nome (ad es., «Zona EU», «Italia Notte», ...)
 - l'inizio del numero destinatario
 - l'insieme delle **fasce (Band)** applicabili
 - la durata dell'intervallo
 - il costo dello scatto alla risposta
- e deve consentire di:
 - sapere se è applicabile ad un certo numero (**isApplicableTo**)
 - ottenere il costo di una chiamata (**getCallCost**)
 - verificare la validità/consistenza della tariffa (**isValid**), cosa che richiede il rispetto di *due vincoli*:
 - a) che le fasce siano **singolarmente valide**
 - b) che, tutte insieme, **coprano tutta la settimana senza buchi**

Rate



NB: il vincolo che le bande debbano *coprire interamente* la giornata non è catturato dal diagramma di struttura UML, *va espresso a parte a parole*

Rate



Essenziale: dall'interfaccia di Rate *non emergono le scelte interne*.
 In particolare, **non emerge nulla riguardo alla suddivisione in bande** 😊
 Ottimo: il cliente non lo sa e non deve saperlo!

- Per questo, *occhio agli accessor* !
 - OK restituire il nome → **getName**
 - PRUDENZA a restituire l'elenco delle bande: OK SOLO SCOPO DI TEST → **getBands**

Rate

Rate
<ul style="list-style-type: none">- bands: Band ([])- intervalInMillis: int- name: String- numberRoot: String- startCallCost: double
<ul style="list-style-type: none">+ getBands(): Band[]+ getCallCost(call: PhoneCall): double- getCostPerInterval(dateTime: LocalDateTime): double+ getName(): String+ isApplicableTo(destinationNumber: String): boolean+ isValid(): boolean+ Rate(name: String, bands: Band[], intervalInMillis: int, startCallCost: double, numberRoot: String)- selectBandsInDay(day: DayOfWeek): Band[]- sortBandsByStartTime(bands: Band[]): void- validateBandsInDay(bandsInDay: Band[]): boolean- validateDay(day: DayOfWeek): boolean

Tempo a disposizione: 35 minuti *esclusi i due metodi `isValid` e `getCallCost`*, da sviluppare dopo.

Nello start kit troverete la classe `DayOfWeekHelper`: usatela!

PhoneCall

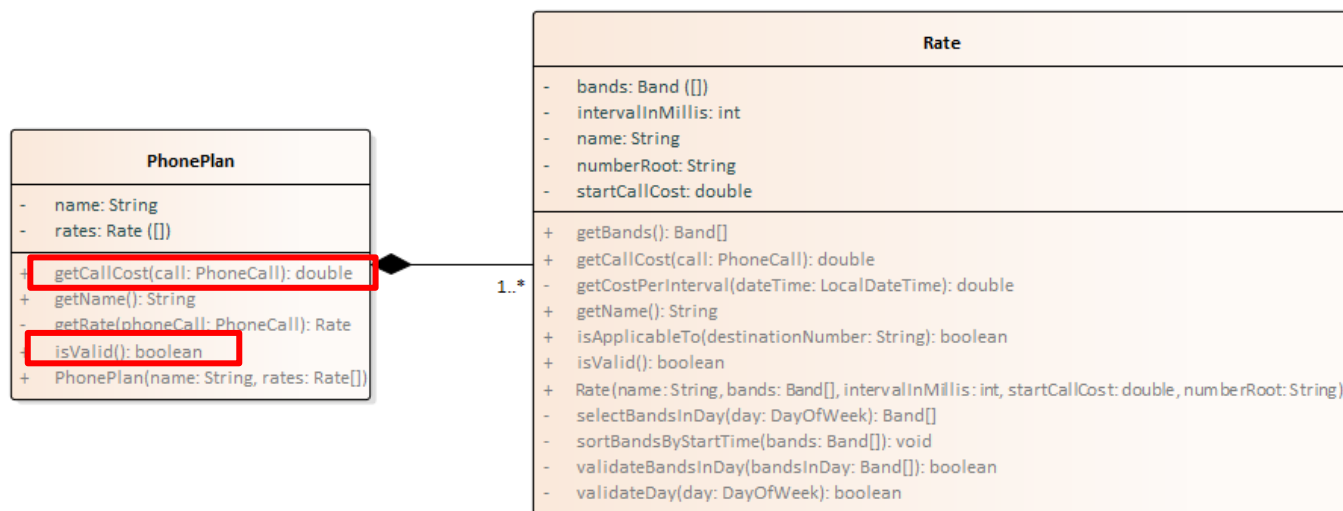
- È fornita già pronta nello start kit!
- Rappresenta una chiamata con tutte le sue caratteristiche
 - Data/Ora di inizio
 - Data/Ora di fine
 - Numero Chiamato
- Ovvio costruttore, ovvi accessor, ovvia **toString**

PhoneCall	
-	<code>callEnd: LocalDateTime</code>
-	<code>callStart: LocalDateTime</code>
-	<code>destNumber: String</code>
+	<code>getDestNumber(): String</code>
+	<code>getEnd(): LocalDateTime</code>
+	<code>getStart(): LocalDateTime</code>
+	<code>PhoneCall(callStart: LocalDateTime, callEnd: LocalDateTime, destNumber: String)</code>
+	<code>toString(): String</code>

PhonePlan

- Il piano telefonico è caratterizzato da:
 - nome del piano
 - insieme di tariffe (tutte quelle possibili?)
- e consente di:
 - ottenere il costo di una chiamata
 - verificare se è valido

(un piano è valido se ogni sua tariffa lo è)

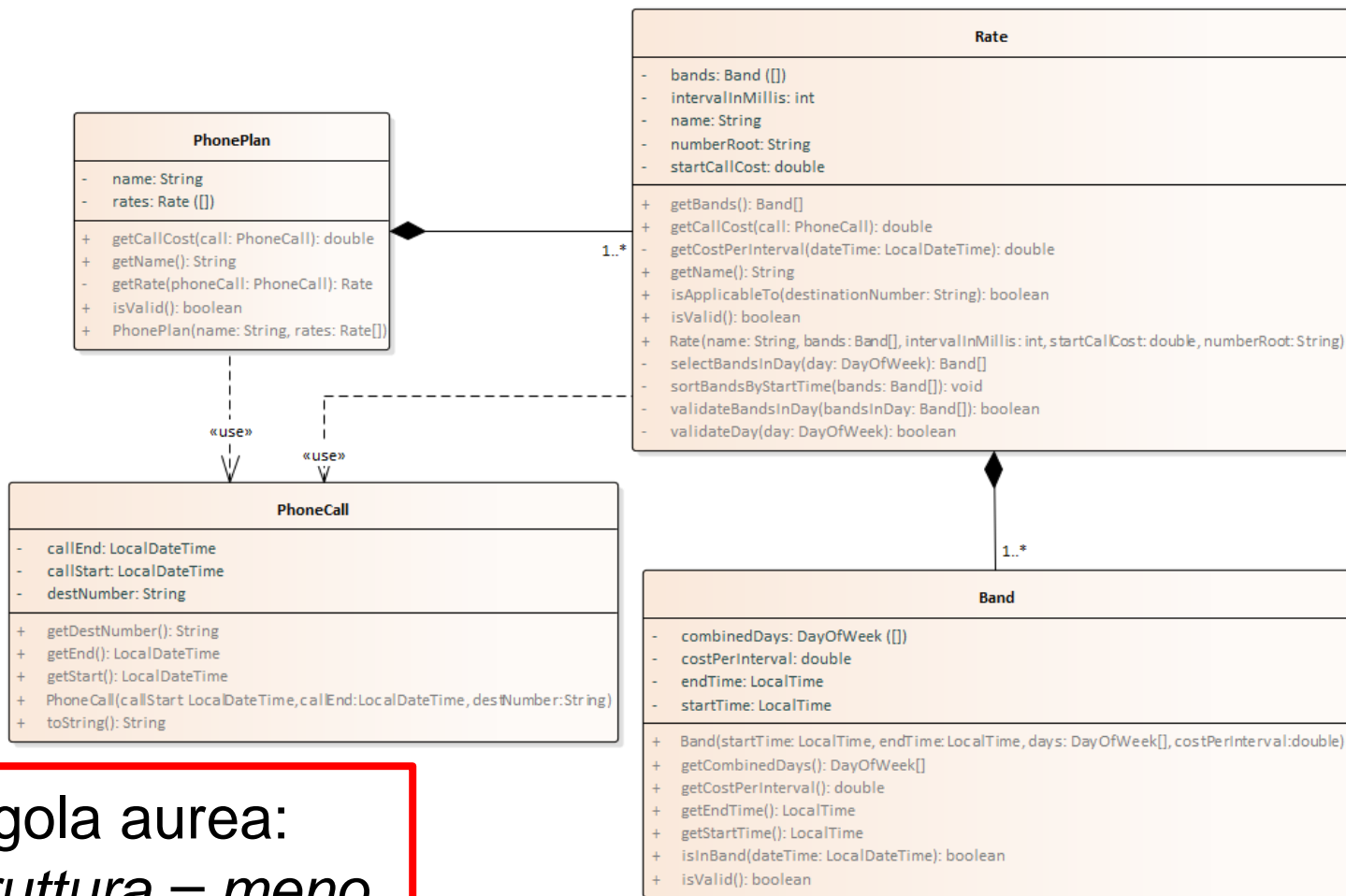


PhonePlan

PhonePlan	
-	name: String
-	rates: Rate ([])
+	getCallCost(call: PhoneCall): double
+	getName(): String
-	getRate(phoneCall: PhoneCall): Rate
+	isValid(): boolean
+	PhonePlan(name: String, rates: Rate[])

Tempo a disposizione: 25 minuti *esclusi i due metodi `isValid` e `getCallCost`, da sviluppare dopo.*

Modello completo



Regola aurea:
*più struttura = meno
 codice negli algoritmi*