

## Fondamenti di Informatica T2

### Lab05 – Ticket Sosta

*Corso di Laurea in Ingegneria Informatica*

Anno accademico 2021/2022

Prof. ROBERTA CALEGARI

Prof. AMBRA MOLESINI

*Dipartimento di Informatica – Scienza e Ingegneria (DISI)*



# Agenda

---

- Lavoriamo con package `java.time`
- Prima parte: **Ticket Sosta**
  - caso di studio estratto da un compito d'esame di qualche anno fa (Luglio 2018)
- Seconda parte: **Refactoring**
  - cosa succede se cambiano i requisiti?

# Ticket Sosta: Problema

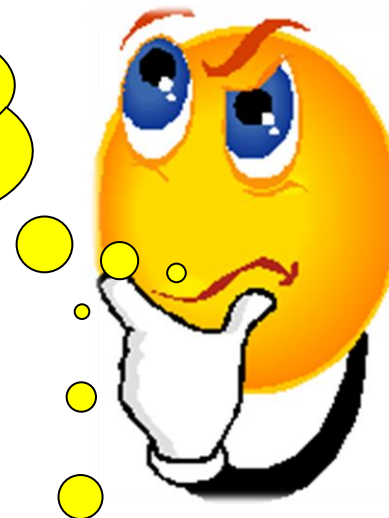
- Per sostare sulle «strisce blu» bisogna pagare
- Il parcometro rilascia un ticket che rappresenta l'importo che è stato pagato e il periodo di validità della sosta
- Ogni zona della città ha la sua *tariffa*
  - *costo orario* *esempio: 0,50 €/ora*
  - *sosta minima* *esempio: si paga sempre minimo un'ora*
  - *eventuale franchigia* *esempio: i primi 20 minuti sono gratuiti*



# Analisi del Problema



Da dove parto a capire  
come è fatto il mio  
sistema?  
**OSSERVO IL DOMINIO!**



Questo dominio è  
molto vasto e  
«complicato»...!

# Analisi del Problema

- IPOTESI SEMPLIFICATIVA 1:  
la sosta non può estendersi su più giorni
  - in realtà potrebbe, anzi è la norma
  - MA allora dovremmo considerare i periodi in cui *non* si paga (di notte, prima di una certa ora, etc)
- IPOTESI SEMPLIFICATIVA 2:  
si paga sempre, a qualunque ora (anche di notte)



# Analisi del Problema

- Per sostare sulle «strisce blu» bisogna pagare
- Il **parcometro** **rilascia** un **ticket** che rappresenta l'**importo** che è stato pagato e **il periodo di validità** della sosta
- Ogni zona della città ha la sua **tariffa**
  - **costo orario** *esempio: 0,50 €/ora*
  - **sosta minima** *esempio: si paga sempre minimo un'ora*
  - **eventuale franchigia** *esempio: i primi 20 minuti sono gratuiti*

# Modello Dei Dati

- Le entità in gioco
  - **Tariffa**: rappresenta una data tariffa
    - PROPRIETÀ: nome, costo orario, durata minima, franchigia
  - **Ticket**: rappresenta il biglietto emesso dal parcometro
    - PROPRIETÀ: ora iniziale, ora finale, costo (cifra pagata)
  - **Parcometro**: rappresenta la macchinetta che emette ticket
    - PROPRIETÀ: la **tariffa** valida in quella zona della città
    - incapsula l'algoritmo di calcolo
- Algoritmi
  - Come si calcola il costo della sosta?



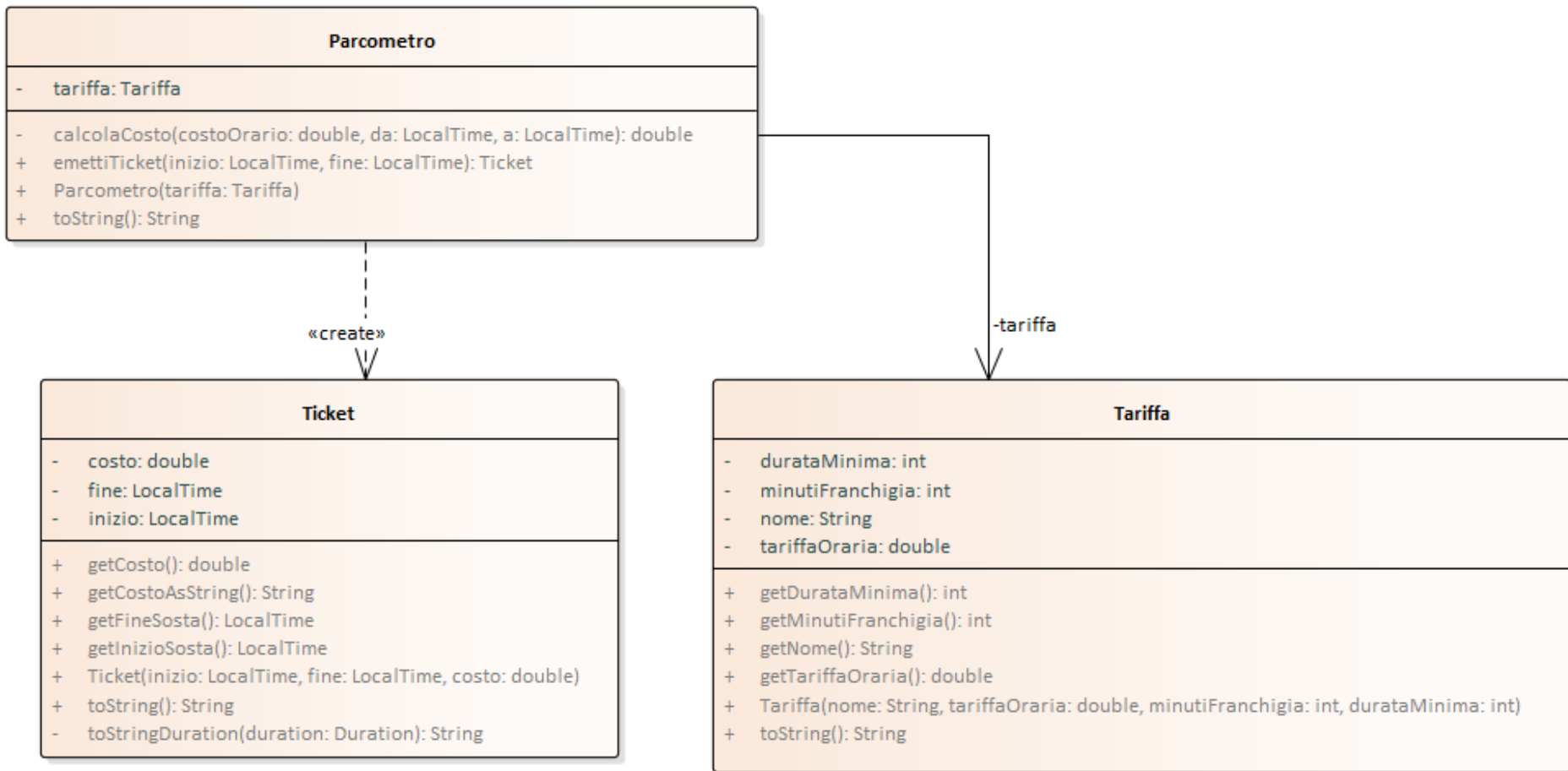
# Algoritmo di Calcolo

- Parametri di ingresso
  - la **tariffa** valida in quella certa zona → *ce l'ha già il parcometro*
  - l'**orario iniziale** e **finale** della sosta → *li specifica il cliente*
- Algoritmo di calcolo
  - Caso base: no franchigia, no minimo  
→  $\text{costo} = \text{costo orario} * \text{durata}$
  - Caso con franchigia  
→ sottrarre franchigia da durata  
→ poi, come nel caso base
  - Caso con durata minima  
→ se durata inferiore al minimo, considerare il minimo  
→ altrimenti, come nel caso base















# Architettura Software



# Struttura del Progetto

- ▼  Lab05-TicketSosta
  - ▼  src
    - ▼  ticketsosta
      - >  Parcometro.java
      - >  Tariffa.java
      - >  Ticket.java
  - ▼  tests
    - >  ticketsosta.test
- >  JRE System Library [jdk-17.0.2]
- >  JUnit 5

## DUE CARTELLE SORGENTI

- src → sorgenti
- tests → test junit

## Nello Startkit troverete:

- Il progetto Eclipse
- la cartella dei test già pronti
- La classe **Tariffa**

# Implementazione: Ticket

È una pura classe-dati  
SUGGERIMENTO: scrivere le proprietà  
e far generare tutto il resto a Eclipse! 😊😊

```
public class Ticket {
    private LocalTime inizio, fine;
    private double costo;

    public Ticket(LocalTime inizio, LocalTime fine, double costo) {
        super();
        this.inizio = inizio;
        this.fine = fine;
        this.costo = costo;
    }

    public LocalTime getInizioSosta() {
        return inizio;
    }

    public LocalTime getFineSosta() {
        return fine;
    }

    public double getCosto() {
        return costo;
    }

    public String getCostoAsString() {
        NumberFormat formatter = NumberFormat.getCurrencyInstance(Locale.ITALY);
        // NB: lo spazio prima/dopo il simbolo di valuta è il non-breakable space
        // (codice 160=0xA0), NON lo spazio classico (codice 32=0x20)
        return formatter.format(costo);
    }
}
```

Una sorta di «toString bis»  
per il costo

```
@Override
public String toString() {
    return "Sosta autorizzata\n" +
        inizio.format(DateTimeFormatter.ofLocalizedTime(FormatStyle.SHORT).withLocale(Locale.ITALY)) +
        " alle " +
        fine.format(DateTimeFormatter.ofLocalizedTime(FormatStyle.SHORT).withLocale(Locale.ITALY)) +
        "\n" +
        "Durata totale: " + toStringDuration(Duration.between(inizio, fine)) +
        "\n" +
        "Totale pagato: " + getCostoAsString();
}

private String toStringDuration(Duration duration) {
    int minuti = duration.toMinutesPart();
    String sMinuti = (minuti < 10 ? "0" : "") + minuti;
    return duration.toHours() + ":" + sMinuti;
}
```

Personalizziamo  
toString

Una sorta di «toString  
bis» per la durata

OCCHIO: il formattatore Italia produce «3,50 €»  
ma lo spazio prima di «€» non è quello classico  
È il «non breakable space» !

# Implementazione: Parcometro

```
public class Parcometro {  
  
    private Tariffa tariffa;  
  
    public Parcometro(Tariffa tariffa) {  
        this.tariffa = tariffa;  
    }  
  
    public Ticket emettiTicket(LocalTime inizio, LocalTime fine) {  
  
        // calcola costo e restituisci ticket  
    }  
  
    private double calcolaCosto(double costoOrario, LocalTime da, LocalTime a) {  
        Duration durataSosta;  
  
        // calcola e restituisci il costo della sosta  
    }  
  
    @Override  
    public String toString() {  
        return "Parcometro configurato con la tariffa " + tariffa.toString();  
    }  
}
```

**SUGGERIMENTO**: nel metodo `calcolaCosto` chiedetevi se ci sono dei casi limite (cosa succede intorno a mezzanotte?)

```
if (a.isBefore(da) || LocalTime.of(0, 0).equals(a)){...}
```



# Ricordate: `java.time`

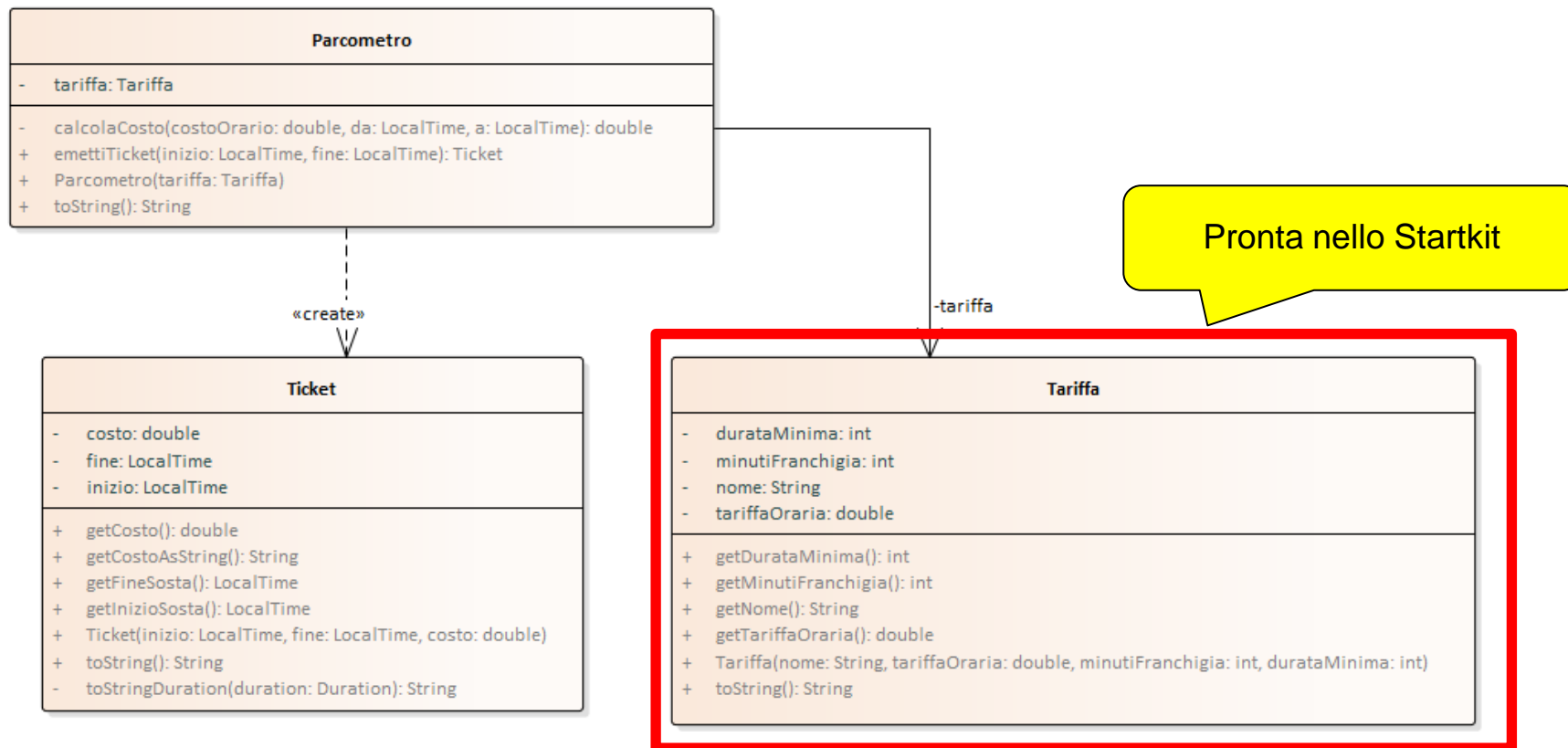
- **Concetti *relativi* (*locali*)**

- **LocalDate**: una data relativa (giorno/mese/anno)
- **LocalTime**: un orario relativo (ore/minuti/secondi)
- **LocalDateTime**: una data + orario relativi
- **Period**: una durata relativa (misurata in giorni, mesi, anni, etc.)

- **Concetti *assoluti***

- **Instant**: un punto sulla linea del tempo *espresso in nanosecondi*
- **Duration**: una durata in (nano)secondi *fra due istanti di tempo*
- **OffsetDateTime**: una data assoluta sulla linea del tempo *espressa come data + orario + delta rispetto a Greenwich (UTC)*
- **ZonedDateTime**: una data assoluta sulla linea del tempo *espressa come data + orario + fuso orario (es. CET, GMT-5, ecc.)*

# Architettura Software



**Tempo a disposizione: 45 minuti**

# SECONDA PARTE

## REFACTORING

# Nuovi requisiti

- Il committente ci ha chiesto le seguenti modifiche ai requisiti
  - possibilità di avere soste più **lunghe di una giornata**
  - possibilità di avere **tariffe diverse nei diversi giorni** della settimana mantenendo l'ipotesi semplificativa di avere la medesima tariffa per tutta la giornata
- MA il nostro progetto iniziale si è basato su un caso particolare di Parcometro: non catturiamo questi requisiti! ☹️
- Perciò, dobbiamo riprendere il progetto e valutare l'impatto dei nuovi requisiti sulla struttura delle classi



# Nuovi requisiti: Tariffa

Tariffa
<ul style="list-style-type: none"><li>- durataMinima: int</li><li>- minutiFranchigia: int</li><li>- nome: String</li><li>- tariffaOraria: double</li></ul>
<ul style="list-style-type: none"><li>+ getDurataMinima(): int</li><li>+ getMinutiFranchigia(): int</li><li>+ getNome(): String</li><li>+ getTariffaOraria(): double</li><li>+ Tariffa(nome: String, tariffaOraria: double, minutiFranchigia: int, durataMinima: int)</li><li>+ toString(): String</li></ul>

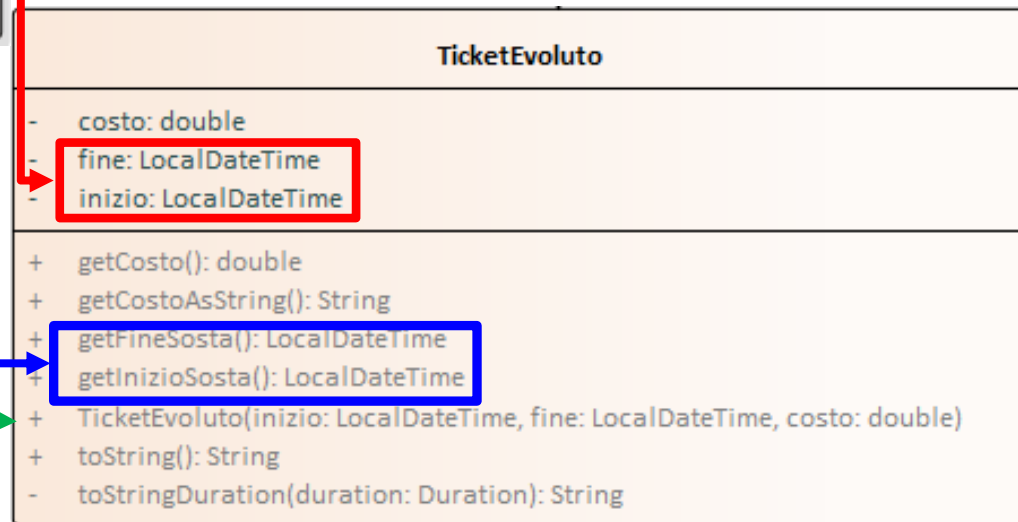
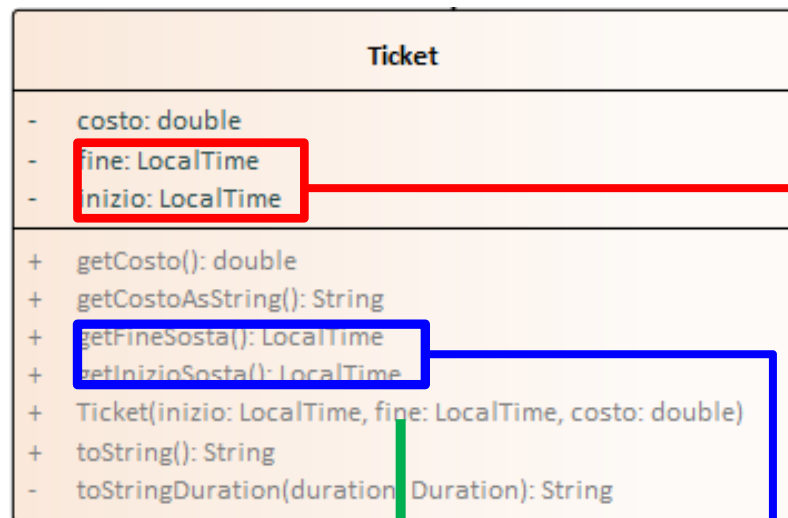
- La classe **Tariffa** *non deve essere modificata*
  - è una pura classe dati
  - non ha «riferimenti temporali» al suo interno

# Nuovi requisiti: Ticket

Ticket	
- costo: double	
- fine: LocalTime	←
- inizio: LocalTime	←
<hr/>	
+ getCosto(): double	
+ getCostoAsString(): String	
+ getFineSosta(): LocalTime	←
+ getInizioSosta(): LocalTime	←
+ Ticket(inizio: LocalTime, fine: LocalTime, costo: double)	
+ toString(): String	↑ ↑
- toStringDuration(duration: Duration): String	

- La classe **Ticket** *deve essere modificata*
  - ha «riferimenti temporali» al suo interno
  - **LocalTime** reca solo informazioni di «orario», nel caso di soste su più giorni dobbiamo poter indicare la data completa di inizio e fine della sosta  
→ rimpiazzato da **LocalDateTime**

# Da Ticket a TicketEvoluto



# Nuovi requisiti: Parcometro

Parcometro	
-	tariffa: Tariffa ←
-	calcolaCosto(costoOrario: double, da: LocalTime, a: LocalTime): double ←
+	emettiTicket(inizio: LocalTime, fine: LocalTime): Ticket ←
+	Parcometro(tariffa: Tariffa)
+	toString(): String

- La classe **Parcometro** *deve essere modificata*
    - ha «riferimenti temporali» al suo interno
    - deve poter contenere più **Tariffe**
- rivedere l'algoritmo di calcolo



# ParcometroEvoluto

- La classe **ParcometroEvoluto**
  - ha «riferimenti temporali» al suo interno di tipo **LocalDateTime**
  - deve poter contenere **più di una Tariffa**
    - **array di Tariffa**: la posizione 0 conterrà la tariffa del Lunedì, la posizione 1 quella del Martedì, etc.
  - va rivisto l'algoritmo di calcolo
    - **calcolaCosto** può essere riusata
    - MA occorre aggiungere una parte algoritmica per
      - a) *trovare la Tariffa corretta*
      - b) *considerare più giorni*→ **calcolaCostoSuPiuGiorni**

# Da Parcometro a ParcometroEvoluto

Parcometro	
-	<code>tariffa: Tariffa</code>
-	<code>calcolaCosto(costoOrario: double, da: LocalTime, a: LocalTime): double</code>
+	<code>emettiTicket(inizio: LocalTime, fine: LocalTime): Ticket</code>
+	<code>Parcometro(tariffa: Tariffa)</code>
+	<code>toString(): String</code>

ParcometroEvoluto	
-	<code>tariffe: Tariffa ([])</code>
-	<code>calcolaCosto(costoOrario: double, da: LocalTime, a: LocalTime): double</code>
-	<code>calcolaCostoSuPiuGiorni(inizio: LocalDateTime, fine: LocalDateTime): double</code>
+	<code>emettiTicket(inizio: LocalDateTime, fine: LocalDateTime): TicketEvoluto</code>
+	<code>ParcometroEvoluto(tariffe: Tariffa[])</code>
+	<code>toString(): String</code>

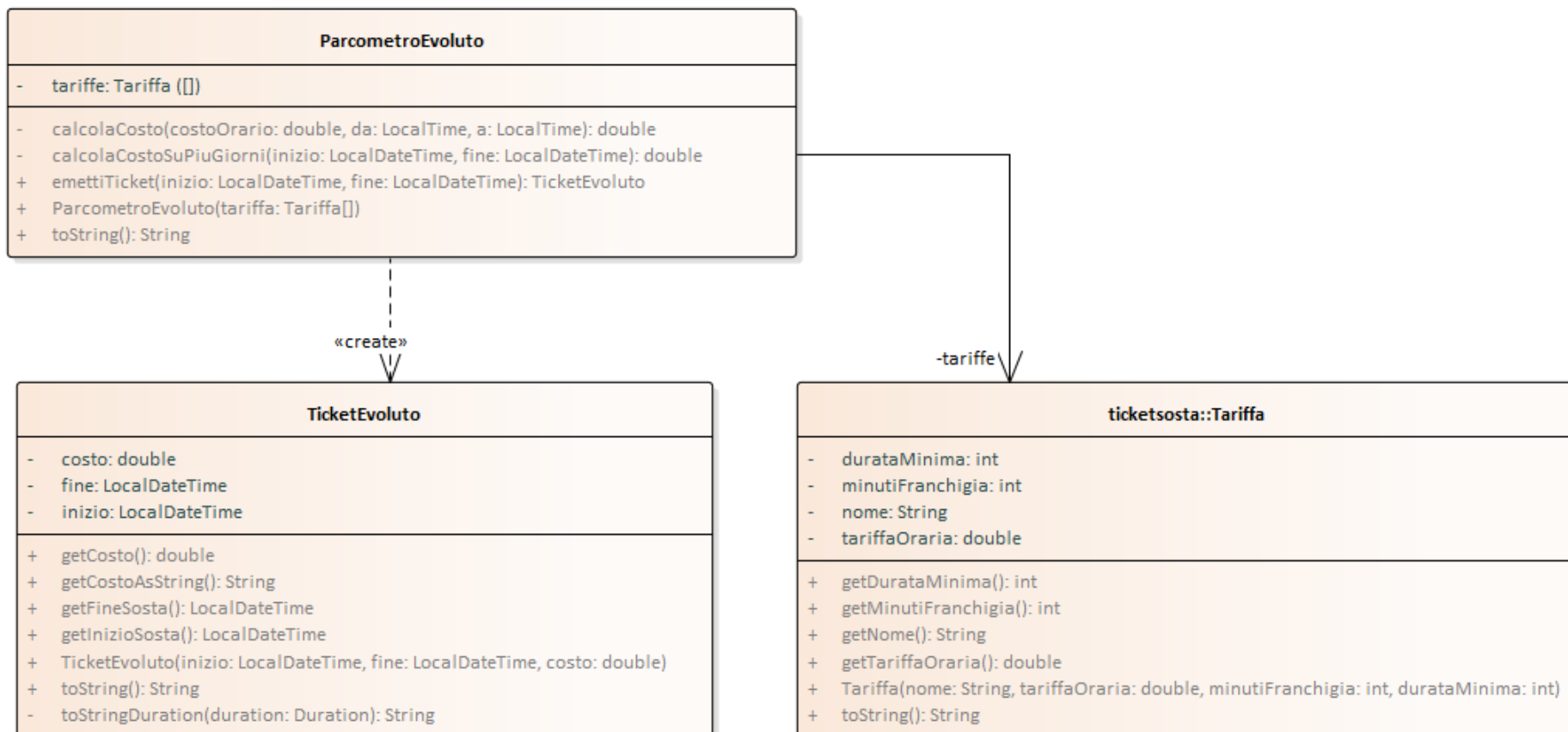
**Attenzione:** anche `toString` va rivista in modo che stampi correttamente tutte le Tariffe



# ParcometroEvoluto: Algoritmo

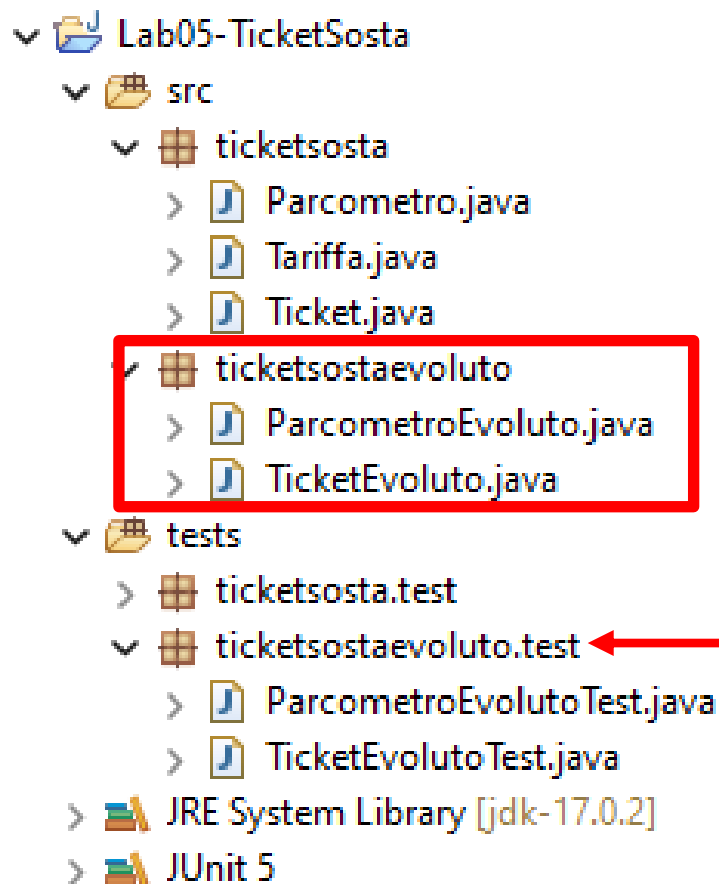
- Caso base: no franchigia, no minimo
  - per ogni giorno di sosta
    - si identifica la tariffa da applicare
    - si calcola la durata della sosta per quel giorno
    - costo += costo orario \* durata (calcolaCosto)
- Caso con franchigia
  - sottrarre franchigia dal primo giorno
  - poi, come nel caso base
- Caso con durata minima
  - se durata inferiore al minimo, considerare il minimo
  - altrimenti come caso base

# Architettura software





# Struttura del progetto

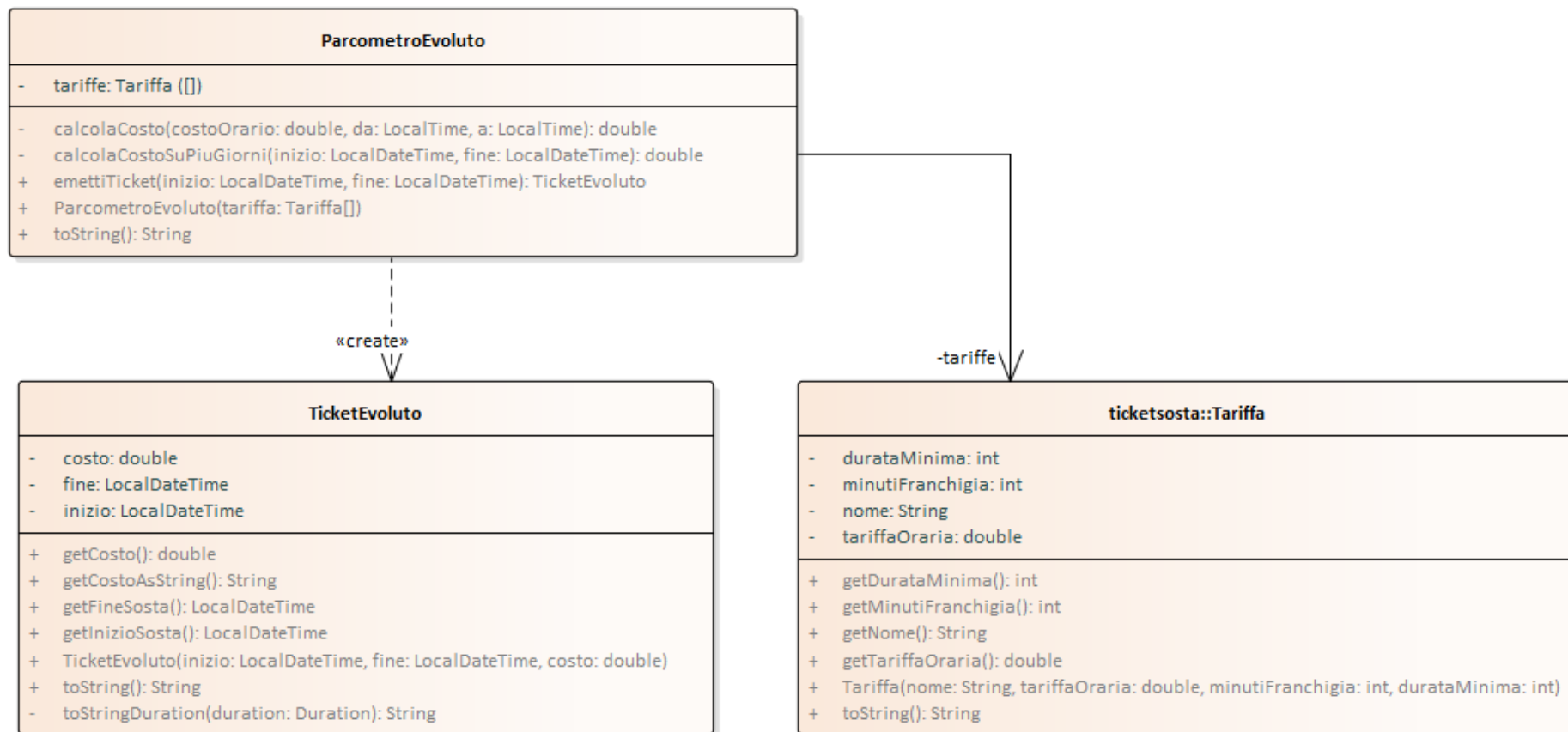


## Aggiungere al progetto

- il package  
**ticketsostaevoluto**
- le due classi da realizzare  
**TicketEvoluto**  
**ParcometroEvoluto**

NB: i test sono già nello Startkit

# Architettura software



**Tempo a disposizione: 60 minuti**

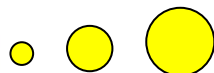
# Considerazioni finali

- Il «cambio dei requisiti» ha avuto un *notevole impatto sul progetto*
  - le ipotesi semplificative avevano *ristretto molto il dominio applicativo* andando ad isolare un «caso particolare» di parcometro
- **MOTIVO: non avevamo operato in ottica «*design for change*»**
  - in realtà, i requisiti non sono mai stabili
    - vengono sempre in mente nuovi requisiti
    - il software evolve continuamente
  - sarebbe bene «*predisporre*» il software per le variazioni future  
→ analizzare e progettare per il «caso generale»,  
non per il «particolare»!

# Considerazioni finali

- Il ParcometroEvoluto sicuramente offre più flessibilità rispetto alla versione base, MA modella ancora un «dominio ristretto»
  - all'interno di una giornata c'è ancora *solo un'unica tariffa* per la sosta  
→ palesemente irrealistico: nel mondo reale non è così!  
In realtà, la tariffa spesso dipende dalla fascia oraria in cui si sosta...
- Per fare le cose per bene, occorrerebbe introdurre il concetto di *fascia oraria*
  - la Tariffa andrebbe legata alla FasciaOraria in cui si applica
  - occorrerebbero controlli per verificare che non ci siano «buchi» nella giornata (non coperti da alcune tariffe)
- Affronteremo questa tematica in una prossima esercitazione 😊

# JUnit & Eclipse



Ho perso JUnit  
nel progetto in  
Eclipse

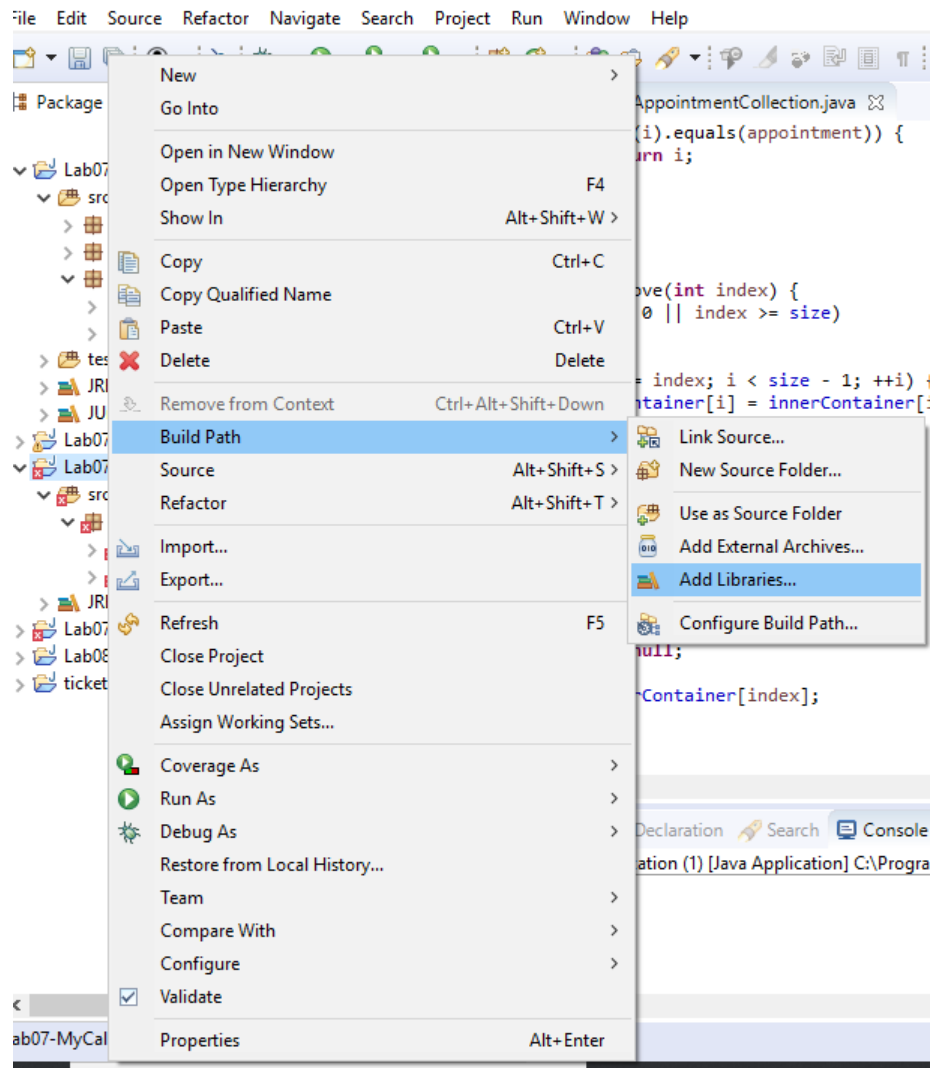
# JUnit & Eclipse

---



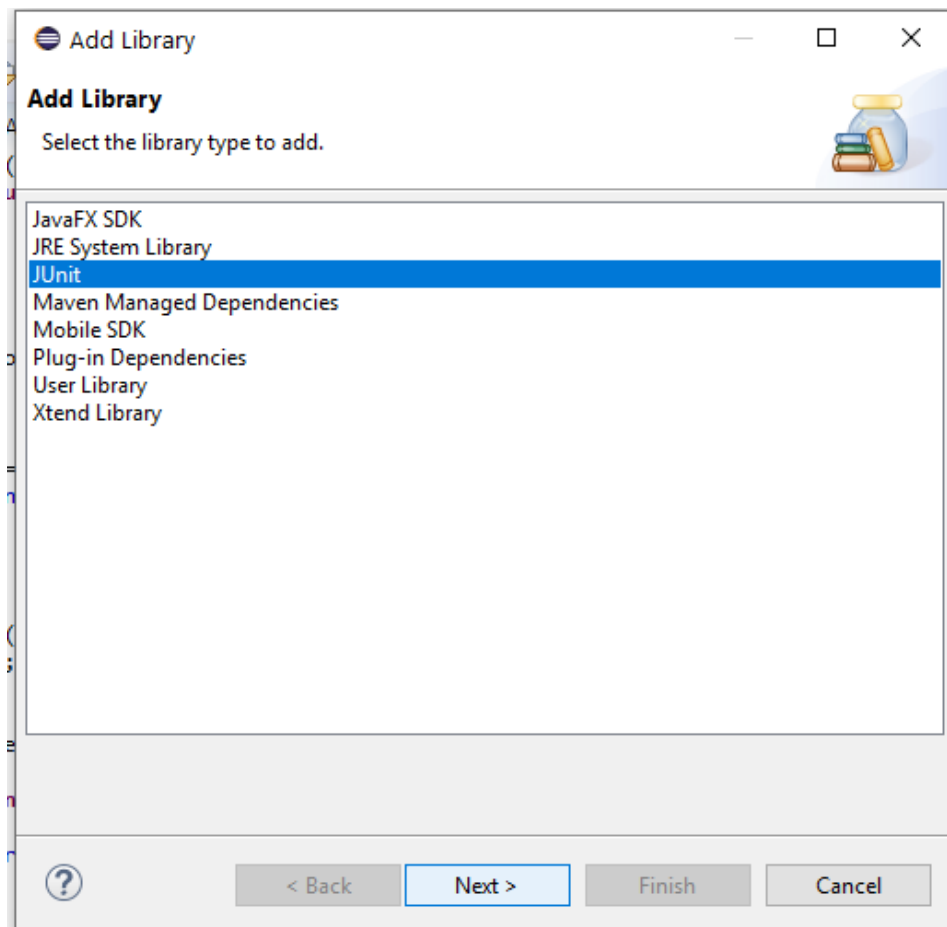
- Bastano pochi passaggi per inserire JUnit nel progetto....

# JUnit & Eclipse



1. Tasto destro sulla cartella del progetto
2. Selezionare «Build Path»
3. Scegliere «Add Libraries»
4. Invio

# JUnit & Eclipse

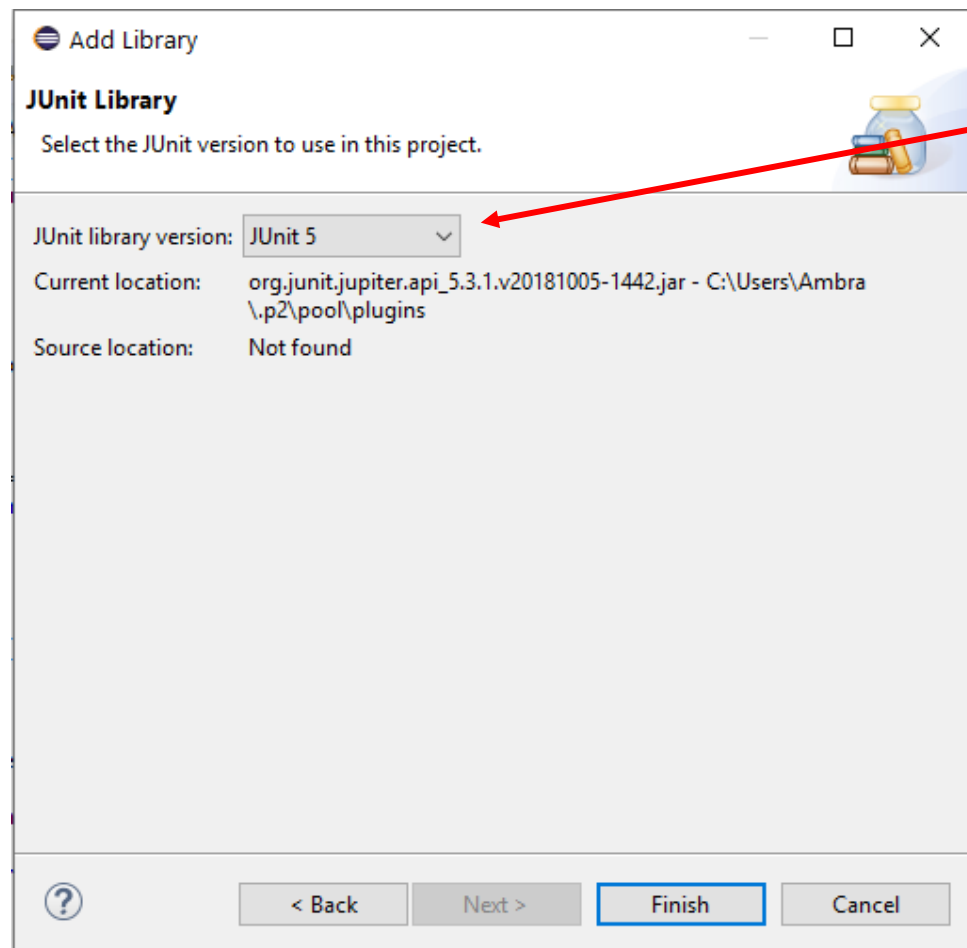


5. Scegliere JUnit

6. Premere «Next»



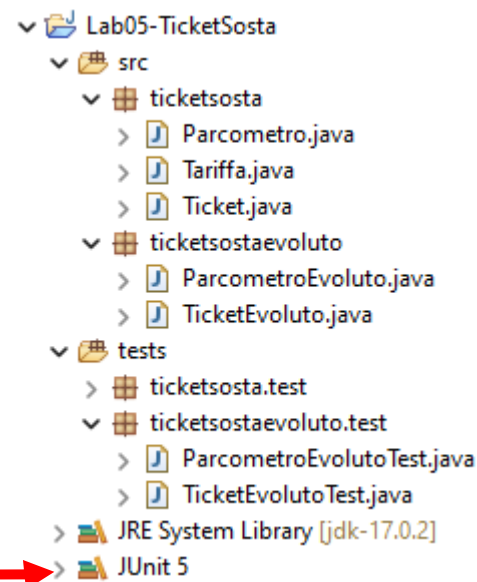
# JUnit & Eclipse



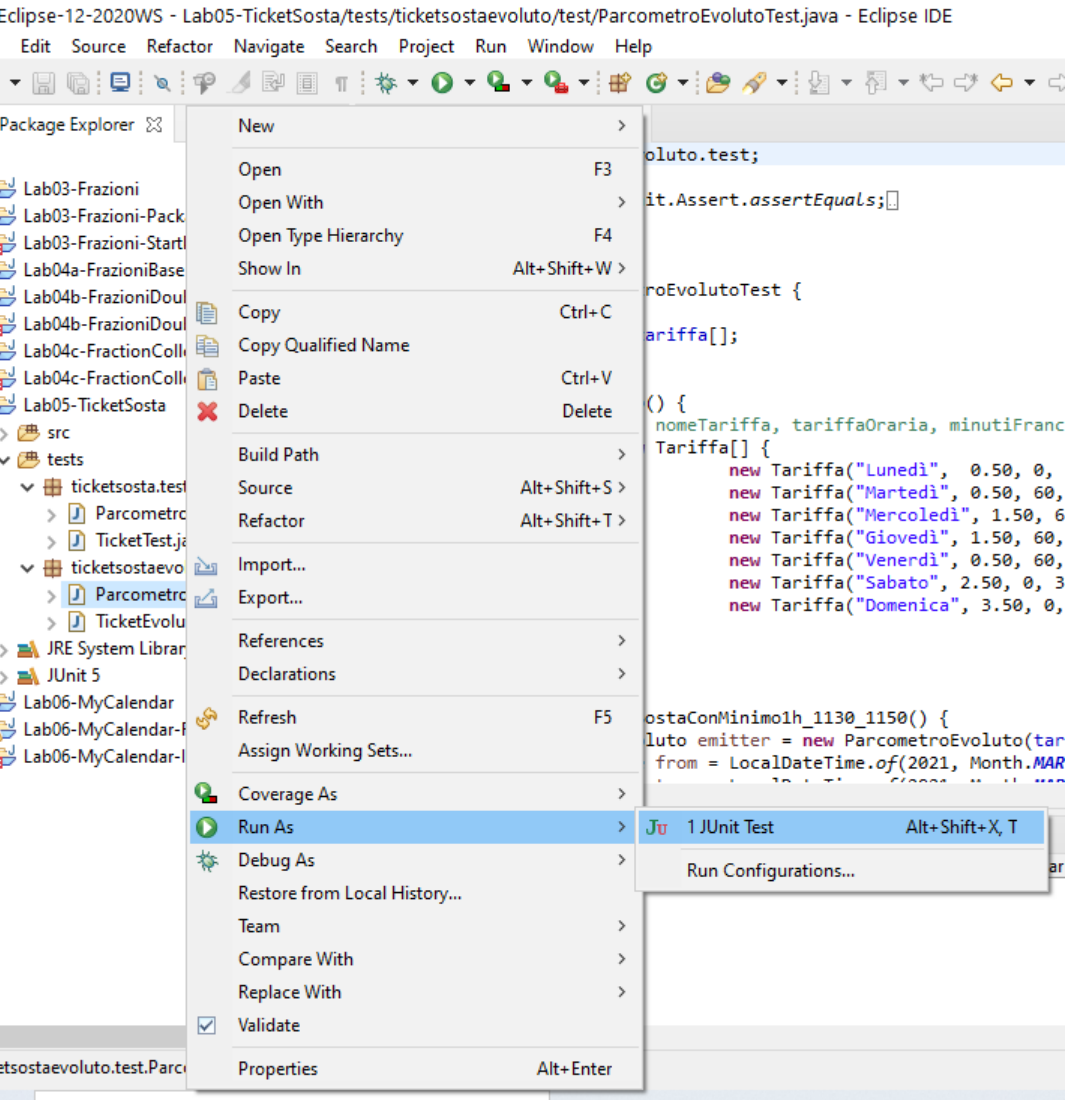
7. Assicuratevi che sia indicato JUnit 5

8. Premere «Finish»

9. La libreria comparirà nel progetto

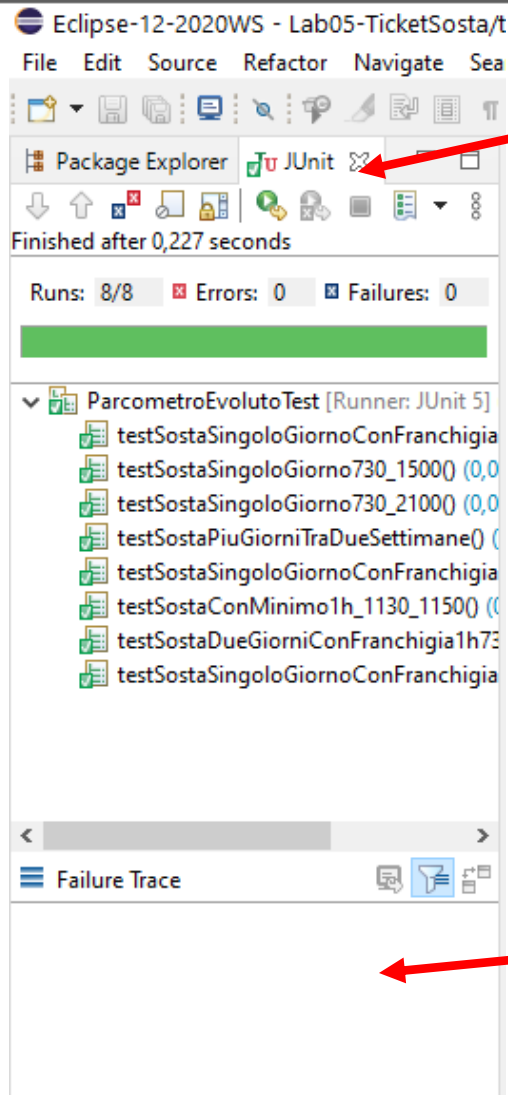


# JUnit & Eclipse – run test



1. Tasto destro sulla classe di test
2. Scegliere «Run As»
3. Scegliere «JUnit Test»
4. Invio

# JUnit & Eclipse – run test



5. Si Apre un nuovo TAB di fianco al Package Explorer
6. Li trovate l'esito dei risultati dei test
7. .. Se è tutto verde è andato tutto bene
8. ...se ci sono test con croce blu significa che delle assert sono state violate
9. .. Se ci sono test con croce rossa significa che i test sono falliti con eccezioni
10. Nel tab Failure Trace trovate il dettaglio di cosa è andato storto